School of Computer Science, Huazhong University of Science and Technology

Laboratory Report on Computer Communications and Networks

Class CS2007 Name Sun Xi Student Number U202015512

sports event	Socket Programming (40%)	Design of protocols for reliable data transmission (20%)	CPT Networking (20%)	Ordinary grades (20%)	totals
score					

Teacher Comments:

Teacher's

signature:

date points

given:

table of contents

table of contents

Experiment 2 Experiment on the design of reliable data transmission protocol **1**

1.1 Environments 1
1.2 System Functional Requirements 1
1.3 System Design 2
1.4 System realization 5
1.5 System Tests and Results Explained 14
1.6 Other issues requiring clarification 18
1.7 References 19

Experiences and Recommendations **20**

2.1 Experiences 20
2.2 Recommendations 20

Experiment 2 Experiment on the design of reliable data transmission protocol

1.1 Environment

1.1.1 Development platform

Operating System: Microsoft Windows 10
Processor: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
IDE: Microsoft Visual Studio Community 2019
third-party dependencies: simulated network environment netsimlib.lib

1.1.2 Operational platforms

Operating System: Microsoft Windows 10
Processor: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
IDE: Microsoft Visual Studio Community 2019
third-party dependencies: simulated network environment netsimlib.lib

1.2 System functional requirements

Reliable Transport Layer protocols consider only unidirectional transmission, i.e., only the sender generates a data message and the receiver only receives the message and gives an acknowledgement. The implementation of a specific protocol requires the specification of the number of binary digits (e.g., 3-bit binary encoded message number) and the window size (e.g., size 4) encode the message segment number, and the message segment number must be encoded according to the specified number of binary digits. The code implementation does not need to be based on the Socket API, does not need to utilize multithreading, and does not need any UI interface.

There are three main reliable transport layer protocols that need to be implemented:

1. Implementing a GBN-based reliable transport protocol is worth 50%.

2. Implementing a reliable SR-based transport protocol is worth 30%.
3. Implement a simplified version of the TCP protocol, 20%. Based on the GBN protocol, this part needs to add the function of supporting fast retransmission and timeout retransmission according to the reliable data transmission mechanism of TCP. Segment numbers should be numbered by segment, with the acknowledgement number being the last segment number received. Flow control and congestion control do not need to be considered.

1.3 system design

Overall design of reliable transmission protocols:

Design the format of the sender and receiver message segments, each of which needs to contain information such as the serial number, checksum, and so on.

To implement the sender's process of sending message segments, the sender needs to take into account the window size limitation when sending message segments, so that only message segments within the window range can be sent. The sender also needs to turn on a timer as a basis for retransmission.

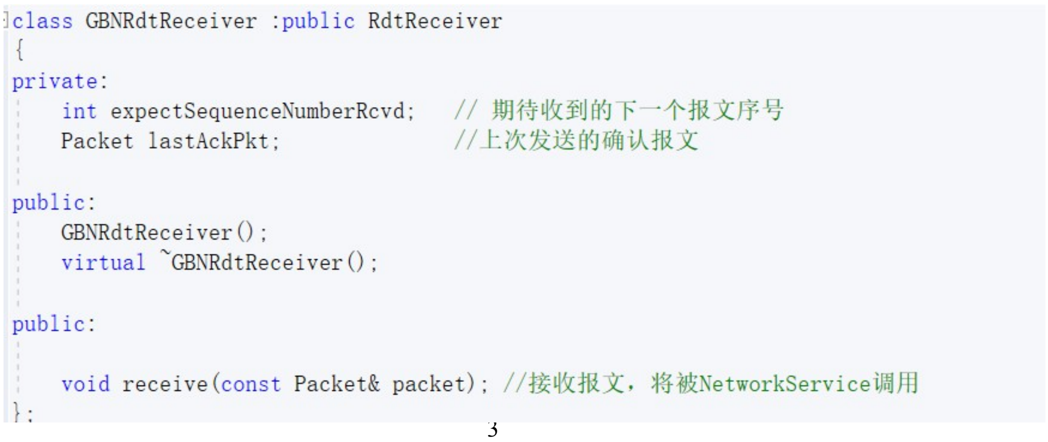
The process of receiving a message segment and sending an acknowledgement message is implemented for the receiver. After receiving the message segment, the receiver needs to verify that the checksum is correct, send an acknowledgement message if it is correct, and store the message segment in a buffer. The receiver also needs to start a timer, and if no new segment is received before the timer expires, the receiver needs to send an acknowledgement of the requested segment.

A simulated network environment is created between the sender and receiver, and functional testing is performed using the simulated network environment.

1.3.1 GBN

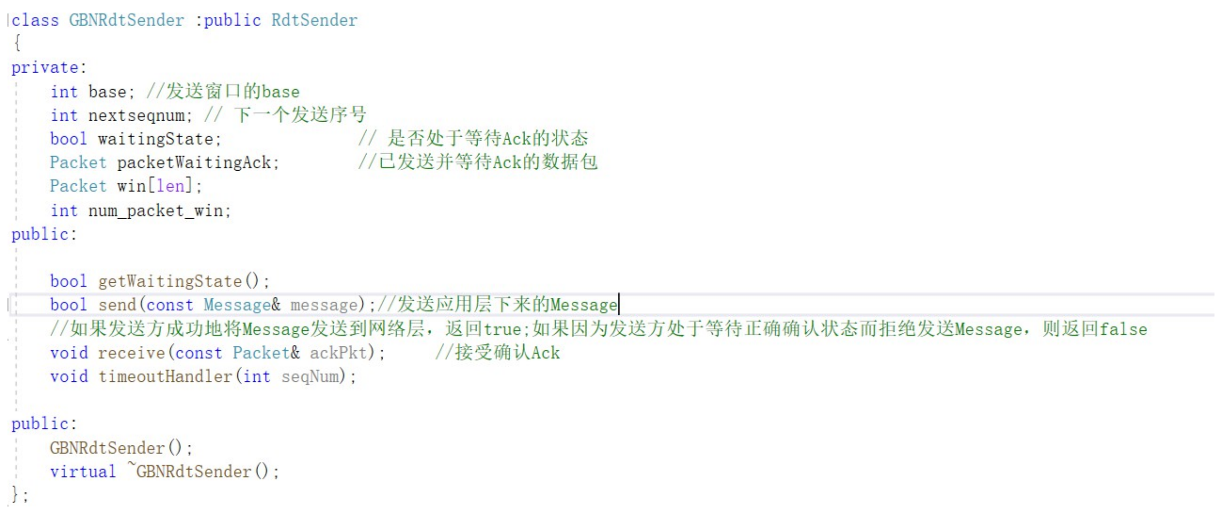
System design for GBN-based reliable transmission protocols:

The transmit window size is N, and the receive window size is 1. The sender sends successive frames in the specified window size, and the receiver process keeps track of the sequence number of the next frame it expects to receive, and includes that number in each acknowledgement (ACK) it sends. The receiver will discard any frame that does not have the exact sequence number it expects (i.e., duplicate frames that have already been acknowledged, or out-of-order frames that it expects to receive later) and will retransmit the last correct ACK.The GBNReceiver and GBNSender designs are shown in Figure 1 and Figure 2:



```
class GBNRdtReceiver :public RdtReceiver
{
private:
    int expectSequenceNumberRcvd;   // 期待收到的下一个报文序号
    Packet lastAckPkt;              //上次发送的确认报文

public:
    GBNRdtReceiver();
    virtual ~GBNRdtReceiver();

public:

    void receive(const Packet& packet); //接收报文，将被NetworkService调用
};
```

Figure 1 GBNReceiver Design

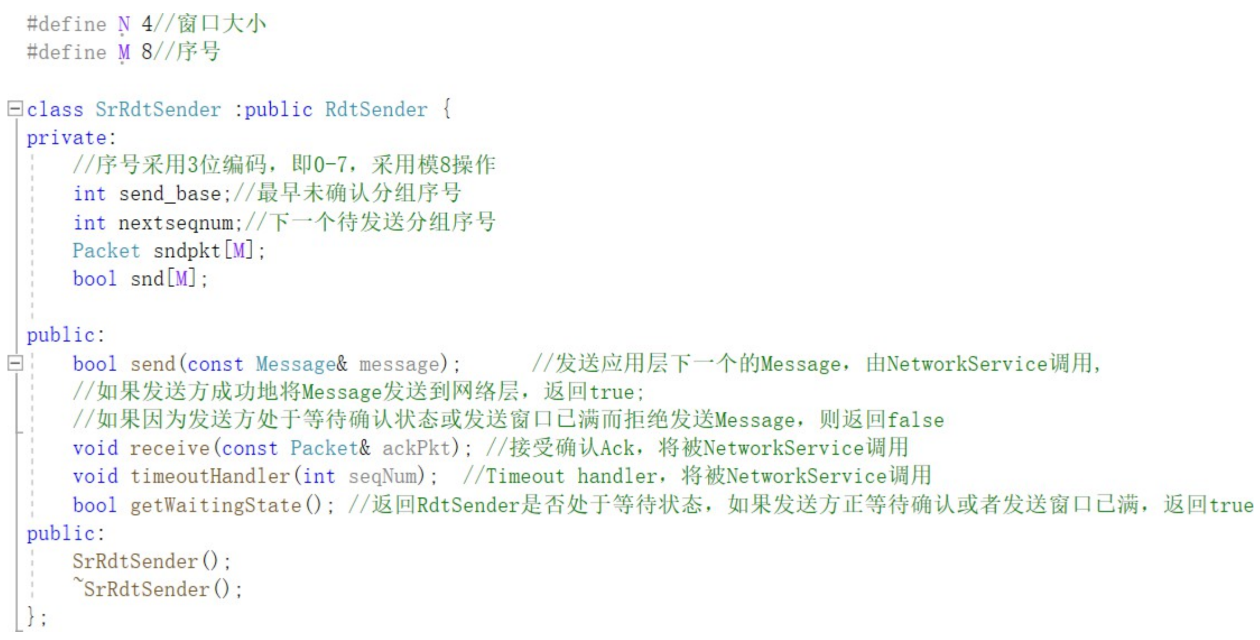


```
class GBNRdtSender :public RdtSender
{
private:
    int base; //发送窗口的base
    int nextseqnum; // 下一个发送序号
    bool waitingState;              // 是否处于等待Ack的状态
    Packet packetWaitingAck;        //已发送并等待Ack的数据包
    Packet win[len];
    int num_packet_win;
public:

    bool getWaitingState();
    bool send(const Message& message);//发送应用层下来的Message
    //如果发送方成功地将Message发送到网络层，返回true;如果因为发送方处于等待正确确认状态而拒绝发送Message，则返回false
    void receive(const Packet& ackPkt);      //接受确认Ack
    void timeoutHandler(int seqNum);

public:
    GBNRdtSender();
    virtual ~GBNRdtSender();
};
```

