一. RDT2.0 信道上可能出现位错

1.代码分析

1) 接收端

```
//检查校验码，生成ACK
if(CheckSum.computeChkSum(recvPack) == recvPack.getTcpH().getTh_sum()) {
    //生成ACK报文段（设置确认号）
    tcpH.setTh_ack(recvPack.getTcpH().getTh_seq());
    ackPack = new TCP_PACKET(tcpH, tcpS, recvPack.getSourceAddr());
    tcpH.setTh_sum(CheckSum.computeChkSum(ackPack));
    //回复ACK报文段
    reply(ackPack);

    if(recvPack.getTcpH().getTh_seq()!=sequence){
        //将接收到的正确有序的数据插入data队列，准备交付
        dataQueue.add(recvPack.getTcpS().getData());
        sequence=recvPack.getTcpH().getTh_seq();
        //sequence++;
    }else{
        System.out.println("收到重复包,重复seq:"+sequence);
    }

}else{
    System.out.println("校验失败");
    tcpH.setTh_ack(-1);
    ackPack = new TCP_PACKET(tcpH, tcpS, recvPack.getSourceAddr());
    tcpH.setTh_sum(CheckSum.computeChkSum(ackPack));
    //回复ACK报文段
    reply(ackPack);
}
```

接收端:对于接收到每一个包,检查其校验和

- 若校验和匹配,则返回一个ack值为**本次接收到的包的seq值**的包,并将本次接收到的包插入data队列准备交付;
- 若校验和不匹配,则返回一个ack值为-1的包

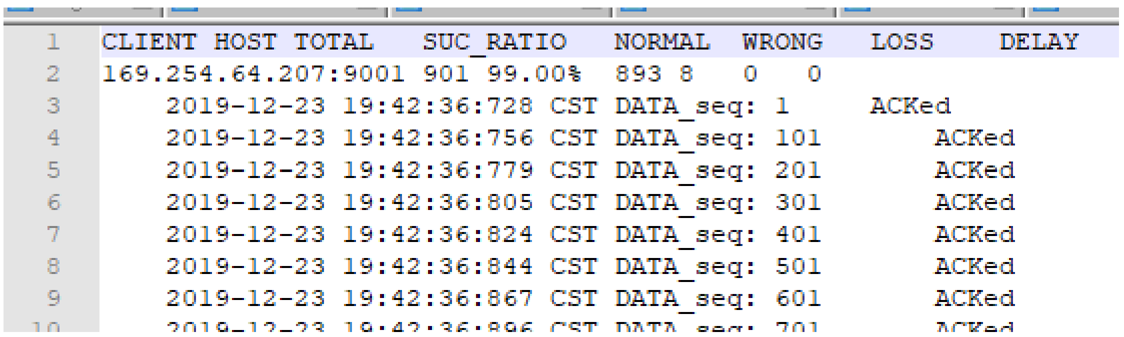
2) 发送端

```
//循环检查确认号对列中是否有新收到的ACK
while(true) {
    if(!ackQueue.isEmpty()){
        int currentAck=ackQueue.poll();
        System.out.println("CurrentAck: "+currentAck);
        if  (currentAck == tcpPack.getTcpH().getTh_seq()){
            System.out.println("Clear: "+tcpPack.getTcpH().getTh_seq());
            //用于3.0:
            //timer.cancel();
            break;
        }else{
            System.out.println("Retransmit: "+tcpPack.getTcpH().getTh_seq());
            udt_send(tcpPack);
        }
    }
}
```

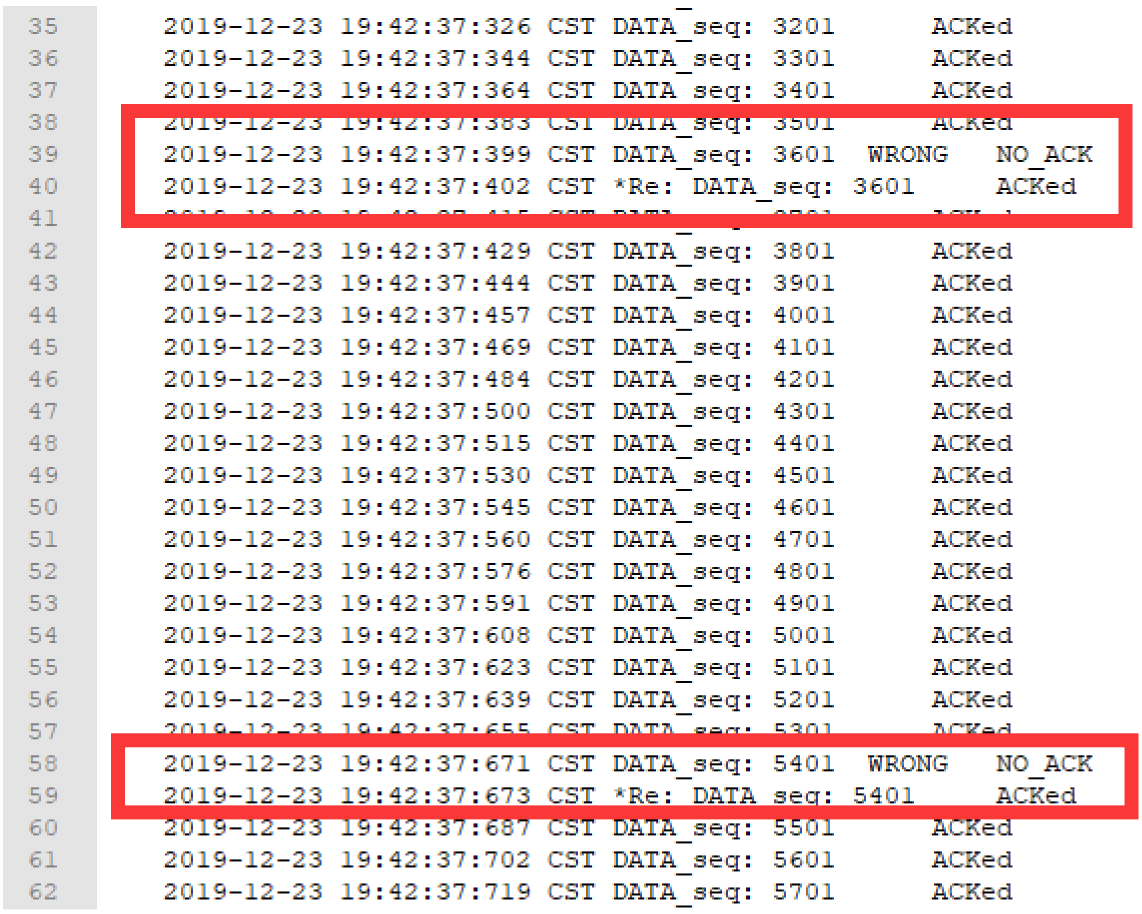
发送端:每次发送一个包后,循环检查确认号对列中是否有新收到的ACK

- 若新收到的ack等于刚刚发送包的seq,则结束本次循环,开始发送下一个包
- 若接收到的ack值不为刚刚发送的seq,则重发之前发送的包,并继续等待ack

2.Log文件



```
CLIENT HOST TOTAL    SUC_RATIO    NORMAL  WRONG    LOSS      DELAY
169.254.64.207:9001 901 99.00%  893 8    0    0
     2019-12-23 19:42:36:728 CST DATA_seq: 1     ACKed
     2019-12-23 19:42:36:756 CST DATA_seq: 101       ACKed
     2019-12-23 19:42:36:779 CST DATA_seq: 201       ACKed
     2019-12-23 19:42:36:805 CST DATA_seq: 301       ACKed
     2019-12-23 19:42:36:824 CST DATA_seq: 401       ACKed
     2019-12-23 19:42:36:844 CST DATA_seq: 501       ACKed
     2019-12-23 19:42:36:867 CST DATA_seq: 601       ACKed
     2019-12-23 19:42:36:896 CST DATA_seq: 701       ACKed
```



```
     2019-12-23 19:42:37:326 CST DATA_seq: 3201      ACKed
     2019-12-23 19:42:37:344 CST DATA_seq: 3301      ACKed
     2019-12-23 19:42:37:364 CST DATA_seq: 3401      ACKed
     2019-12-23 19:42:37:383 CST DATA_seq: 3501      ACKed
     2019-12-23 19:42:37:399 CST DATA_seq: 3601  WRONG    NO_ACK
     2019-12-23 19:42:37:402 CST *Re: DATA_seq: 3601      ACKed
     2019-12-23 19:42:37:415 CST DATA_seq: 3701      ACKed
     2019-12-23 19:42:37:429 CST DATA_seq: 3801      ACKed
     2019-12-23 19:42:37:444 CST DATA_seq: 3901      ACKed
     2019-12-23 19:42:37:457 CST DATA_seq: 4001      ACKed
     2019-12-23 19:42:37:469 CST DATA_seq: 4101      ACKed
     2019-12-23 19:42:37:484 CST DATA_seq: 4201      ACKed
     2019-12-23 19:42:37:500 CST DATA_seq: 4301      ACKed
     2019-12-23 19:42:37:515 CST DATA_seq: 4401      ACKed
     2019-12-23 19:42:37:530 CST DATA_seq: 4501      ACKed
     2019-12-23 19:42:37:545 CST DATA_seq: 4601      ACKed
     2019-12-23 19:42:37:560 CST DATA_seq: 4701      ACKed
     2019-12-23 19:42:37:576 CST DATA_seq: 4801      ACKed
     2019-12-23 19:42:37:591 CST DATA_seq: 4901      ACKed
     2019-12-23 19:42:37:608 CST DATA_seq: 5001      ACKed
     2019-12-23 19:42:37:623 CST DATA_seq: 5101      ACKed
     2019-12-23 19:42:37:639 CST DATA_seq: 5201      ACKed
     2019-12-23 19:42:37:655 CST DATA_seq: 5301      ACKed
     2019-12-23 19:42:37:671 CST DATA_seq: 5401  WRONG    NO_ACK
     2019-12-23 19:42:37:673 CST *Re: DATA_seq: 5401      ACKed
     2019-12-23 19:42:37:687 CST DATA_seq: 5501      ACKed
     2019-12-23 19:42:37:702 CST DATA_seq: 5601      ACKed
     2019-12-23 19:42:37:719 CST DATA_seq: 5701      ACKed
```

分析日志文件可知,本次共有8个包发生了位错误(校验和匹配不成功),对于发生了位错误的包,接收端都马上进行了重发包,并且接收端成功接收到并返回对应ack包

二. RDT2.2 ACK包可能出现位错

1.代码分析

1) 接收端

```
        //检查校验码，生成ACK
        if(CheckSum.computeChkSum(recvPack) == recvPack.getTcpH().getTh_sum()) {
            //生成ACK报文段（设置确认号）
            tcpH.setTh_ack(recvPack.getTcpH().getTh_seq());
            ackPack = new TCP_PACKET(tcpH, tcpS, recvPack.getSourceAddr());
            tcpH.setTh_sum(CheckSum.computeChkSum(ackPack));
            //回复ACK报文段
            reply(ackPack);

            if(recvPack.getTcpH().getTh_seq()!=sequence){
                //将接收到的正确有序的数据插入data队列，准备交付
                dataQueue.add(recvPack.getTcpS().getData());
                sequence=recvPack.getTcpH().getTh_seq();
                //sequence++;
            }else{
                System.out.println("收到重复包,重复seq:"+sequence);
            }

        }else{
            System.out.println("校验失败");
            tcpH.setTh_ack(-1);
            ackPack = new TCP_PACKET(tcpH, tcpS, recvPack.getSourceAddr());
            tcpH.setTh_sum(CheckSum.computeChkSum(ackPack));
            //回复ACK报文段
            reply(ackPack);
        }
```

接收端回复包中仅使用ACK,与RDT2.0的代码类似

- 接收方正确接收一个包后，发送ACK
- 在ACK包中，接收方必须通过序号指明是对哪个数据包的确认

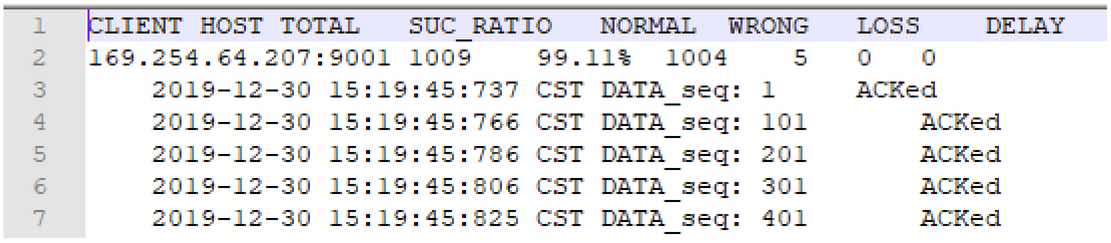
接收方需要记录上次接收的包的seq值,若与本次接收的相同,则不能将它插入data队列

2) 发送端

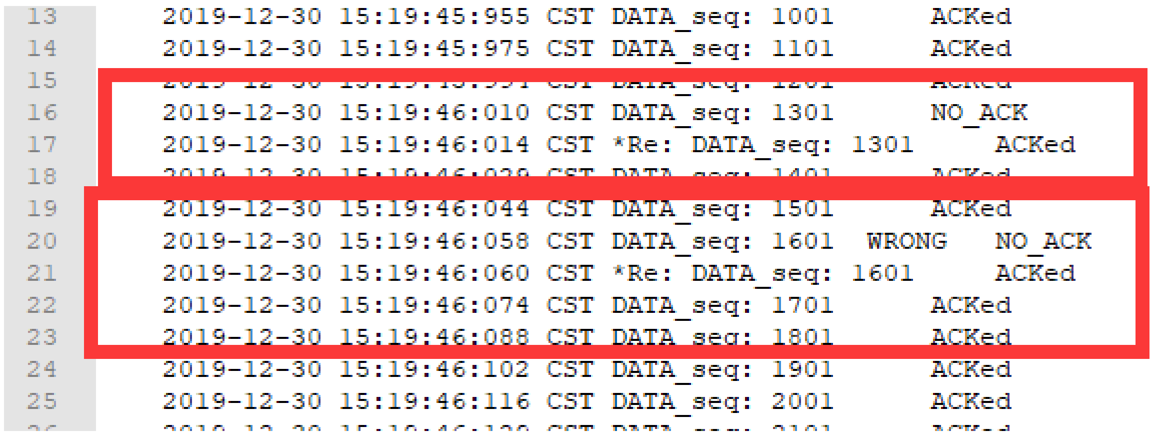
```
    //接收到ACK报文：检查校验和，将确认号插入ack队列;NACK的确认号为－1；3.0版本不需要修改
    public void recv(TCP_PACKET recvPack) {
        if(CheckSum.computeChkSum(recvPack)==recvPack.getTcpH().getTh_sum()){
            System.out.println("Receive ACK Number： "+
recvPack.getTcpH().getTh_ack());
            ackQueue.add(recvPack.getTcpH().getTh_ack());
            System.out.println();
        }else{
            System.out.println("Receive Wrong ACK Number： ");
            ackQueue.add(-1);
            System.out.println();
        }
    }
```