Figure 2 **GBNSender** Design

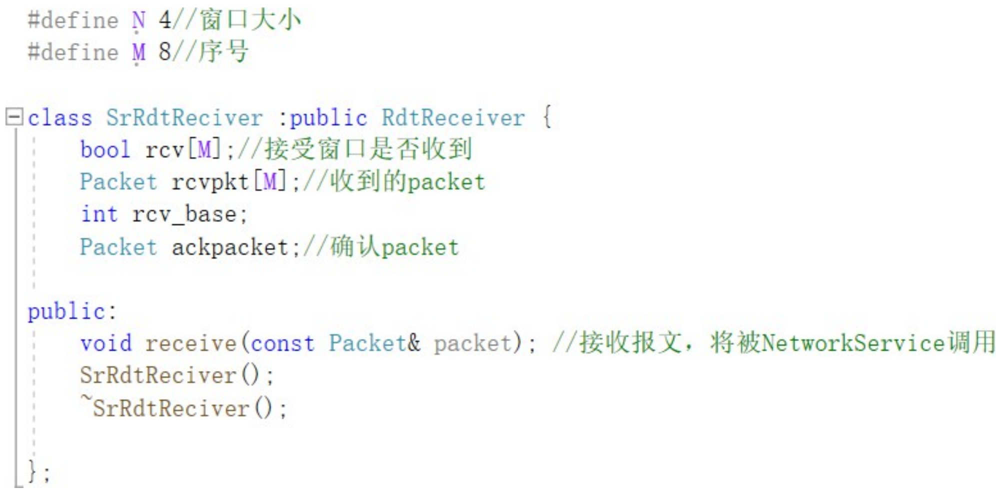
1.3.2 SR

Both the send and receive window sizes are greater than 1, and the sending process continues to send multiple frames specified by the window size even after a frame is lost.The SRSender and SRReciever designs are shown in Figures 3 and 4:



```
#define N 4//窗口大小
#define M 8//序号

class SrRdtSender :public RdtSender {
private:
    //序号采用3位编码，即0-7，采用模8操作
    int send_base;//最早未确认分组序号
    int nextseqnum;//下一个待发送分组序号
    Packet sndpkt[M];
    bool snd[M];

public:
    bool send(const Message& message);      //发送应用层下一个的Message，由NetworkService调用,
    //如果发送方成功地将Message发送到网络层，返回true;
    //如果因为发送方处于等待确认状态或发送窗口已满而拒绝发送Message，则返回false
    void receive(const Packet& ackPkt); //接受确认Ack，将被NetworkService调用
    void timeoutHandler(int seqNum);  //Timeout handler，将被NetworkService调用
    bool getWaitingState(); //返回RdtSender是否处于等待状态，如果发送方正等待确认或者发送窗口已满，返回true
public:
    SrRdtSender();
    ~SrRdtSender();
};
```

Figure 3 **SRSender** Design

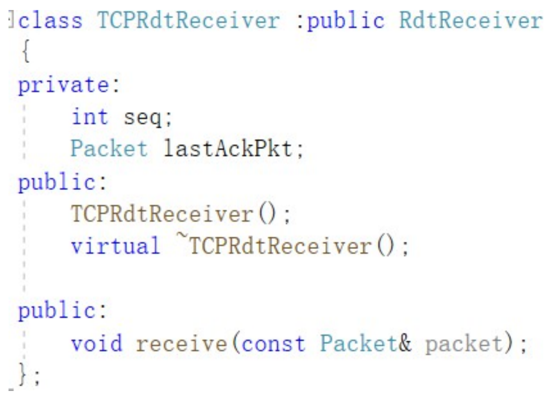


```
#define N 4//窗口大小
#define M 8//序号

class SrRdtReciver :public RdtReceiver {
    bool rcv[M];//接受窗口是否收到
    Packet rcvpkt[M];//收到的packet
    int rcv_base;
    Packet ackpacket;//确认packet

public:
    void receive(const Packet& packet); //接收报文，将被NetworkService调用
    SrRdtReciver();
    ~SrRdtReciver();

};
```

Figure 4 **SRReceiver** Design

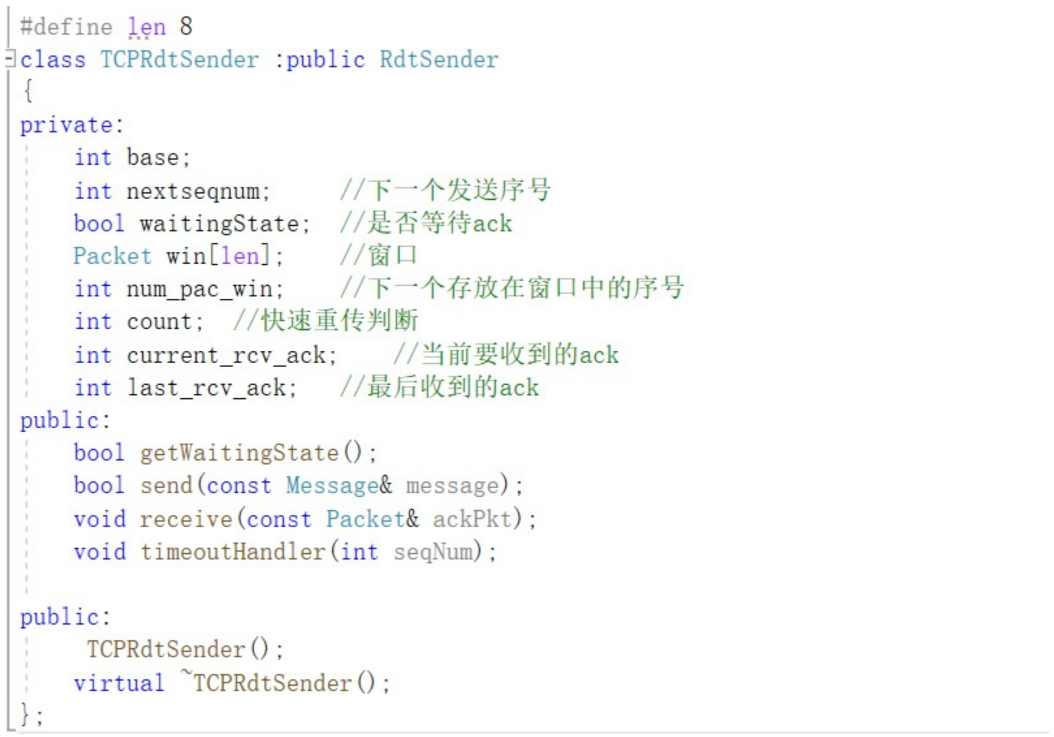
1.3.3 TCP

Combining the two reliable transport protocols, GBN and SR, the TCP sender needs to maintain only the least-squares number of the sent but unacknowledged packet and the sequence number of the next byte to be sent. However, TCP uses selective retransmission rather than window length to limit the number of unfinished, unacknowledged packets in the pipeline. The sender retransmits only the unacknowledged least-serial packets, not all packets after the least-serial number, as is the case with the GBN protocol. tcpReceiver and tcpsender are shown in Figure 5 and Figure 6:



```
class TCPRdtReceiver :public RdtReceiver
{
private:
    int seq;
    Packet lastAckPkt;
public:
    TCPRdtReceiver();
    virtual ~TCPRdtReceiver();

public:
    void receive(const Packet& packet);
};
```

Figure 5 **TCPReceiver** Design



```
#define len 8
class TCPRdtSender :public RdtSender
{
private:
    int base;
    int nextseqnum;     //下一个发送序号
    bool waitingState;  //是否等待ack
    Packet win[len];    //窗口
    int num_pac_win;    //下一个存放在窗口中的序号
    int count;  //快速重传判断
    int current_rcv_ack;    //当前要收到的ack
    int last_rcv_ack;   //最后收到的ack
public:
    bool getWaitingState();
    bool send(const Message& message);
    void receive(const Packet& ackPkt);
    void timeoutHandler(int seqNum);

public:
     TCPRdtSender();
    virtual ~TCPRdtSender();
};
```

Figure 6 TCPSender Design

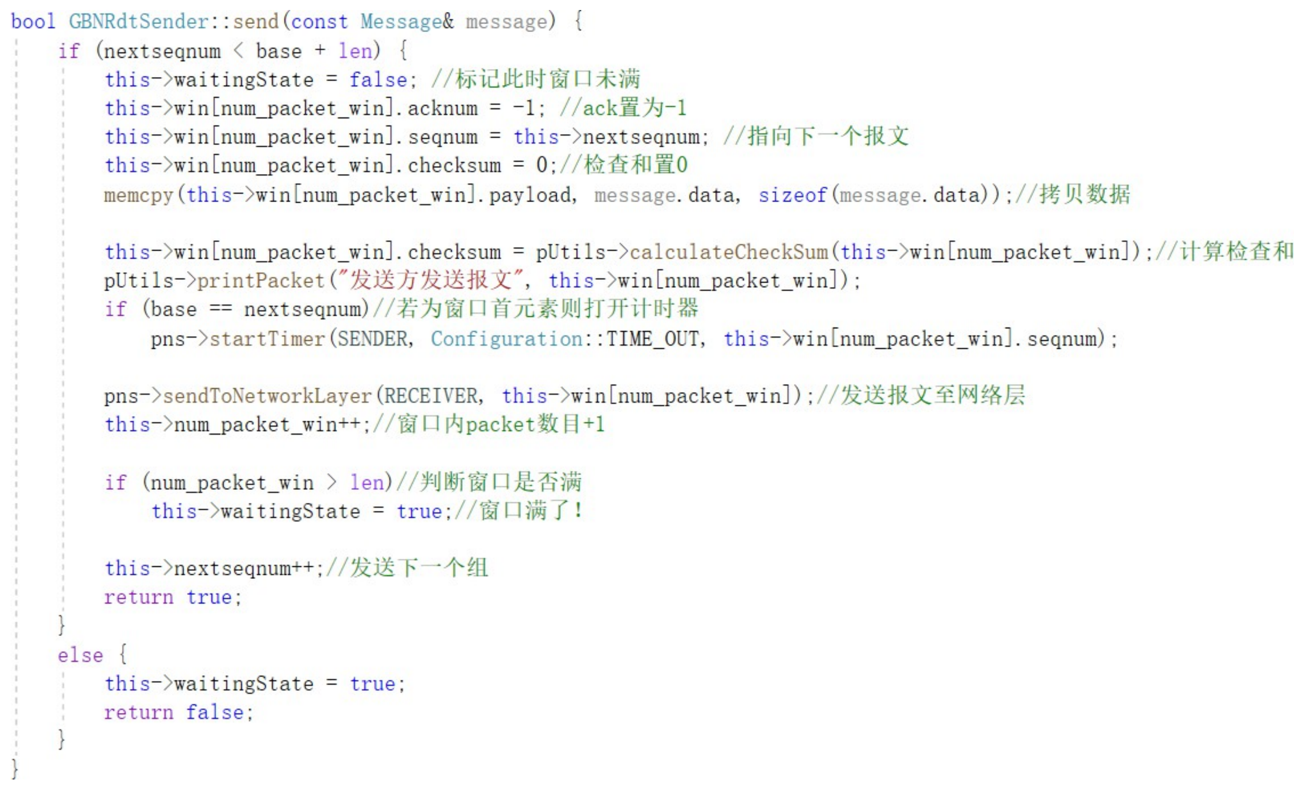
1.4 system implementation

1.4.1 GBN

(1) GBNSender

The sender maintains a window size, sets up a loop to send packets, determines if nextSeq is in the window, sends a whole window of packets at once and waits for the ACK corresponding to the sequence number to be returned. According to the ACK number returned by the receiving end, adjust the value of the window base to ACK+1. If the maximum value of the window is reached, close the timer, if not, set the timer to continue waiting. If it is not, set the timer to continue waiting. If it times out, the sender did not receive the expected ACK within the time limit, and will resend the sequence number and all packets after it.

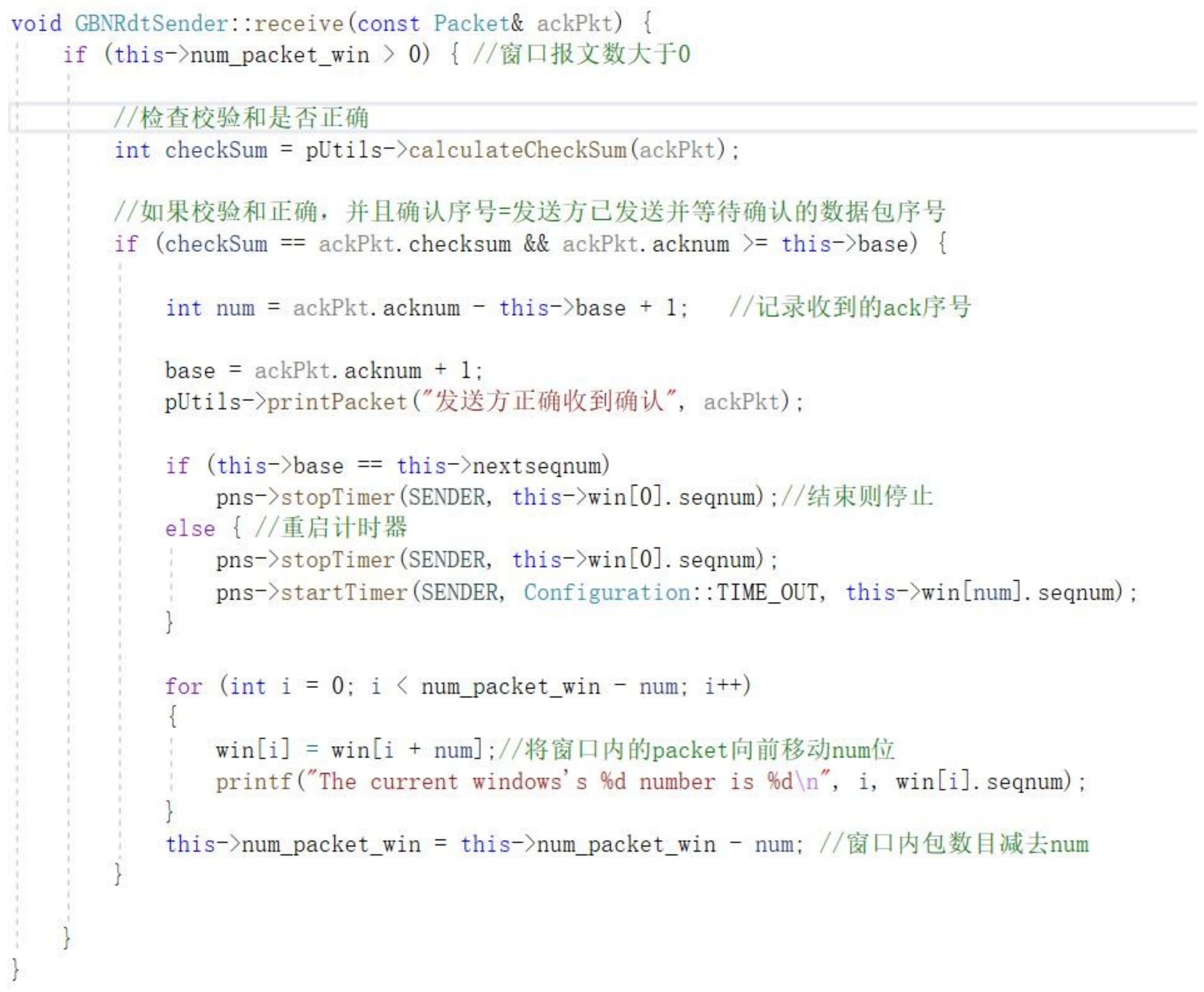
(a) The send method is implemented as shown in Figure 7:



```
bool GBNRdtSender::send(const Message& message) {
    if (nextseqnum < base + len) {
        this->waitingState = false; //标记此时窗口未满
        this->win[num_packet_win].acknum = -1; //ack置为-1
        this->win[num_packet_win].seqnum = this->nextseqnum; //指向下一个报文
        this->win[num_packet_win].checksum = 0;//检查和置0
        memcpy(this->win[num_packet_win].payload, message.data, sizeof(message.data));//拷贝数据

        this->win[num_packet_win].checksum = pUtils->calculateCheckSum(this->win[num_packet_win]);//计算检查和
        pUtils->printPacket("发送方发送报文", this->win[num_packet_win]);
        if (base == nextseqnum)//若为窗口首元素则打开计时器
            pns->startTimer(SENDER, Configuration::TIME_OUT, this->win[num_packet_win].seqnum);

        pns->sendToNetworkLayer(RECEIVER, this->win[num_packet_win]);//发送报文至网络层
        this->num_packet_win++;//窗口内packet数目+1

        if (num_packet_win > len)//判断窗口是否满
            this->waitingState = true;//窗口满了！

        this->nextseqnum++;//发送下一个组
        return true;
    }
    else {
        this->waitingState = true;
        return false;
    }
}
```

Figure 7 GBNSender::send implementation

(b) The receive method is implemented as shown in Figure 8:



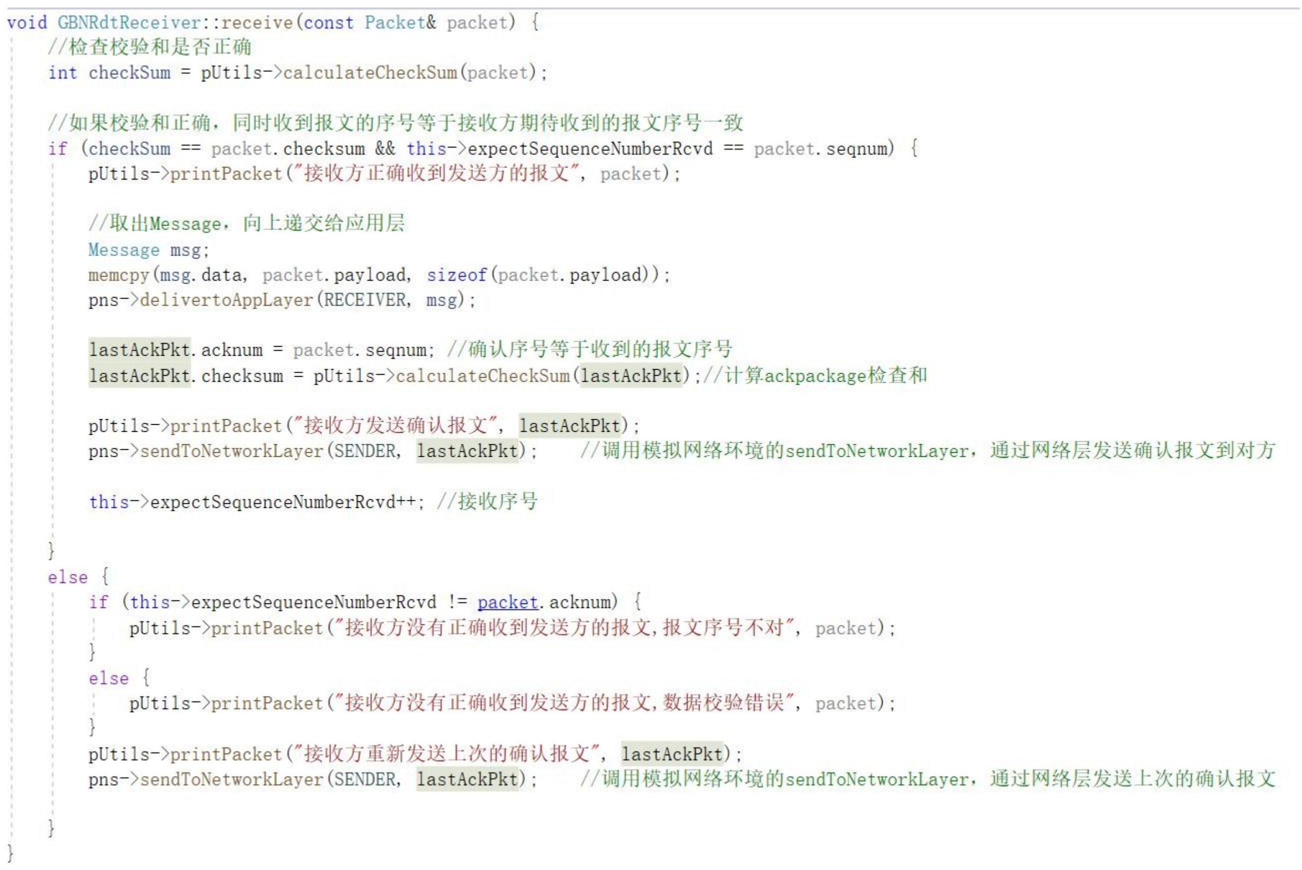
```
void GBNRdtSender::receive(const Packet& ackPkt) {
    if (this->num_packet_win > 0) { //窗口报文数大于0

        //检查校验和是否正确
        int checkSum = pUtils->calculateCheckSum(ackPkt);

        //如果校验和正确，并且确认序号=发送方已发送并等待确认的数据包序号
        if (checkSum == ackPkt.checksum && ackPkt.acknum >= this->base) {

            int num = ackPkt.acknum - this->base + 1;   //记录收到的ack序号

            base = ackPkt.acknum + 1;
            pUtils->printPacket("发送方正确收到确认", ackPkt);

            if (this->base == this->nextseqnum)
                pns->stopTimer(SENDER, this->win[0].seqnum);//结束则停止
            else { //重启计时器
                pns->stopTimer(SENDER, this->win[0].seqnum);
                pns->startTimer(SENDER, Configuration::TIME_OUT, this->win[num].seqnum);
            }

            for (int i = 0; i < num_packet_win - num; i++)
            {
                win[i] = win[i + num];//将窗口内的packet向前移动num位
                printf("The current windows's %d number is %d\n", i, win[i].seqnum);
            }
            this->num_packet_win = this->num_packet_win - num; //窗口内包数目减去num
        }

    }
}
```

Figure 8 GBNSender::recieve implementation

（2） GBNReceiver

Whenever a grouping of expected sequence numbers is received, leave the grouping and increment the expected increment by 1, then return the sequence number of the ACK; if not, simply discard and return the ACK of the previous sequence number.

(a) The receive method is shown in Figure 9:



```
void GBNRdtReceiver::receive(const Packet& packet) {
    //检查校验和是否正确
    int checkSum = pUtils->calculateCheckSum(packet);

    //如果校验和正确，同时收到报文的序号等于接收方期待收到的报文序号一致
    if (checkSum == packet.checksum && this->expectSequenceNumberRcvd == packet.seqnum) {
        pUtils->printPacket("接收方正确收到发送方的报文", packet);

        //取出Message，向上递交给应用层
        Message msg;
        memcpy(msg.data, packet.payload, sizeof(packet.payload));
        pns->delivertoAppLayer(RECEIVER, msg);

        lastAckPkt.acknum = packet.seqnum; //确认序号等于收到的报文序号
        lastAckPkt.checksum = pUtils->calculateCheckSum(lastAckPkt);//计算ackpackage检查和

        pUtils->printPacket("接收方发送确认报文", lastAckPkt);
        pns->sendToNetworkLayer(SENDER, lastAckPkt);    //调用模拟网络环境的sendToNetworkLayer，通过网络层发送确认报文到对方

        this->expectSequenceNumberRcvd++; //接收序号

    }
    else {
        if (this->expectSequenceNumberRcvd != packet.acknum) {
            pUtils->printPacket("接收方没有正确收到发送方的报文,报文序号不对", packet);
        }
        else {
            pUtils->printPacket("接收方没有正确收到发送方的报文,数据校验错误", packet);
        }
        pUtils->printPacket("接收方重新发送上次的确认报文", lastAckPkt);
        pns->sendToNetworkLayer(SENDER, lastAckPkt);    //调用模拟网络环境的sendToNetworkLayer，通过网络层发送上次的确认报文

    }
}
```

Figure 9 GBNReceiver::receive implementation

1.4.2 SR

The send and receive windows are equal in size and half the maximum sequence number.