发送端收到发生位错误的ack包时,认为接收方没有正确收到该包,故重复发送本次包

2.Log文件分析



```
CLIENT HOST TOTAL    SUC_RATIO    NORMAL   WRONG    LOSS     DELAY
169.254.64.207:9001 1009     99.11%  1004     5    0    0
    2019-12-30 15:19:45:737 CST DATA_seq: 1     ACKed
    2019-12-30 15:19:45:766 CST DATA_seq: 101       ACKed
    2019-12-30 15:19:45:786 CST DATA_seq: 201       ACKed
    2019-12-30 15:19:45:806 CST DATA_seq: 301       ACKed
    2019-12-30 15:19:45:825 CST DATA_seq: 401       ACKed
```



```
    2019-12-30 15:19:45:955 CST DATA_seq: 1001       ACKed
    2019-12-30 15:19:45:975 CST DATA_seq: 1101       ACKed
    2019-12-30 15:19:45:991 CST DATA_seq: 1201       ACKed
    2019-12-30 15:19:46:010 CST DATA_seq: 1301       NO_ACK
    2019-12-30 15:19:46:014 CST *Re: DATA_seq: 1301      ACKed
    2019-12-30 15:19:46:029 CST DATA_seq: 1401       ACKed
    2019-12-30 15:19:46:044 CST DATA_seq: 1501       ACKed
    2019-12-30 15:19:46:058 CST DATA_seq: 1601   WRONG    NO_ACK
    2019-12-30 15:19:46:060 CST *Re: DATA_seq: 1601      ACKed
    2019-12-30 15:19:46:074 CST DATA_seq: 1701       ACKed
    2019-12-30 15:19:46:088 CST DATA_seq: 1801       ACKed
    2019-12-30 15:19:46:102 CST DATA_seq: 1901       ACKed
    2019-12-30 15:19:46:116 CST DATA_seq: 2001       ACKed
```

分析Log文件可知,对于发送端发送的数据包发生的位错误(**WRONG NO_ACK**),接收端能够检测出并返回对应ack让接收端重发

对于接收端发生的ack包发生的位错误(**NO_ACK**),发送端也能检测出并进行包重发

三. RDT3.0 通道上可能出错和丢失数据

1.代码分析

1) 接收端

接收端代码与之前相同

2) 发送端

```java
class My_UDT_RetransTask extends TimerTask {
        private Client senderClient;
        private TCP_PACKET reTransPacket;

        public My_UDT_RetransTask(Client client, TCP_PACKET packet){
            this.senderClient = client;
            this.reTransPacket = packet;
        }

        @Override
        public void run() {
            System.out.println("超时重发包");
            this.senderClient.send(this.reTransPacket);
        }
    }
```

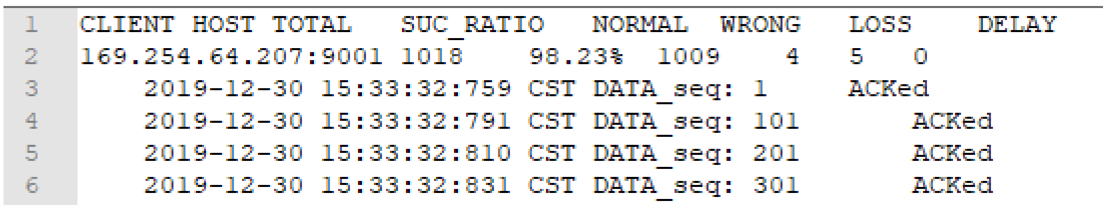
```
//用于3.0版本：设置计时器和超时重传任务
timer = new UDT_Timer();
UDT_RetransTask reTrans = new UDT_RetransTask(client, tcpPack);

//每隔3秒执行重传，直到收到ACK
timer.schedule(reTrans, 3000, 3000);
```

发送本次数据包后,开启一个计时器,三秒内若未收到ack则重发本次数据包

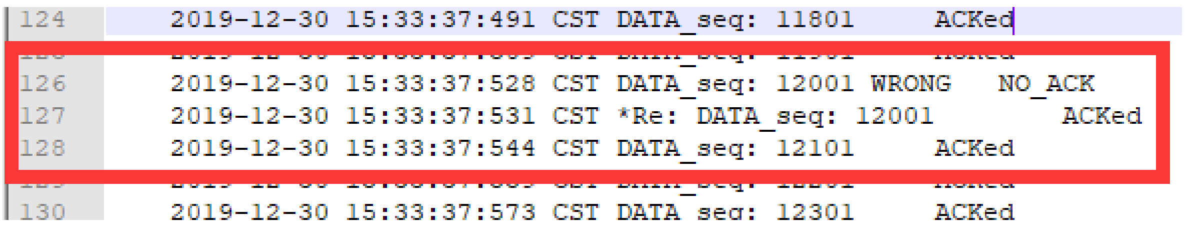
```
while(true) {
            if(!ackQueue.isEmpty()){
                int currentAck=ackQueue.poll();
                System.out.println("CurrentAck: "+currentAck);

                if  (currentAck == tcpPack.getTcpH().getTh_seq()){
                    System.out.println("Clear: "+tcpPack.getTcpH().getTh_seq());

                    //用于3.0：停止等待时需关闭计时器
                    System.out.println("关闭计时器");
                    timer.cancel();
                    break;
                }else{
                    System.out.println("Retransmit:
"+tcpPack.getTcpH().getTh_seq());
                    udt_send(tcpPack);
                    //break;
                }
            }
        }
```

当收到本次数据包时,需要将该数据包对应的计时器关闭

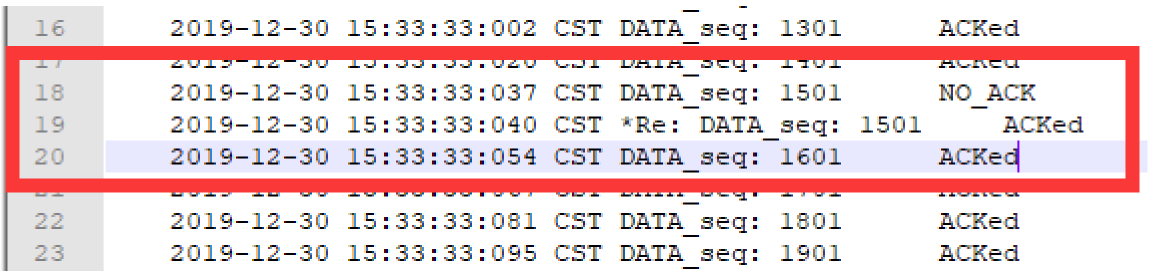
2.Log文件分析



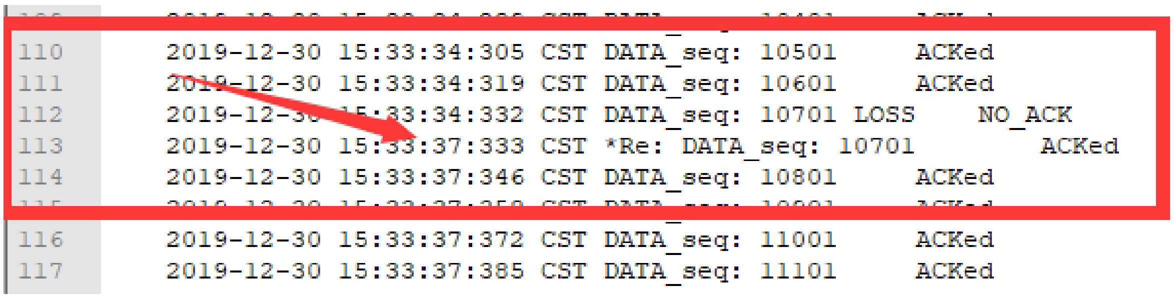
```
CLIENT HOST TOTAL    SUC_RATIO    NORMAL  WRONG    LOSS     DELAY
169.254.64.207:9001 1018     98.23%   1009     4    5    0
     2019-12-30 15:33:32:759 CST DATA_seq: 1      ACKed
     2019-12-30 15:33:32:791 CST DATA_seq: 101        ACKed
     2019-12-30 15:33:32:810 CST DATA_seq: 201        ACKed
     2019-12-30 15:33:32:831 CST DATA_seq: 301        ACKed
```



```
     2019-12-30 15:33:37:491 CST DATA_seq: 11801      ACKed
     [illegible]
     2019-12-30 15:33:37:528 CST DATA_seq: 12001 WRONG    NO_ACK
     2019-12-30 15:33:37:531 CST *Re: DATA_seq: 12001          ACKed
     2019-12-30 15:33:37:544 CST DATA_seq: 12101      ACKed
     [illegible]
     2019-12-30 15:33:37:573 CST DATA_seq: 12301      ACKed
```



```
     2019-12-30 15:33:33:002 CST DATA_seq: 1301       ACKed
     [illegible]
     2019-12-30 15:33:33:037 CST DATA_seq: 1501       NO_ACK
     2019-12-30 15:33:33:040 CST *Re: DATA_seq: 1501       ACKed
     2019-12-30 15:33:33:054 CST DATA_seq: 1601       ACKed
     [illegible]
     2019-12-30 15:33:33:081 CST DATA_seq: 1801       ACKed
     2019-12-30 15:33:33:095 CST DATA_seq: 1901       ACKed
```



```
2019-12-30 15:33:34:305 CST DATA_seq: 10501      ACKed
2019-12-30 15:33:34:319 CST DATA_seq: 10601      ACKed
2019-12-30 15:33:34:332 CST DATA_seq: 10701 LOSS     NO_ACK
2019-12-30 15:33:37:333 CST *Re: DATA_seq: 10701         ACKed
2019-12-30 15:33:37:346 CST DATA_seq: 10801      ACKed
2019-12-30 15:33:37:372 CST DATA_seq: 11001      ACKed
2019-12-30 15:33:37:385 CST DATA_seq: 11101      ACKed
```

分析Log文件,当发生丢包时(**LOSS NO_ACK**),发送端会在3s后自动重发包

四. RDT4.0 选择响应协议

1.代码分析

1) 接收端

```
    public void addRecvPacket(TCP_PACKET packet){
        // 判断是否有序
        int seq=packet.getTcpH().getTh_seq();
        if((seq==lastSaveSeq+lastLength)||lastSaveSeq==-1){
            lastLength=packet.getTcpS().getData().length;
            lastSaveSeq=seq;
            waitWrite(packet);
        }else if(seq>lastSaveSeq){
            System.out.println("缓存seq:"+seq+"到列表,last is:"+lastSaveSeq);
            recvContent.add(new Window(packet));
        }
    }
```

接收端对于每一个校验和正确的接收包,都进行应答

- 若接收到的包的seq有序,则执行waitWrite()函数将其递交给上层
- 若收到的包的seq失序,则缓存到一个有序集合recvContent里

```
    public void waitWrite(TCP_PACKET packet){
        int seq;

        File fw = new File("recvData.txt");
        BufferedWriter writer;
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//设置
日期格式
        try {
            writer = new BufferedWriter(new FileWriter(fw, true));
            Window window;
            int[] data=packet.getTcpS().getData();
            for(int i = 0; i < data.length; i++) {
                writer.write(data[i] + "\n");
            }
            writer.flush();      //清空输出缓存
            Iterator<Window> it=recvContent.iterator();
            // 在缓存队列里看是否还有有序的包,一起向上递交
            while (it.hasNext()){
                window=it.next();
                seq=window.packet.getTcpH().getTh_seq();
                data=window.packet.getTcpS().getData();
                if(seq==lastSaveSeq+lastLength){// 判断是否有序
                    lastLength=packet.getTcpS().getData().length;
                    lastSaveSeq=seq;
                    for(int i = 0; i < data.length; i++) {
                        writer.write(data[i] + "\n");
                    }
                    writer.flush();      //清空输出缓存
                    it.remove();
                }
                else{
//                    System.out.println("退出循环,当前seq为:"+seq+"
last:"+lastSaveSeq);
                    break;
                }
            }
            writer.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

waitWrite()函数功能: 将本次有序包递交给上层,并检查缓存队列里否还有有序的包,一起向上递交

2) 发送端

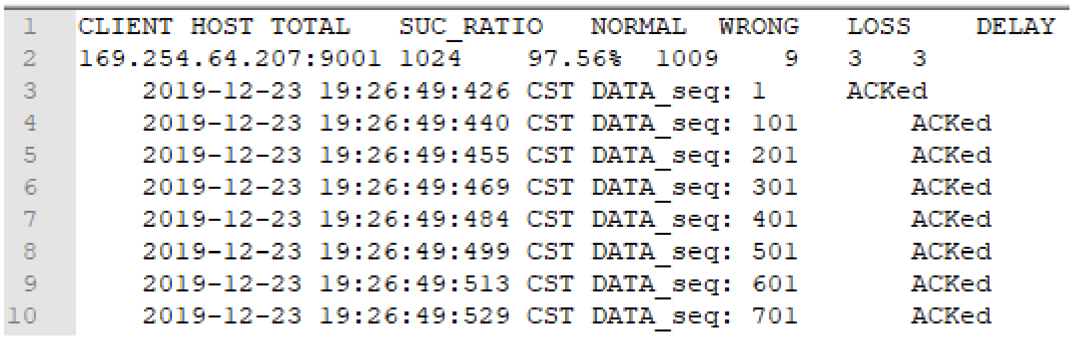
```
        while (!sendWindow.continueSend()){
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