（1） SRSender

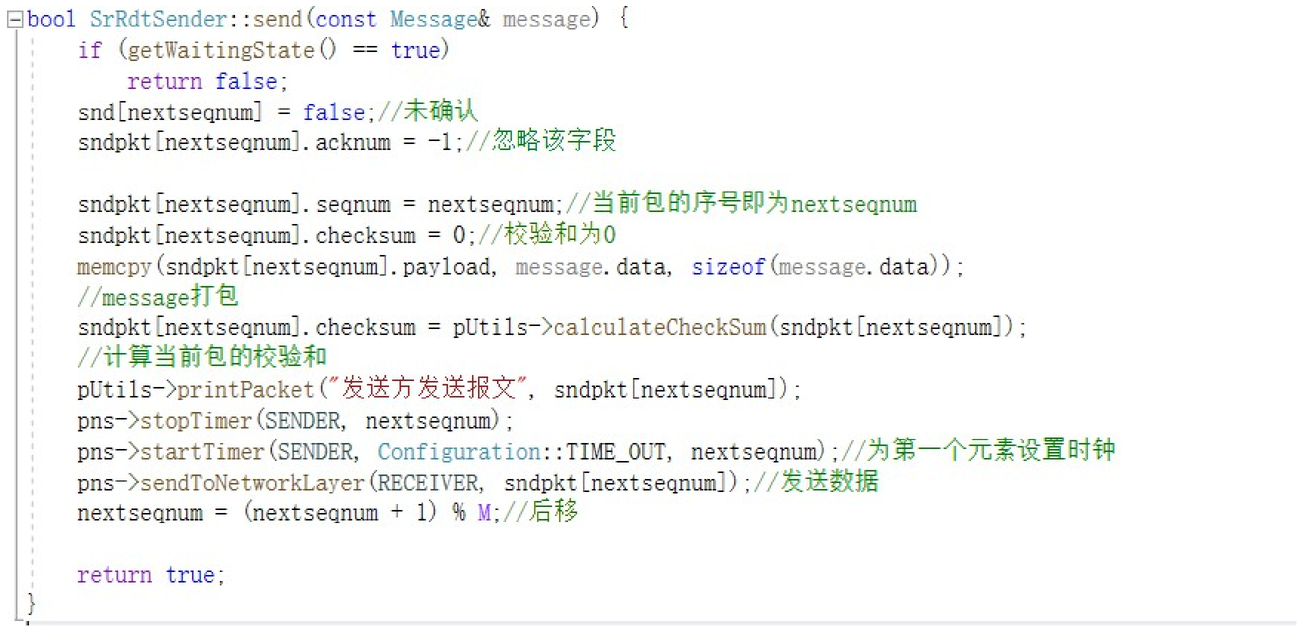
Events and actions on the sender side:

• Data is received from the upper layer. When data is received from the upper layer, the SR sender checks for the next available serial number for that packet. If the serial number is within the sender's window, the data is packetized and sent; otherwise, as in GBN, the data is either cached or returned to the upper layer for later transmission.

• Timeout. Timers are once again being used to prevent lost packets. However, each packet must now have its own logical timer, since only one packet can be sent after the timeout occurs.

• If an ACK is received, the SR sender marks the acknowledged packet as received if the packet's serial number is within the window. If the packet's serial number is equal to send_base, the window base serial number moves forward to the unacknowledged packet with the smallest serial number. If the window moves and there are unsent packets whose serial numbers fall within the window, those packets are sent.

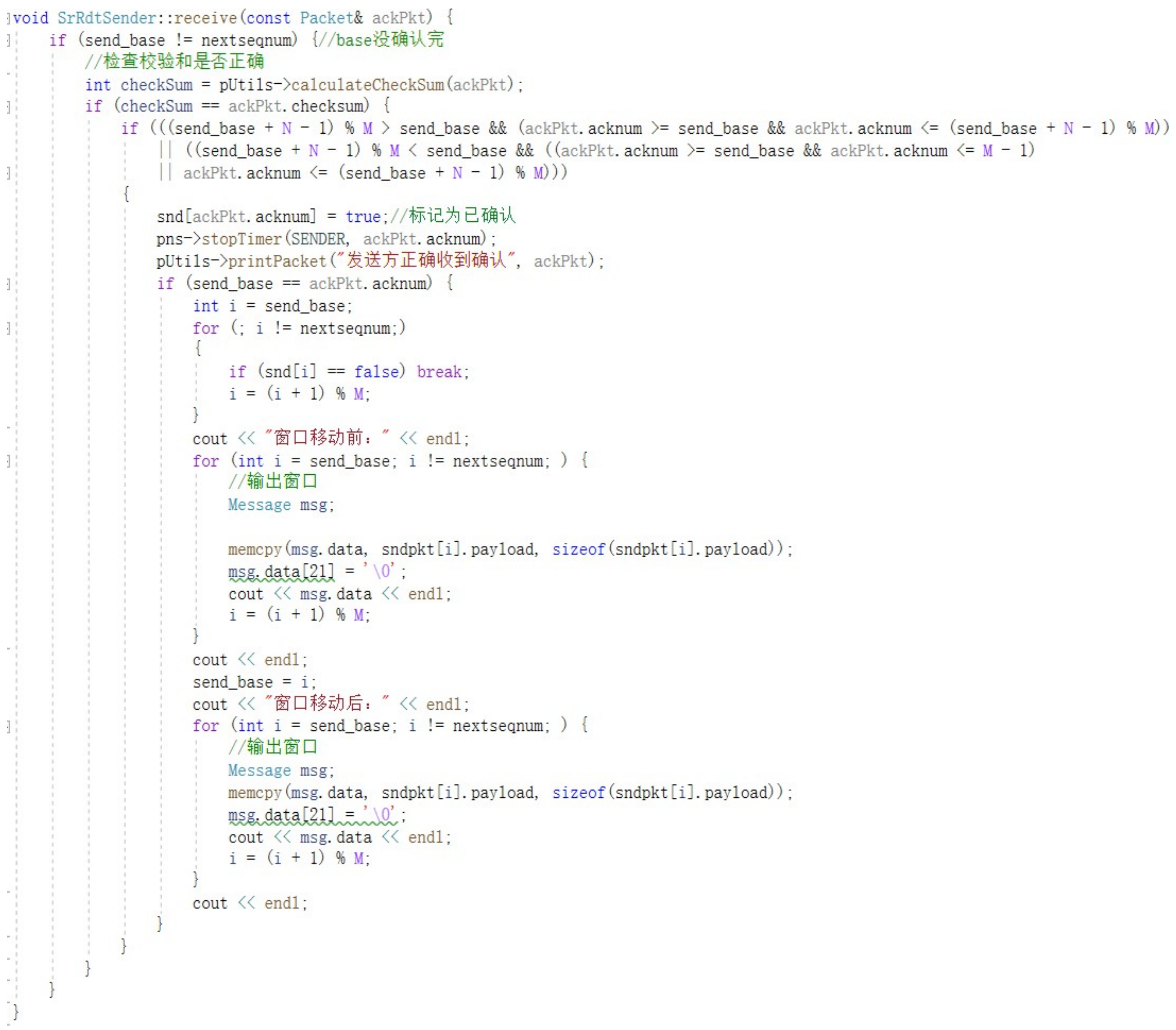
(a) The send method is implemented as shown in Figure 10:



```
bool SrRdtSender::send(const Message& message) {
    if (getWaitingState() == true)
        return false;
    snd[nextseqnum] = false;//未确认
    sndpkt[nextseqnum].acknum = -1;//忽略该字段

    sndpkt[nextseqnum].seqnum = nextseqnum;//当前包的序号即为nextseqnum
    sndpkt[nextseqnum].checksum = 0;//校验和为0
    memcpy(sndpkt[nextseqnum].payload, message.data, sizeof(message.data));
    //message打包
    sndpkt[nextseqnum].checksum = pUtils->calculateCheckSum(sndpkt[nextseqnum]);
    //计算当前包的校验和
    pUtils->printPacket("发送方发送报文", sndpkt[nextseqnum]);
    pns->stopTimer(SENDER, nextseqnum);
    pns->startTimer(SENDER, Configuration::TIME_OUT, nextseqnum);//为第一个元素设置时钟
    pns->sendToNetworkLayer(RECEIVER, sndpkt[nextseqnum]);//发送数据
    nextseqnum = (nextseqnum + 1) % M;//后移

    return true;
}
```

Figure 10 SRSender::send implementation

(b) The receive method is implemented as shown in Figure 11:



```
void SrRdtSender::receive(const Packet& ackPkt) {
    if (send_base != nextseqnum) {//base没确认完
        //检查校验和是否正确
        int checkSum = pUtils->calculateCheckSum(ackPkt);
        if (checkSum == ackPkt.checksum) {
            if (((send_base + N - 1) % M > send_base && (ackPkt.acknum >= send_base && ackPkt.acknum <= (send_base + N - 1) % M))
                || ((send_base + N - 1) % M < send_base && ((ackPkt.acknum >= send_base && ackPkt.acknum <= M - 1)
                || ackPkt.acknum <= (send_base + N - 1) % M)))
            {
                snd[ackPkt.acknum] = true;//标记为已确认
                pns->stopTimer(SENDER, ackPkt.acknum);
                pUtils->printPacket("发送方正确收到确认", ackPkt);
                if (send_base == ackPkt.acknum) {
                    int i = send_base;
                    for (; i != nextseqnum;)
                    {
                        if (snd[i] == false) break;
                        i = (i + 1) % M;
                    }
                    cout << "窗口移动前：" << endl;
                    for (int i = send_base; i != nextseqnum; ) {
                        //输出窗口
                        Message msg;

                        memcpy(msg.data, sndpkt[i].payload, sizeof(sndpkt[i].payload));
                        msg.data[21] = '\0';
                        cout << msg.data << endl;
                        i = (i + 1) % M;
                    }
                    cout << endl;
                    send_base = i;
                    cout << "窗口移动后：" << endl;
                    for (int i = send_base; i != nextseqnum; ) {
                        //输出窗口
                        Message msg;
                        memcpy(msg.data, sndpkt[i].payload, sizeof(sndpkt[i].payload));
                        msg.data[21] = '\0';
                        cout << msg.data << endl;
                        i = (i + 1) % M;
                    }
                    cout << endl;
                }
            }
        }
    }
}
```

Figure 11 SRSender::receive implementation

(2) SRReceiver

Receiver events and actions:

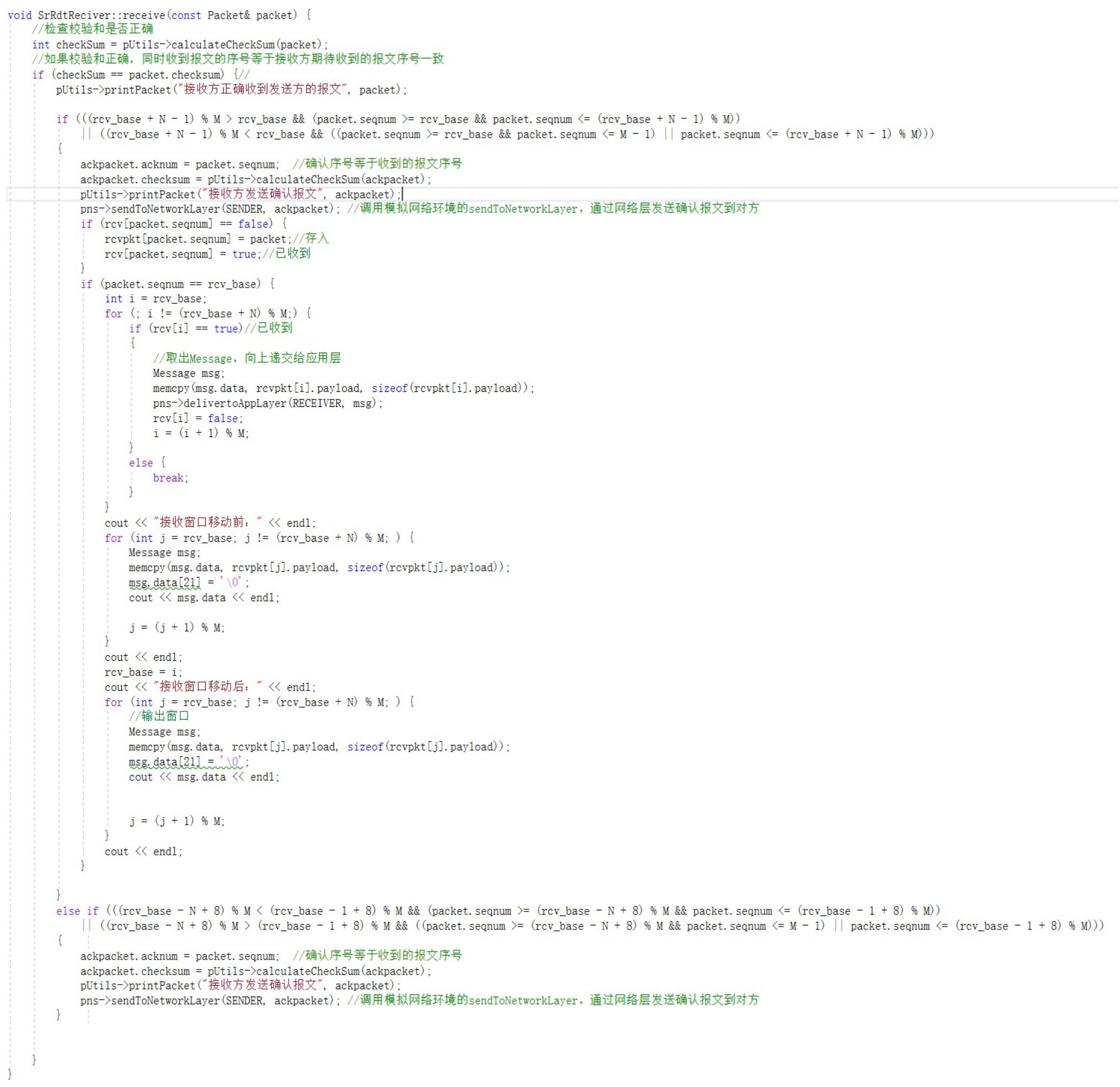
• Packets with serial numbers within [rcv_base, rcv_base+N-1] are received correctly. In this case, the received packet falls

Within the receiver's window, a select ACK is sent back to the sender. If the packet was previously not received, the packet is cached. If the packet's sequential number is equal to the receiver's base sequential number (rcv_base) that packet, as well as previously cached sequential (starting at rcv_base) packets, are delivered to the upper layer. The receive window then delivers these packets upward by the number of the forward-moving packet.

• The packet with serial number within [rcv_base-N, rcv_base-1] was received correctly. In this case, an ACK MUST be generated, even if the packet is one that the receiver has previously acknowledged.

- Other circumstances. Ignore the grouping.

(a) The receive method is shown in Figure 12:



```
void SrRdtReciver::receive(const Packet& packet) {
    //检查校验和是否正确
    int checkSum = pUtils->calculateCheckSum(packet);
    //如果校验和正确，同时收到报文的序号等于接收方期待收到的报文序号一致
    if (checkSum == packet.checksum) {//
        pUtils->printPacket("接收方正确收到发送方的报文", packet);

        if (((rcv_base + N - 1) % M > rcv_base && (packet.seqnum >= rcv_base && packet.seqnum <= (rcv_base + N - 1) % M))
            || ((rcv_base + N - 1) % M < rcv_base && ((packet.seqnum >= rcv_base && packet.seqnum <= M - 1) || packet.seqnum <= (rcv_base + N - 1) % M)))
        {
            ackpacket.acknum = packet.seqnum;  //确认序号等于收到的报文序号
            ackpacket.checksum = pUtils->calculateCheckSum(ackpacket);
            pUtils->printPacket("接收方发送确认报文", ackpacket);
            pns->sendToNetworkLayer(SENDER, ackpacket); //调用模拟网络环境的sendToNetworkLayer，通过网络层发送确认报文到对方
            if (rcv[packet.seqnum] == false) {
                rcvpkt[packet.seqnum] = packet;//存入
                rcv[packet.seqnum] = true;//已收到
            }
            if (packet.seqnum == rcv_base) {
                int i = rcv_base;
                for (; i != (rcv_base + N) % M;) {
                    if (rcv[i] == true)//已收到
                    {
                        //取出Message，向上递交给应用层
                        Message msg;
                        memcpy(msg.data, rcvpkt[i].payload, sizeof(rcvpkt[i].payload));
                        pns->delivertoAppLayer(RECEIVER, msg);
                        rcv[i] = false;
                        i = (i + 1) % M;
                    }
                    else {
                        break;
                    }
                }
                cout << "接收窗口移动前：" << endl;
                for (int j = rcv_base; j != (rcv_base + N) % M; ) {
                    Message msg;
                    memcpy(msg.data, rcvpkt[j].payload, sizeof(rcvpkt[j].payload));
                    msg.data[21] = '\0';
                    cout << msg.data << endl;

                    j = (j + 1) % M;
                }
                cout << endl;
                rcv_base = i;
                cout << "接收窗口移动后：" << endl;
                for (int j = rcv_base; j != (rcv_base + N) % M; ) {
                    //输出窗口
                    Message msg;
                    memcpy(msg.data, rcvpkt[j].payload, sizeof(rcvpkt[j].payload));
                    msg.data[21] = '\0';
                    cout << msg.data << endl;


                    j = (j + 1) % M;
                }
                cout << endl;
            }

        }
        else if (((rcv_base - N + 8) % M < (rcv_base - 1 + 8) % M && (packet.seqnum >= (rcv_base - N + 8) % M && packet.seqnum <= (rcv_base - 1 + 8) % M))
            || ((rcv_base - N + 8) % M > (rcv_base - 1 + 8) % M && ((packet.seqnum >= (rcv_base - N + 8) % M && packet.seqnum <= M - 1) || packet.seqnum <= (rcv_base - 1 + 8) % M)))
        {
            ackpacket.acknum = packet.seqnum;  //确认序号等于收到的报文序号
            ackpacket.checksum = pUtils->calculateCheckSum(ackpacket);
            pUtils->printPacket("接收方发送确认报文", ackpacket);
            pns->sendToNetworkLayer(SENDER, ackpacket); //调用模拟网络环境的sendToNetworkLayer，通过网络层发送确认报文到对方
        }


    }
}
```

Figure 12 SRReceiver::receive implementation

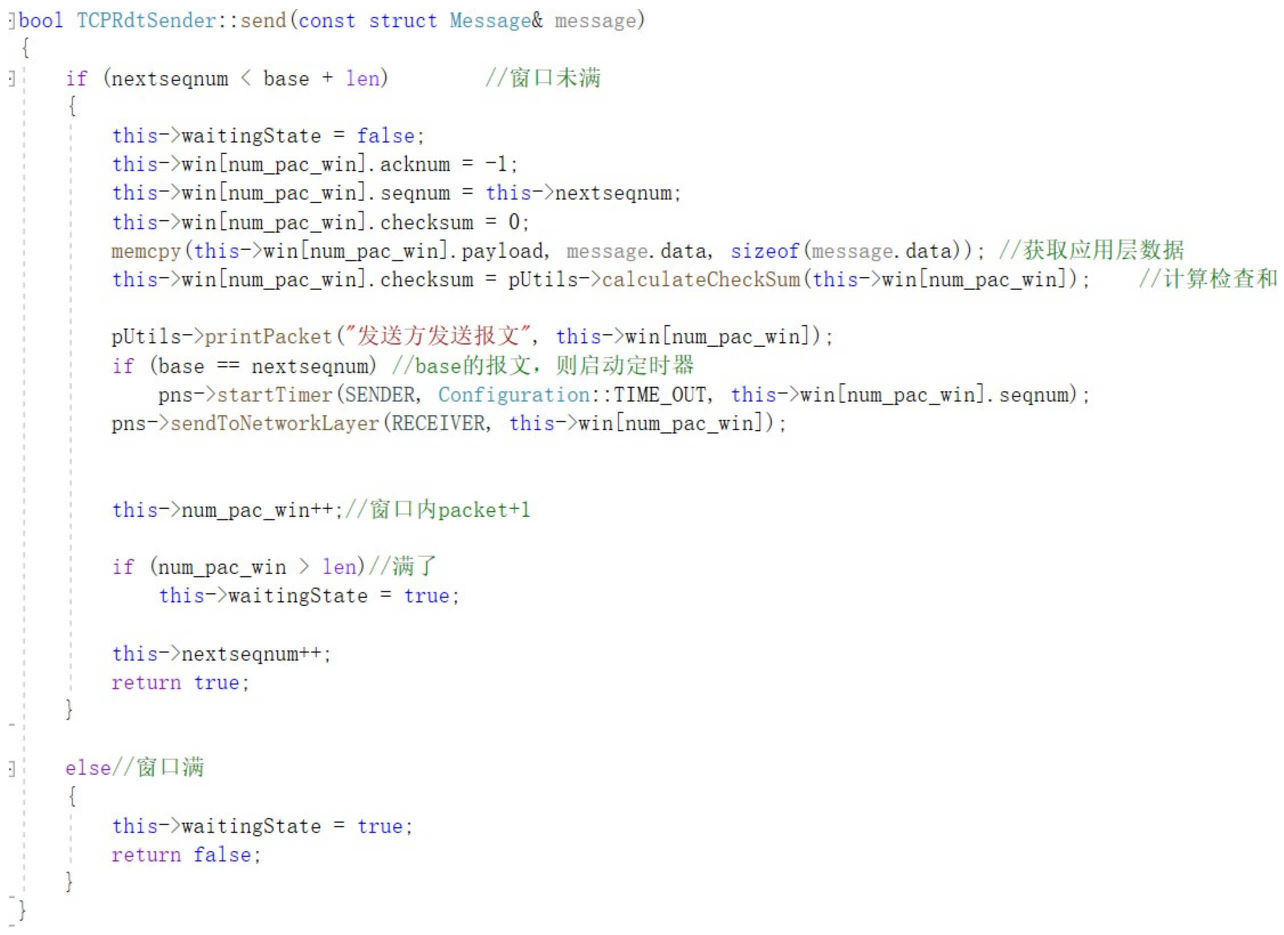
1.4.3 TCP

(1) TCPSender

Events and actions on the sender side:

Use the sending window to control the number of message segments to be sent. After receiving an acknowledgement message, move the window by updating the start and end sequence numbers of the window and send a new message segment. Use the timeout retransmission mechanism to deal with the situation where a message segment fails to be sent. When the timeout period is reached, the message segment is retransmitted if an acknowledgement message has not been received. A single timeout timer is used to handle timeout retransmissions, eliminating the need to estimate the RTT's dynamically adjusted timer Timeout parameter.

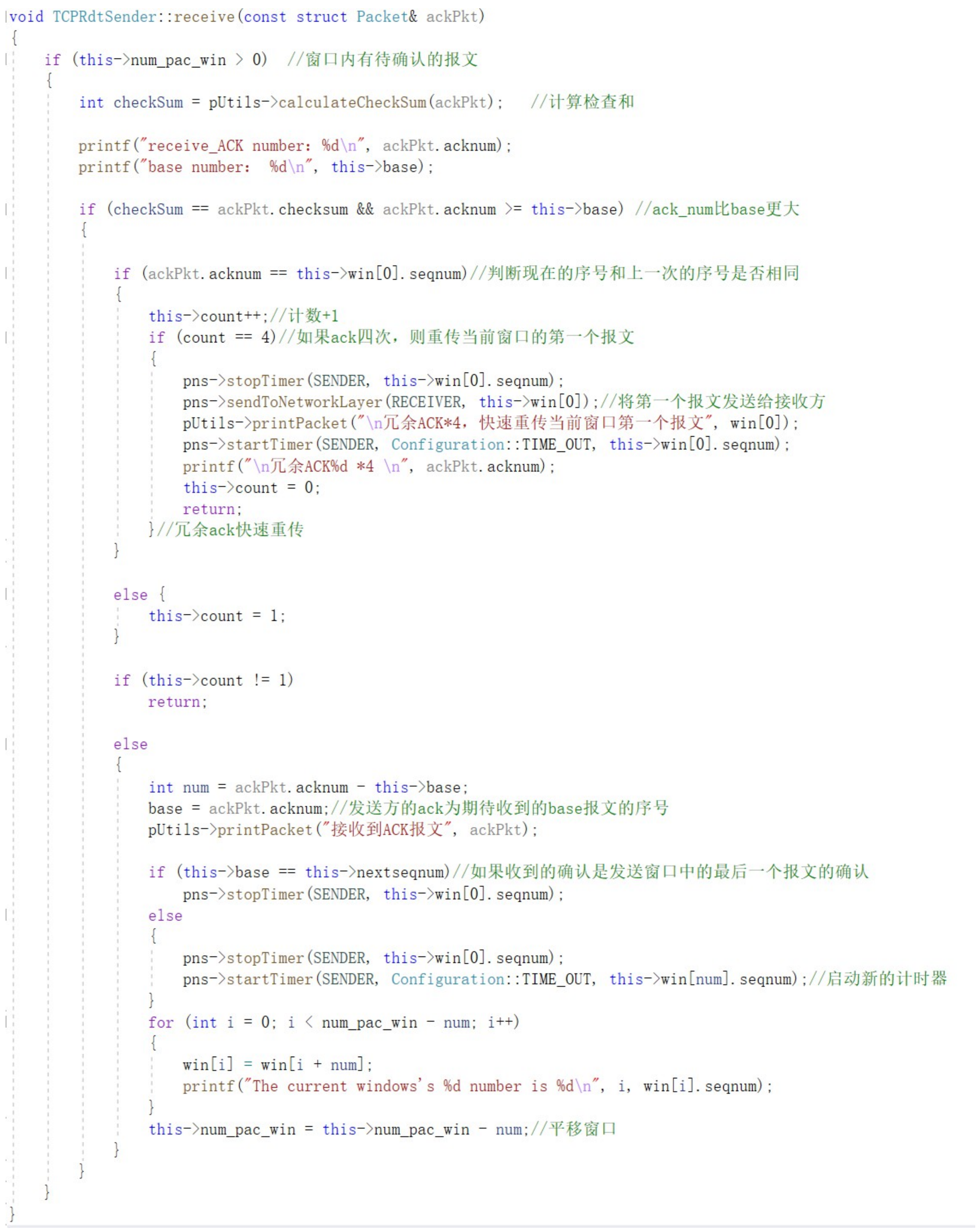
(a) The send method is implemented as shown in Figure 13:



```
bool TCPRdtSender::send(const struct Message& message)
{
    if (nextseqnum < base + len)        //窗口未满
    {
        this->waitingState = false;
        this->win[num_pac_win].acknum = -1;
        this->win[num_pac_win].seqnum = this->nextseqnum;
        this->win[num_pac_win].checksum = 0;
        memcpy(this->win[num_pac_win].payload, message.data, sizeof(message.data)); //获取应用层数据
        this->win[num_pac_win].checksum = pUtils->calculateCheckSum(this->win[num_pac_win]);    //计算检查和

        pUtils->printPacket("发送方发送报文", this->win[num_pac_win]);
        if (base == nextseqnum) //base的报文，则启动定时器
            pns->startTimer(SENDER, Configuration::TIME_OUT, this->win[num_pac_win].seqnum);
        pns->sendToNetworkLayer(RECEIVER, this->win[num_pac_win]);


        this->num_pac_win++;//窗口内packet+1

        if (num_pac_win > len)//满了
            this->waitingState = true;

        this->nextseqnum++;
        return true;
    }

    else//窗口满
    {
        this->waitingState = true;
        return false;
    }
}
```

Figure 13 TCPSender::send implementation

(b) The receive method is shown in Figure 14:



```
void TCPRdtSender::receive(const struct Packet& ackPkt)
{
    if (this->num_pac_win > 0)  //窗口内有待确认的报文
    {
        int checkSum = pUtils->calculateCheckSum(ackPkt);   //计算检查和

        printf("receive_ACK number: %d\n", ackPkt.acknum);
        printf("base number:  %d\n", this->base);

        if (checkSum == ackPkt.checksum && ackPkt.acknum >= this->base) //ack_num比base更大
        {

            if (ackPkt.acknum == this->win[0].seqnum)//判断现在的序号和上一次的序号是否相同
            {
                this->count++;//计数+1
                if (count == 4)//如果ack四次，则重传当前窗口的第一个报文
                {
                    pns->stopTimer(SENDER, this->win[0].seqnum);
                    pns->sendToNetworkLayer(RECEIVER, this->win[0]);//将第一个报文发送给接收方
                    pUtils->printPacket("\n冗余ACK*4, 快速重传当前窗口第一个报文", win[0]);
                    pns->startTimer(SENDER, Configuration::TIME_OUT, this->win[0].seqnum);
                    printf("\n冗余ACK%d *4 \n", ackPkt.acknum);
                    this->count = 0;
                    return;
                }//冗余ack快速重传
            }

            else {
                this->count = 1;
            }

            if (this->count != 1)
                return;

            else
            {
                int num = ackPkt.acknum - this->base;
                base = ackPkt.acknum;//发送方的ack为期待收到的base报文的序号
                pUtils->printPacket("接收到ACK报文", ackPkt);

                if (this->base == this->nextseqnum)//如果收到的确认是发送窗口中的最后一个报文的确认
                    pns->stopTimer(SENDER, this->win[0].seqnum);
                else
                {
                    pns->stopTimer(SENDER, this->win[0].seqnum);
                    pns->startTimer(SENDER, Configuration::TIME_OUT, this->win[num].seqnum);//启动新的计时器
                }
                for (int i = 0; i < num_pac_win - num; i++)
                {
                    win[i] = win[i + num];
                    printf("The current windows's %d number is %d\n", i, win[i].seqnum);
                }
                this->num_pac_win = this->num_pac_win - num;//平移窗口
            }
        }
    }
}
```

Figure 14 TCPSender::receive implementation

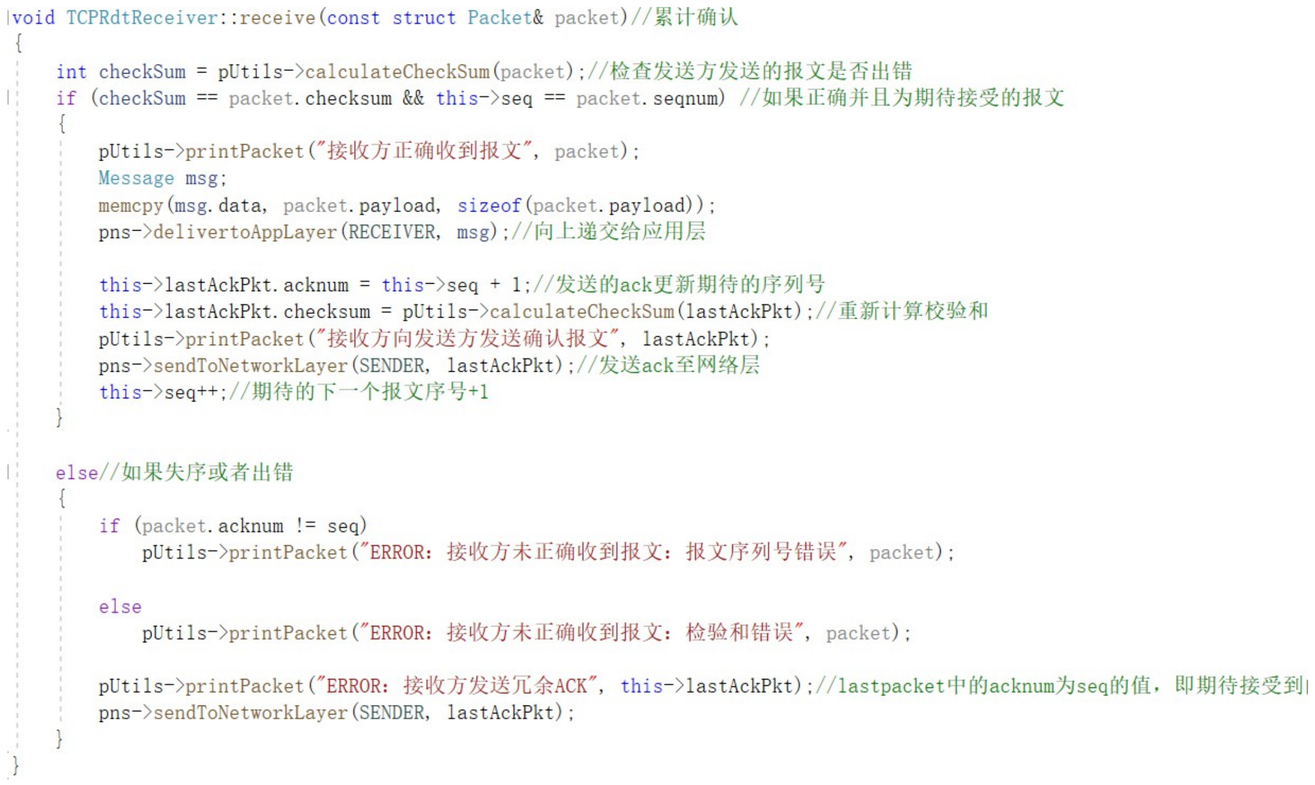
(2) TCPReciever

Receiver events and actions:

On the receiving side, a receive window is used to control the number of message segments received, and an acknowledgement message is used to tell the sender which messages to

The message segment has been received. Use the fast retransmission mechanism to handle missing message segments. When a duplicate message segment is received, send an acknowledgement message to confirm all previous message segments. Use the acknowledgement number to determine the serial number of the last message segment received.