发送端每发送一个包则判断当前窗口是否还有空闲,若有则发送下一个包,若无则等待窗口空闲.这里固定窗口大小为100

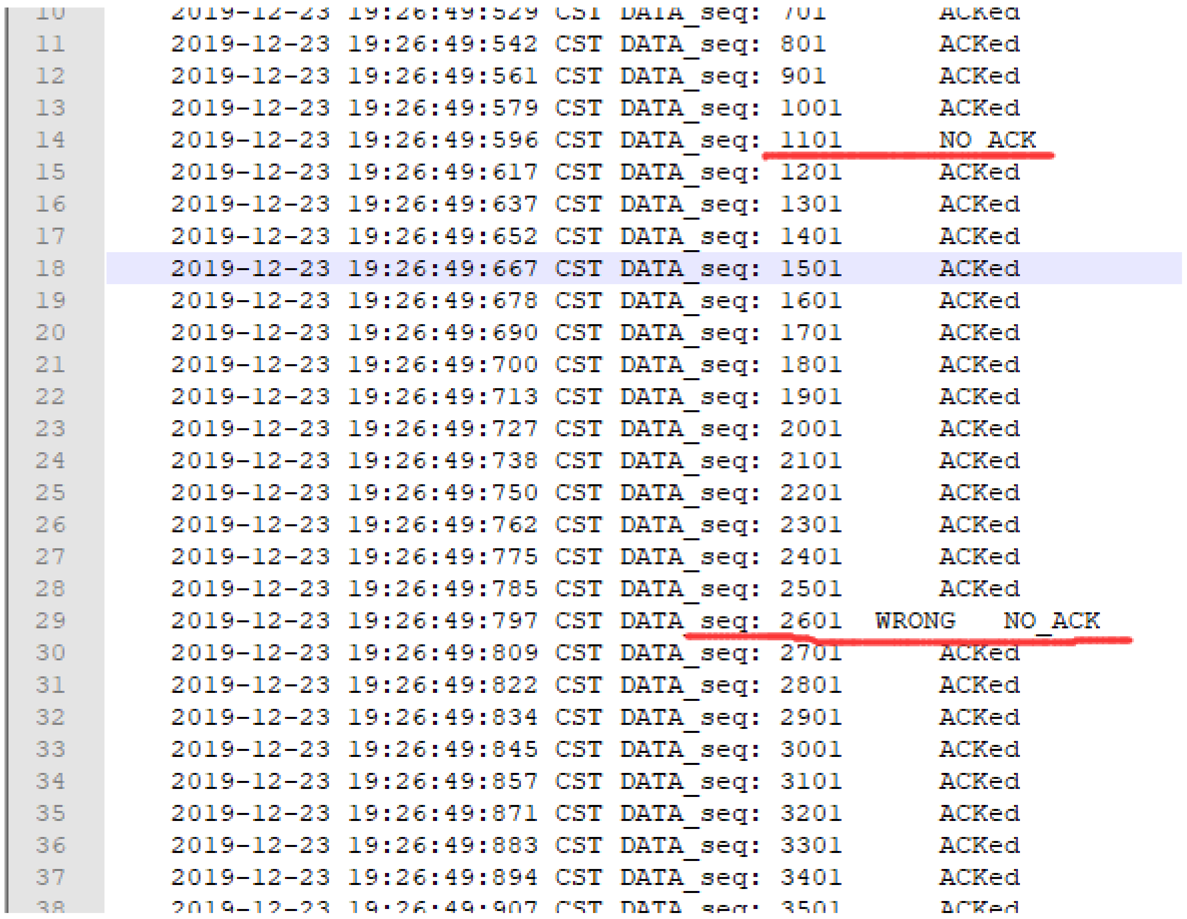
```
    public void waitOvertime() {
        TimerTask dealOverTime = new TimerTask() {
            @Override
            public void run() {
                int index = startWindowIndex;
                boolean updateStart=true;
                Window window;
                while (index < endWindosIndex) {
                    // 如果第index个包超时了
                    window = sendContent.get(index);
                    if(updateStart && window.ack){
                        startWindowIndex=index+1;
                        logger.info("更新start值:"+startWindowIndex);
                    }else if(!window.ack){
                        updateStart=false;
                        if (TIMEOUTTIME < (System.currentTimeMillis() -
window.getStartSendTime())) {
                            //  它没有收到ack,则尝试重发
                            sendWindow(window,false);
                        }
                    }
                    index++;
                }
            }
        };
        new Timer().schedule(dealOverTime, 0, 200);
    }
```

发送端处理超时的包,从滑动窗口头开始逐个检查是否超时,如果有超时且未收到ack的包,则进行重发,若头部的有新的连续ack,则更新窗口头部的下标

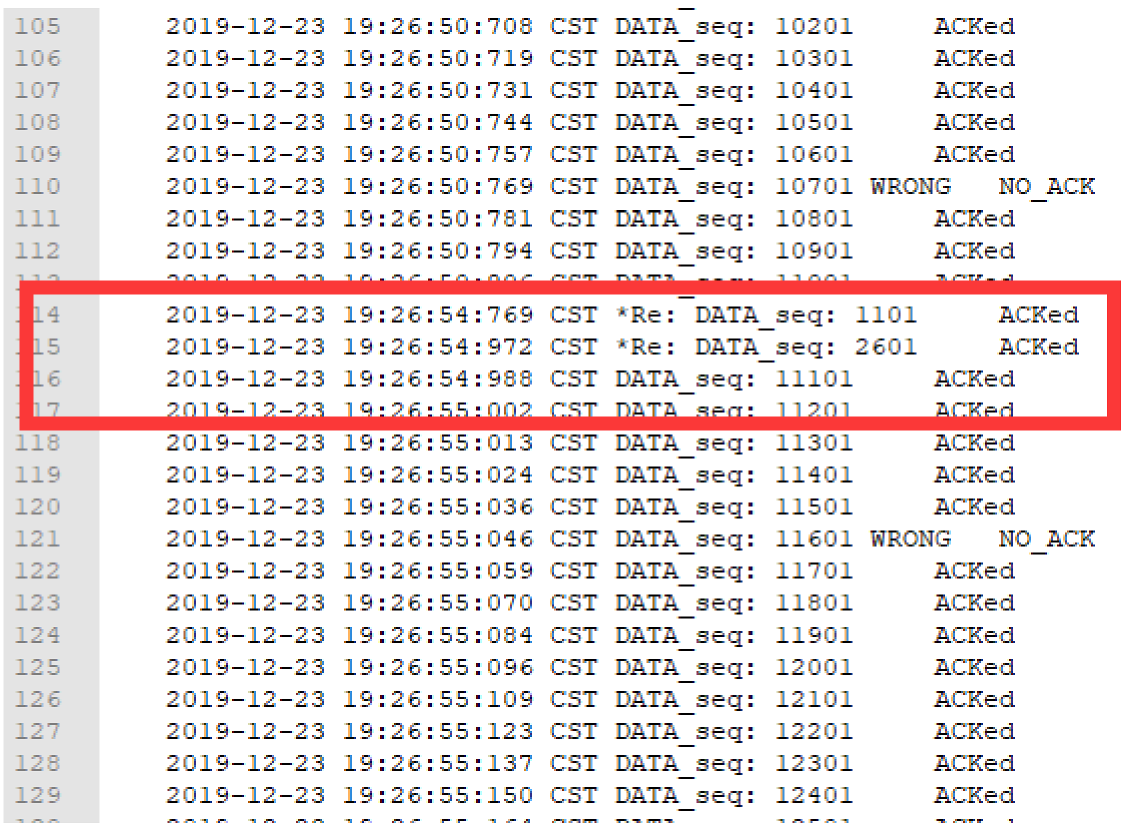
2.Log文件分析



```
CLIENT HOST TOTAL   SUC_RATIO   NORMAL  WRONG   LOSS    DELAY
169.254.64.207:9001 1024    97.56%  1009    9   3   3
    2019-12-23 19:26:49:426 CST DATA_seq: 1     ACKed
    2019-12-23 19:26:49:440 CST DATA_seq: 101       ACKed
    2019-12-23 19:26:49:455 CST DATA_seq: 201       ACKed
    2019-12-23 19:26:49:469 CST DATA_seq: 301       ACKed
    2019-12-23 19:26:49:484 CST DATA_seq: 401       ACKed
    2019-12-23 19:26:49:499 CST DATA_seq: 501       ACKed
    2019-12-23 19:26:49:513 CST DATA_seq: 601       ACKed
    2019-12-23 19:26:49:529 CST DATA_seq: 701       ACKed
```



```
    2019-12-23 19:26:49:529 CST DATA_seq: 701       ACKed
    2019-12-23 19:26:49:542 CST DATA_seq: 801       ACKed
    2019-12-23 19:26:49:561 CST DATA_seq: 901       ACKed
    2019-12-23 19:26:49:579 CST DATA_seq: 1001      ACKed
    2019-12-23 19:26:49:596 CST DATA_seq: 1101      NO_ACK
    2019-12-23 19:26:49:617 CST DATA_seq: 1201      ACKed
    2019-12-23 19:26:49:637 CST DATA_seq: 1301      ACKed
    2019-12-23 19:26:49:652 CST DATA_seq: 1401      ACKed
    2019-12-23 19:26:49:667 CST DATA_seq: 1501      ACKed
    2019-12-23 19:26:49:678 CST DATA_seq: 1601      ACKed
    2019-12-23 19:26:49:690 CST DATA_seq: 1701      ACKed
    2019-12-23 19:26:49:700 CST DATA_seq: 1801      ACKed
    2019-12-23 19:26:49:713 CST DATA_seq: 1901      ACKed
    2019-12-23 19:26:49:727 CST DATA_seq: 2001      ACKed
    2019-12-23 19:26:49:738 CST DATA_seq: 2101      ACKed
    2019-12-23 19:26:49:750 CST DATA_seq: 2201      ACKed
    2019-12-23 19:26:49:762 CST DATA_seq: 2301      ACKed
    2019-12-23 19:26:49:775 CST DATA_seq: 2401      ACKed
    2019-12-23 19:26:49:785 CST DATA_seq: 2501      ACKed
    2019-12-23 19:26:49:797 CST DATA_seq: 2601  WRONG   NO_ACK
    2019-12-23 19:26:49:809 CST DATA_seq: 2701      ACKed
    2019-12-23 19:26:49:822 CST DATA_seq: 2801      ACKed
    2019-12-23 19:26:49:834 CST DATA_seq: 2901      ACKed
    2019-12-23 19:26:49:845 CST DATA_seq: 3001      ACKed
    2019-12-23 19:26:49:857 CST DATA_seq: 3101      ACKed
    2019-12-23 19:26:49:871 CST DATA_seq: 3201      ACKed
    2019-12-23 19:26:49:883 CST DATA_seq: 3301      ACKed
    2019-12-23 19:26:49:894 CST DATA_seq: 3401      ACKed
    2019-12-23 19:26:49:907 CST DATA_seq: 3501      ACKed
```



```
2019-12-23 19:26:50:708 CST DATA_seq: 10201      ACKed
2019-12-23 19:26:50:719 CST DATA_seq: 10301      ACKed
2019-12-23 19:26:50:731 CST DATA_seq: 10401      ACKed
2019-12-23 19:26:50:744 CST DATA_seq: 10501      ACKed
2019-12-23 19:26:50:757 CST DATA_seq: 10601      ACKed
2019-12-23 19:26:50:769 CST DATA_seq: 10701 WRONG   NO_ACK
2019-12-23 19:26:50:781 CST DATA_seq: 10801      ACKed
2019-12-23 19:26:50:794 CST DATA_seq: 10901      ACKed
2019-12-23 19:26:50:806 CST DATA_seq: 11001      ACKed
2019-12-23 19:26:54:769 CST *Re: DATA_seq: 1101      ACKed
2019-12-23 19:26:54:972 CST *Re: DATA_seq: 2601      ACKed
2019-12-23 19:26:54:988 CST DATA_seq: 11101      ACKed
2019-12-23 19:26:55:002 CST DATA_seq: 11201      ACKed
2019-12-23 19:26:55:013 CST DATA_seq: 11301      ACKed
2019-12-23 19:26:55:024 CST DATA_seq: 11401      ACKed
2019-12-23 19:26:55:036 CST DATA_seq: 11501      ACKed
2019-12-23 19:26:55:046 CST DATA_seq: 11601 WRONG   NO_ACK
2019-12-23 19:26:55:059 CST DATA_seq: 11701      ACKed
2019-12-23 19:26:55:070 CST DATA_seq: 11801      ACKed
2019-12-23 19:26:55:084 CST DATA_seq: 11901      ACKed
2019-12-23 19:26:55:096 CST DATA_seq: 12001      ACKed
2019-12-23 19:26:55:109 CST DATA_seq: 12101      ACKed
2019-12-23 19:26:55:123 CST DATA_seq: 12201      ACKed
2019-12-23 19:26:55:137 CST DATA_seq: 12301      ACKed
2019-12-23 19:26:55:150 CST DATA_seq: 12401      ACKed
```

分析Log文件,对于NOACK的包,都能在超时的时候进行重发包

五. 拥塞控制

1.代码分析

1) 接收端

```
    public int addRecvPacket(TCP_PACKET packet){
        int seq=packet.getTcpH().getTh_seq();
        if(seq==lastSaveSeq+lastLength || lastSaveSeq==-1){
            lastLength=packet.getTcpS().getData().length;
            lastSaveSeq=seq;
            contentList.add(packet);
            waitWrite();
            logger.info("有序接收,缓存seq:"+seq+"到列表,返回ack:"+lastSaveSeq);
        }else if(seq>lastSaveSeq){
            recvBuffer.add(packet);
            logger.info("失序接收,缓存seq:"+seq+"到列表,返回ack:"+lastSaveSeq);
        }
        return lastSaveSeq;
    }
```

与选择响应协议一致,对于每一个校验和正确的接收包,都进行应答

- 若接收到的包的seq有序,则执行waitWrite()函数将其递交给上层
- 若收到的包的seq失序,则缓存到一个有序集合recvContent里

2) 发送端

```
    void dealWithOvertime() {
        TimerTask dealOverTime = new TimerTask() {
            @Override
            public void run() {
                int index = startWindowIndex;
                Window window;
                while (index <= ackWindowIndex) {
                    // 如果第index个包超时了
                    window = sendContent.get(index);
                    if (TIMEOUTTIME < (System.currentTimeMillis() -
window.getStartSendTime())) {
                        //  它没有收到ack,则尝试重发
                        if (!window.isAck()) {
                            sendWindow(sendContent.get(index),1);
                            break;
                        }
                    }
                    index++;
                }
            }
        };
        new Timer().schedule(dealOverTime, 0, 1000);
    }
```

发送端处理超时的包,从滑动窗口头开始逐个检查是否超时,如果有超时且未收到ack的包,则进行重发

```
    public void recv(TCP_PACKET recvPack){
        boolean isBadNet = false;
        Window window = null;
        int ackNum=recvPack.getTcpH().getTh_ack();
        logger.info("接收到ack:"+ackNum);

        int ackIndex=indexMap.get(ackNum);
        if(ackIndex>=startWindowIndex){
            // 如果收到的不是延迟到达的包,则处理
            int tempSeq;
            int index=startWindowIndex;

            // 当滑动窗口还有空间
            for (; index <=ackWindowIndex ; index++) {
                window=sendContent.get(index);
                tempSeq=window.packet.getTcpH().getTh_seq();

                // 包里的ack 大于滑动窗口里Index下标对应包的窗口的话,说明前面的也收到了
                if (ackIndex >= indexMap.get(tempSeq)) {
                    logger.info(getWindowInfo()+"接收到ackNum:"+tempSeq+" (大于当
前)index为:"+index+"的窗口块已经ack");
                    window.setAck(true);
                } else {
                    // 该窗口的ack数量+1
                    window.setDuplicateAckNum(window.getDuplicateAckNum() + 1);

                    // 如果该包收到3次ack时,说明网络拥塞
                    if ((window.getDuplicateAckNum() >= MAX_Duplicate_NUM)&&
(!window.isAck())) {
                        isBadNet = true;
                    }
                    break;
                }
            }
            updateWindowSize(ackIndex);
        }else{
            logger.warning("收到延迟ack包,ackIndex值:"+ackIndex);
        }

        if (isBadNet) {
            // 拥塞避免 如果有包被重复收到MAX_Duplicate_NUM次以上,说明网络不好,缩小窗口
            int oldSsthresh=ssthresh;
            ssthresh = Math.max((cwnd / 2),2);
            // TCP Tahoe方式
            // cwnd = 1;

            // TCP Reno方式
            cwnd=oldSsthresh+1;//快速回恢复

            logger.warning(String.format(getWindowInfo()+"网络拥挤,设置新门限:%d,阻
塞窗口大小为:%d, 当前窗口范围(%d,%d),acknum=%d\n",
ssthresh,cwnd,startWindowIndex,endWindosIndex,ackWindowIndex));

            // 快速重传
            updateWindowSize(ackIndex);
            window.setDuplicateAckNum(0);
            sendWindow(window,2);

        }else {
            // 网络状况良好,增大滑动窗口
            cwnd=(cwnd <= ssthresh)?cwnd*2:cwnd+1;// 加法增大
            if(cwnd>MAX_Window_Size){
                cwnd=MAX_Window_Size;
            }
            updateWindowSize(ackIndex);
            logger.info(String.format(getWindowInfo()+"网络良好,设置阻塞窗口大小:%d,
当前窗口范围(%d,%d),ackWindowIndex=%d\n",
cwnd,startWindowIndex,endWindosIndex,ackWindowIndex));

        }

    }
```