(a) The receive method is implemented as shown in Figure 15:



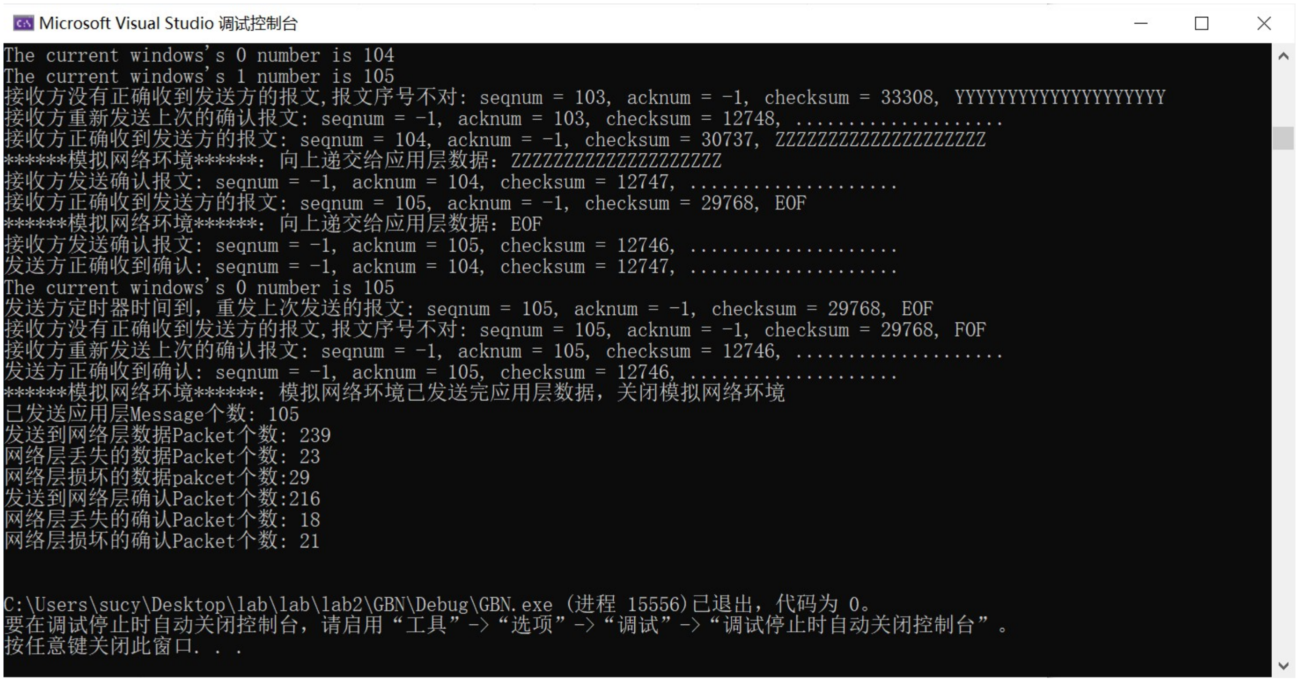
```
void TCPRdtReceiver::receive(const struct Packet& packet)//累计确认
{
    int checkSum = pUtils->calculateCheckSum(packet);//检查发送方发送的报文是否出错
    if (checkSum == packet.checksum && this->seq == packet.seqnum) //如果正确并且为期待接受的报文
    {
        pUtils->printPacket("接收方正确收到报文", packet);
        Message msg;
        memcpy(msg.data, packet.payload, sizeof(packet.payload));
        pns->delivertoAppLayer(RECEIVER, msg);//向上递交给应用层

        this->lastAckPkt.acknum = this->seq + 1;//发送的ack更新期待的序列号
        this->lastAckPkt.checksum = pUtils->calculateCheckSum(lastAckPkt);//重新计算校验和
        pUtils->printPacket("接收方向发送方发送确认报文", lastAckPkt);
        pns->sendToNetworkLayer(SENDER, lastAckPkt);//发送ack至网络层
        this->seq++;//期待的下一个报文序号+1
    }

    else//如果失序或者出错
    {
        if (packet.acknum != seq)
            pUtils->printPacket("ERROR: 接收方未正确收到报文: 报文序列号错误", packet);

        else
            pUtils->printPacket("ERROR: 接收方未正确收到报文: 检验和错误", packet);

        pUtils->printPacket("ERROR: 接收方发送冗余ACK", this->lastAckPkt);//lastpacket中的acknum为seq的值，即期待接受到
        pns->sendToNetworkLayer(SENDER, lastAckPkt);
    }
}
```

Figure 15 TCPReceiver::receive implementation

1.5 Description of system testing and results

1.5.1 GBN

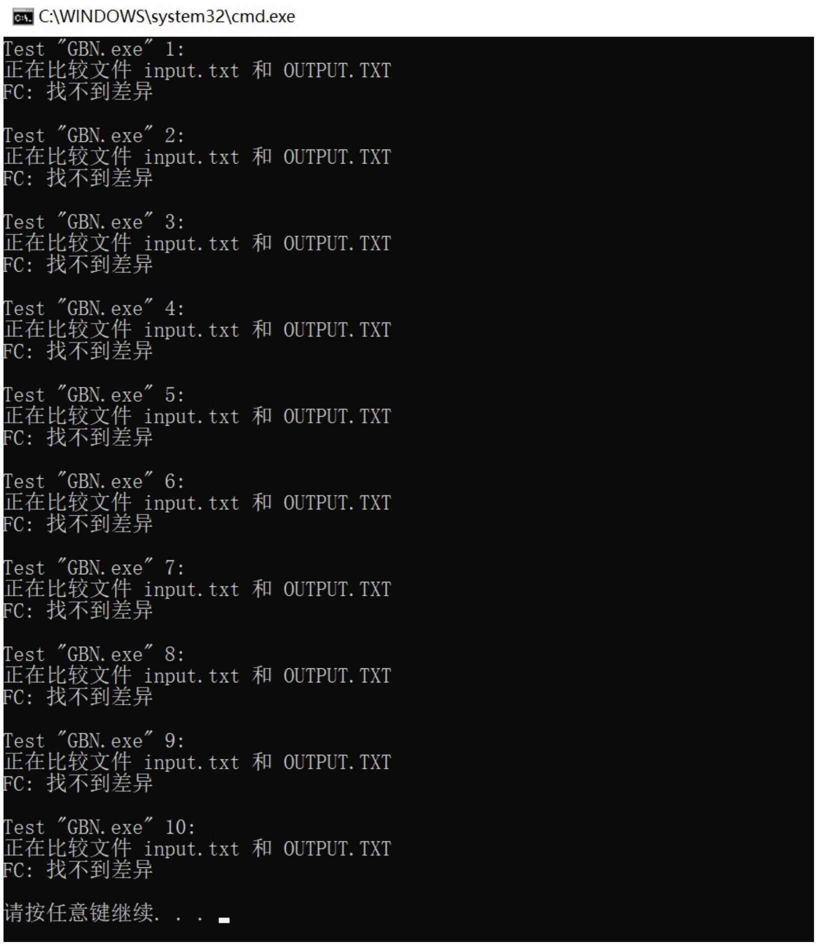
After several tests, this protocol works as expected. Each time the sliding window is moved the console outputs the contents of the sliding window the sliding window is moved correctly. The console output is shown in Figure 16 and the script test results are shown in Figure 17.



```
Microsoft Visual Studio 调试控制台
The current windows's 0 number is 104
The current windows's 1 number is 105
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 103, acknum = -1, checksum = 33308, YYYYYYYYYYYYYYYYYYYY
接收方重新发送上次的确认报文: seqnum = -1, acknum = 103, checksum = 12748, ....................
接收方正确收到发送方的报文: seqnum = 104, acknum = -1, checksum = 30737, ZZZZZZZZZZZZZZZZZZZZ
******模拟网络环境******: 向上递交给应用层数据: ZZZZZZZZZZZZZZZZZZZZ
接收方发送确认报文: seqnum = -1, acknum = 104, checksum = 12747, ....................
接收方正确收到发送方的报文: seqnum = 105, acknum = -1, checksum = 29768, EOF
******模拟网络环境******: 向上递交给应用层数据: EOF
接收方发送确认报文: seqnum = -1, acknum = 105, checksum = 12746, ....................
发送方正确收到确认: seqnum = -1, acknum = 104, checksum = 12747, ....................
The current windows's 0 number is 105
发送方定时器时间到, 重发上次发送的报文: seqnum = 105, acknum = -1, checksum = 29768, EOF
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 105, acknum = -1, checksum = 29768, FOF
接收方重新发送上次的确认报文: seqnum = -1, acknum = 105, checksum = 12746, ....................
发送方正确收到确认: seqnum = -1, acknum = 105, checksum = 12746, ....................
******模拟网络环境******: 模拟网络环境已发送完应用层数据, 关闭模拟网络环境
已发送应用层Message个数: 105
发送到网络层数据Packet个数: 239
网络层丢失的数据Packet个数: 23
网络层损坏的数据pakcet个数:29
发送到网络层确认Packet个数:216
网络层丢失的确认Packet个数: 18
网络层损坏的确认Packet个数: 21


C:\Users\sucy\Desktop\lab\lab\lab2\GBN\Debug\GBN.exe (进程 15556)已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用"工具"->"选项"->"调试"->"调试停止时自动关闭控制台"。
按任意键关闭此窗口. . .
```

Figure 16 GBN Console Outputs

The contents of the input and output files are the same each time:

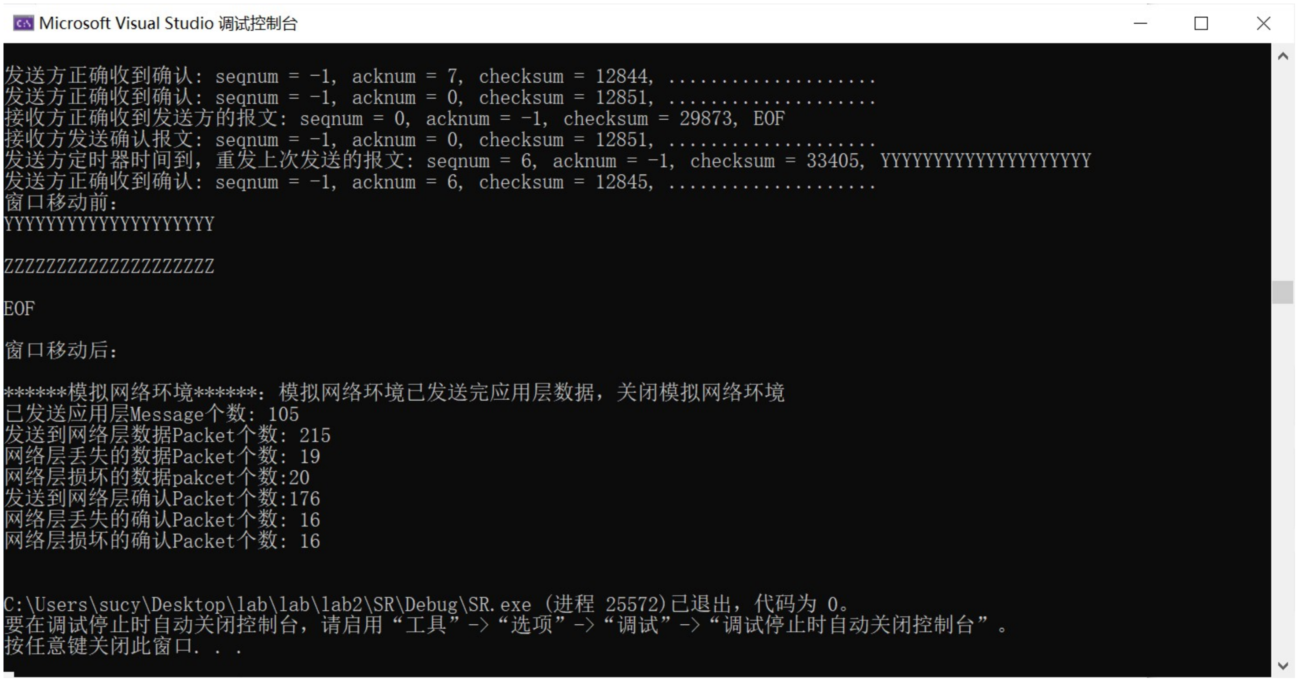


```
C:\WINDOWS\system32\cmd.exe
Test "GBN.exe" 1:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 2:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 3:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

请按任意键继续. . .
```

Figure 17 GBN Output Results Test

1.5.2 SR

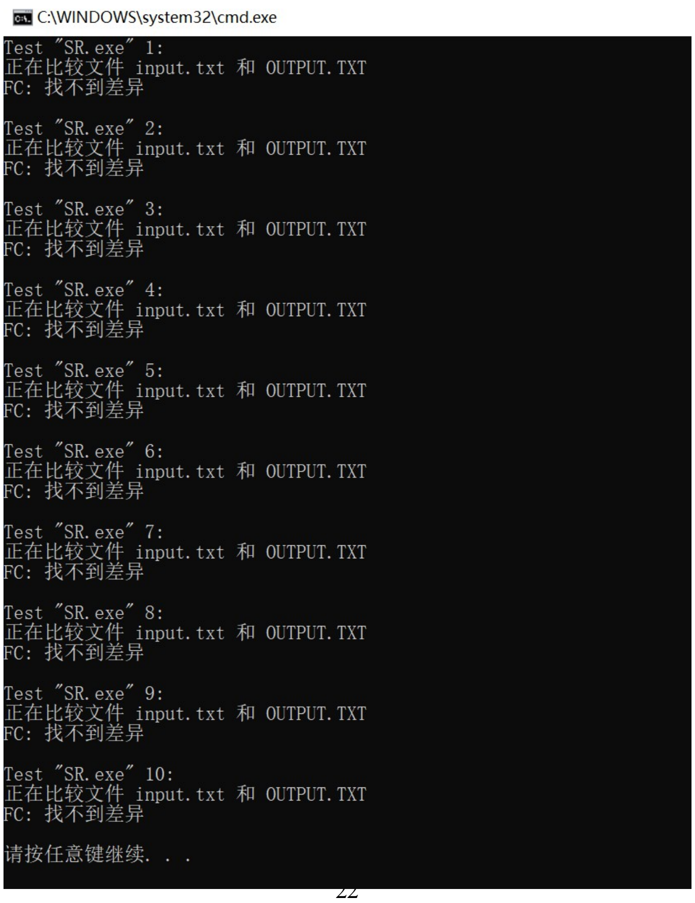
After several tests, this protocol works as expected. Each time the sliding window is moved, the console outputs the content of the sliding window, which is correct, and redirects the console output to the file result.txt. The console output is shown in Figure 18 and the script test results are shown in Figure 19.