对于一个到达的未出错的ack包(即校验和正确的包)

发送端先判断是否延迟到达的包(比较接收到的ack值和当前的滑动窗口左沿的ack来判断)

- 若收到的不是延迟到达的包,则更新滑动窗口的左沿,并将ack值对应的窗口及其左边的窗口设置为已经ack,并将ack值对应的下一个窗口的DuplicateAckNum+1, 若此时该窗口的DuplicateAckNum大于等于3,说明此时网络环境差,则设置isBadNet为true,表示需要进行拥塞控制

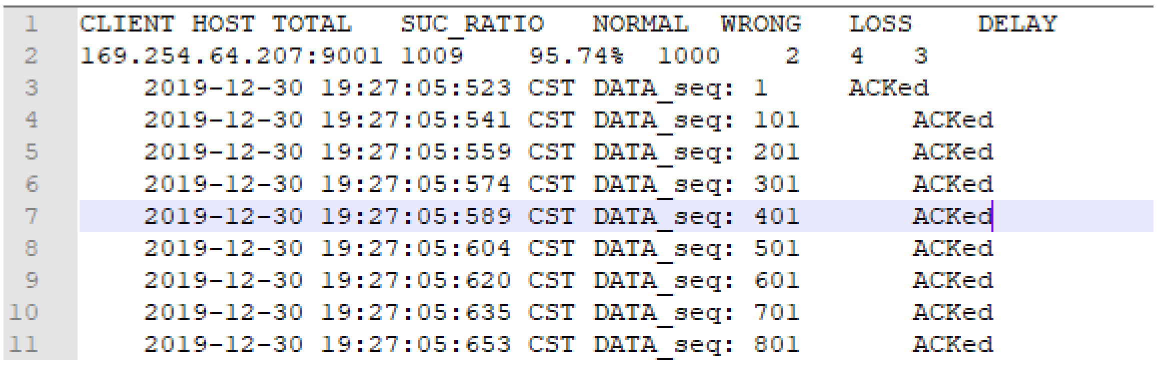
快速恢复\乘法减小

当isBadNet为true,进行拥塞避免, 窗口门限设置为当前窗口大小的1/2(**乘法减小**),窗口大小cwnd设置为原来的门限值+1(**Reno方式,快速恢复**), 并进行快速重传,发送接收端返回的ack对应的下一个窗口的包

加法增大

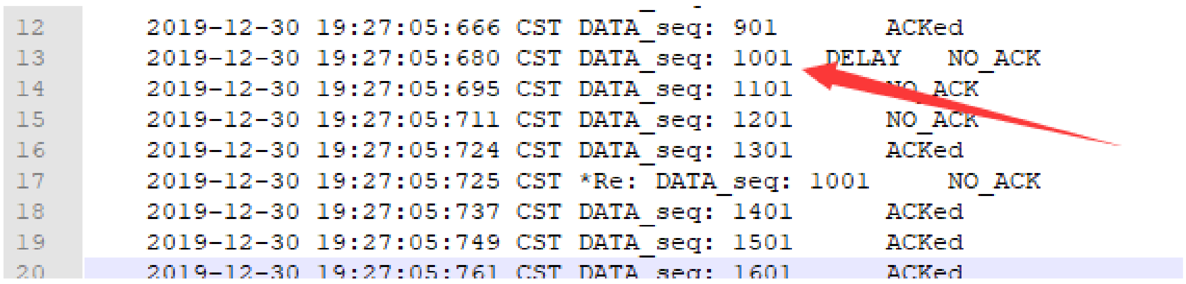
当isBadNet为false时,网络良好, 增大滑动窗口, 当窗口值不大于门限值时,平方增大,大于门限值时,采用加法增大

2.Log文件分析



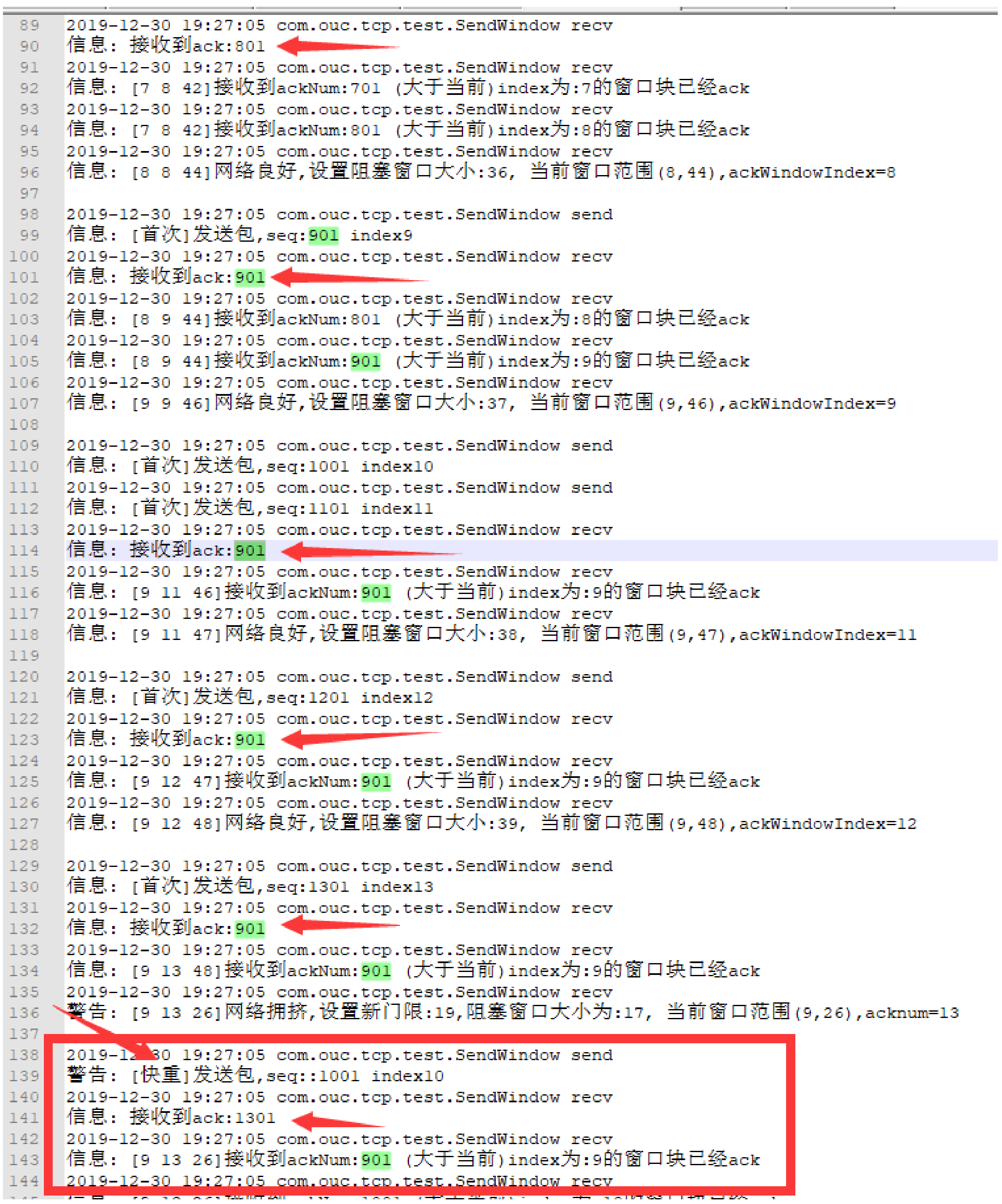
```
CLIENT HOST TOTAL    SUC_RATIO    NORMAL  WRONG   LOSS     DELAY
169.254.64.207:9001 1009     95.74%  1000     2    4    3
    2019-12-30 19:27:05:523 CST DATA_seq: 1     ACKed
    2019-12-30 19:27:05:541 CST DATA_seq: 101       ACKed
    2019-12-30 19:27:05:559 CST DATA_seq: 201       ACKed
    2019-12-30 19:27:05:574 CST DATA_seq: 301       ACKed
    2019-12-30 19:27:05:589 CST DATA_seq: 401       ACKed
    2019-12-30 19:27:05:604 CST DATA_seq: 501       ACKed
    2019-12-30 19:27:05:620 CST DATA_seq: 601       ACKed
    2019-12-30 19:27:05:635 CST DATA_seq: 701       ACKed
    2019-12-30 19:27:05:653 CST DATA_seq: 801       ACKed
```