Microsoft Visual Studio 调试控制台

```
发送方正确收到确认: seqnum = -1, acknum = 7, checksum = 12844, ....................
发送方正确收到确认: seqnum = -1, acknum = 0, checksum = 12851, ....................
接收方正确收到发送方的报文: seqnum = 0, acknum = -1, checksum = 29873, EOF
接收方发送确认报文: seqnum = -1, acknum = 0, checksum = 12851, ....................
发送方定时器时间到, 重发上次发送的报文: seqnum = 6, acknum = -1, checksum = 33405, YYYYYYYYYYYYYYYYYYYY
发送方正确收到确认: seqnum = -1, acknum = 6, checksum = 12845, ....................
窗口移动前:
YYYYYYYYYYYYYYYYYYYY

ZZZZZZZZZZZZZZZZZZZZ

EOF

窗口移动后:

******模拟网络环境******: 模拟网络环境已发送完应用层数据, 关闭模拟网络环境
已发送应用层Message个数: 105
发送到网络层数据Packet个数: 215
网络层丢失的数据Packet个数: 19
网络层损坏的数据pakcet个数:20
发送到网络层确认Packet个数:176
网络层丢失的确认Packet个数: 16
网络层损坏的确认Packet个数: 16


C:\Users\sucy\Desktop\lab\lab\lab2\SR\Debug\SR.exe (进程 25572)已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用"工具"->"选项"->"调试"->"调试停止时自动关闭控制台"。
按任意键关闭此窗口. . .
```

Figure 18 SR Console Outputs



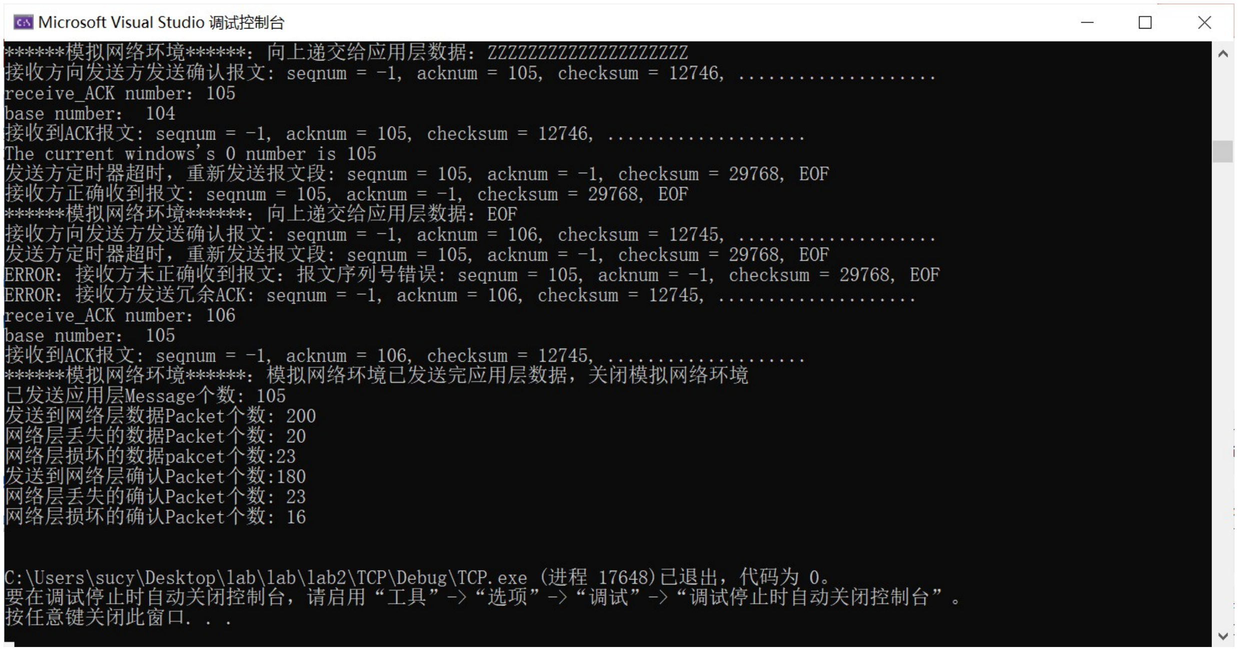
C:\WINDOWS\system32\cmd.exe

```
Test "SR.exe" 1:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 2:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 3:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

请按任意键继续. . .
```

Figure 19 **SR** Output Result Test

1.5.3 TCP

After several tests, this protocol can achieve the expected results. Each time the sliding window is moved, the content of the sliding window is output on the console, the content of the sliding window is correct, and at the same time, the output of the console is redirected to the file

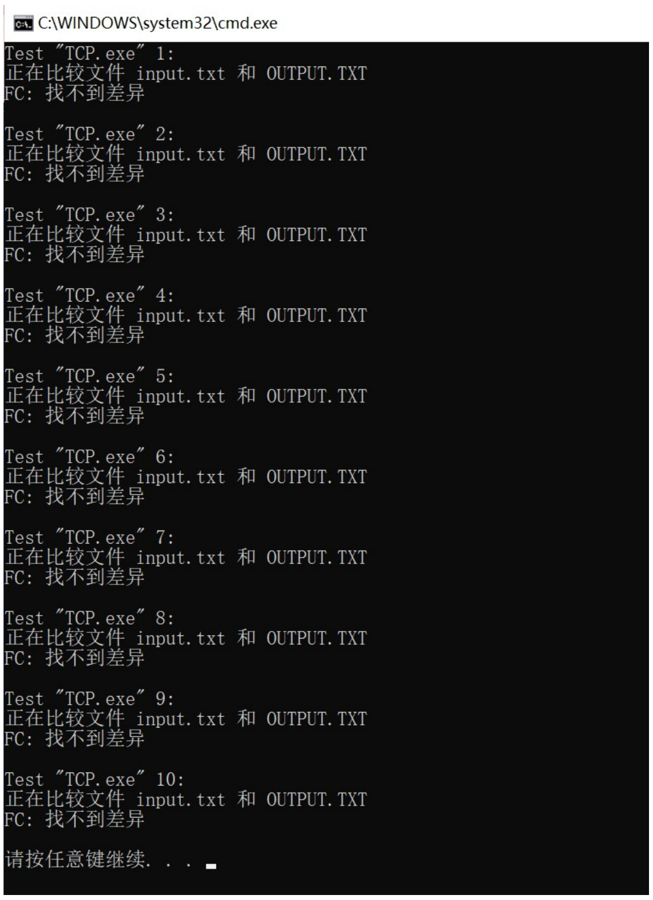


Microsoft Visual Studio 调试控制台

```
******模拟网络环境******: 向上递交给应用层数据: ZZZZZZZZZZZZZZZZZZZZ
接收方向发送方发送确认报文: seqnum = -1, acknum = 105, checksum = 12746, ....................
receive_ACK number: 105
base number:  104
接收到ACK报文: seqnum = -1, acknum = 105, checksum = 12746, ....................
The current windows's 0 number is 105
发送方定时器超时，重新发送报文段: seqnum = 105, acknum = -1, checksum = 29768, EOF
接收方正确收到报文: seqnum = 105, acknum = -1, checksum = 29768, EOF
******模拟网络环境******: 向上递交给应用层数据: EOF
接收方向发送方发送确认报文: seqnum = -1, acknum = 106, checksum = 12745, ....................
发送方定时器超时，重新发送报文段: seqnum = 105, acknum = -1, checksum = 29768, EOF
ERROR: 接收方未正确收到报文: 报文序列号错误: seqnum = 105, acknum = -1, checksum = 29768, EOF
ERROR: 接收方发送冗余ACK: seqnum = -1, acknum = 106, checksum = 12745, ....................
receive_ACK number: 106
base number:  105
接收到ACK报文: seqnum = -1, acknum = 106, checksum = 12745, ....................
******模拟网络环境******: 模拟网络环境已发送完应用层数据，关闭模拟网络环境
已发送应用层Message个数: 105
发送到网络层数据Packet个数: 200
网络层丢失的数据Packet个数: 20
网络层损坏的数据pakcet个数:23
发送到网络层确认Packet个数:180
网络层丢失的确认Packet个数: 23
网络层损坏的确认Packet个数: 16


C:\Users\sucy\Desktop\lab\lab\lab2\TCP\Debug\TCP.exe (进程 17648)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

result.txt, and each time a quick retransmission is performed, a prompt message is output on the console. The console output is shown in Figure 20, and the script test results are shown in Figure 21.

Figure 20 **TCP** Console Output



```
Test "TCP.exe" 1:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 2:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 3:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

请按任意键继续. . .
```

Figure 21 TCP Output Results Test

1.6 Other issues requiring clarification

Test scripts for 3 protocols: check_win.bat

set appname="TCP.exe" ::Execution file names differ for

different test protocols set inputname="input.txt"

set outputname="output.txt"

set resultname="result.txt"

```
for /l %%i in (1,1,10) do (
        echo Test %appname% %%i.
        %appname% > %resultname% 2>&1
        fc /N %inputname% %outputname%
)
pause
```

1.7 bibliography

not have

Experiences and Recommendations

2.1 experience

Learn to communicate using sockets and how to write network programs using sockets in the Socket Programming lab. Learned about different socket types, how to create socket instances, how to bind sockets to specific ports, and how to communicate using sockets. Also, catch and handle exceptions, such as request 404; complete the development of a compliant web server, and be able to complete the implementation of a reliable data transfer protocol.

In the Reliable Data Transfer Protocol Design Experiment, a reliable data transfer protocol is designed and tested by simulating a network environment. A standard set of test data is used to simulate a real network environment and different parameters are used to test the performance of the protocol. The protocol needs to take into account the possible problems of packet loss and disorder in the network and take appropriate measures to ensure that the data can be transmitted correctly network. On the basis of StopWait protocol, we design GBN, compare the characteristics of different protocols, design SR, combine GBN and SR to realize TCP with fast retransmission, and gradually realize more complex protocol requirements, implement corresponding functions and conduct functional tests.

In the CPT-based networking experiment, we learned how to use CPT to design network protocols and understood the working principle of network protocols. Understood the principles and methods of using Cisco Packet Tracer, a network simulation tool, to complete the simulation of local area networks and wide area networks, and configure the simulation equipment, so that the simulation network can achieve the expected experimental goals. Use CPT to build complex network topologies and use different protocols for data transmission. Through CPT, we can better understand the working principle of computer networks and can easily test different network design schemes.

2.2 suggestion

Lots of takeaways, no particular suggestions. Thanks to the course team for the experimental design.