快速重传证明



```
    2019-12-30 19:27:05:666 CST DATA_seq: 901       ACKed
    2019-12-30 19:27:05:680 CST DATA_seq: 1001  DELAY    NO_ACK
    2019-12-30 19:27:05:695 CST DATA_seq: 1101      NO_ACK
    2019-12-30 19:27:05:711 CST DATA_seq: 1201      NO_ACK
    2019-12-30 19:27:05:724 CST DATA_seq: 1301      ACKed
    2019-12-30 19:27:05:725 CST *Re: DATA_seq: 1001      NO_ACK
    2019-12-30 19:27:05:737 CST DATA_seq: 1401      ACKed
    2019-12-30 19:27:05:749 CST DATA_seq: 1501      ACKed
    2019-12-30 19:27:05:761 CST DATA_seq: 1601      ACKed
```

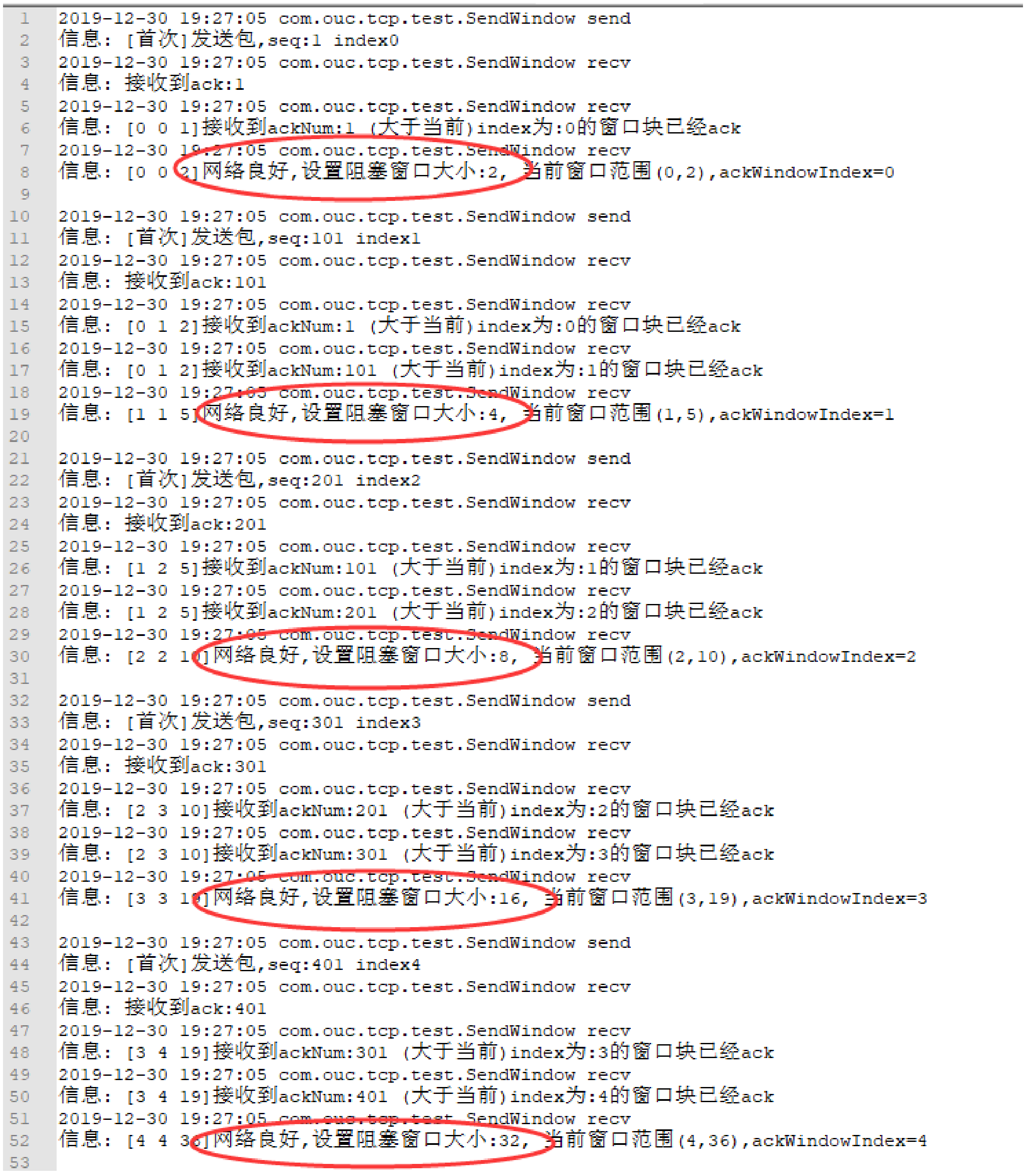
查看Log文件发现1001的包是延迟到达的,观察发送端的日志RDTSender.log 如下



```
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:801
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [7 8 42]接收到ackNum:701 (大于当前)index为:7的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [7 8 42]接收到ackNum:801 (大于当前)index为:8的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [8 8 44]网络良好,设置阻塞窗口大小:36, 当前窗口范围(8,44),ackWindowIndex=8

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:901 index9
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:901
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [8 9 44]接收到ackNum:801 (大于当前)index为:8的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [8 9 44]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 9 46]网络良好,设置阻塞窗口大小:37, 当前窗口范围(9,46),ackWindowIndex=9

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1001 index10
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1101 index11
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:901
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 11 46]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 11 47]网络良好,设置阻塞窗口大小:38, 当前窗口范围(9,47),ackWindowIndex=11

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1201 index12
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:901
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 12 47]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 12 48]网络良好,设置阻塞窗口大小:39, 当前窗口范围(9,48),ackWindowIndex=12

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1301 index13
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:901
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 48]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
警告: [9 13 26]网络拥挤,设置新门限:19,阻塞窗口大小为:17, 当前窗口范围(9,26),acknum=13

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
警告: [快重]发送包,seq::1001 index10
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:1301
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 26]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
```

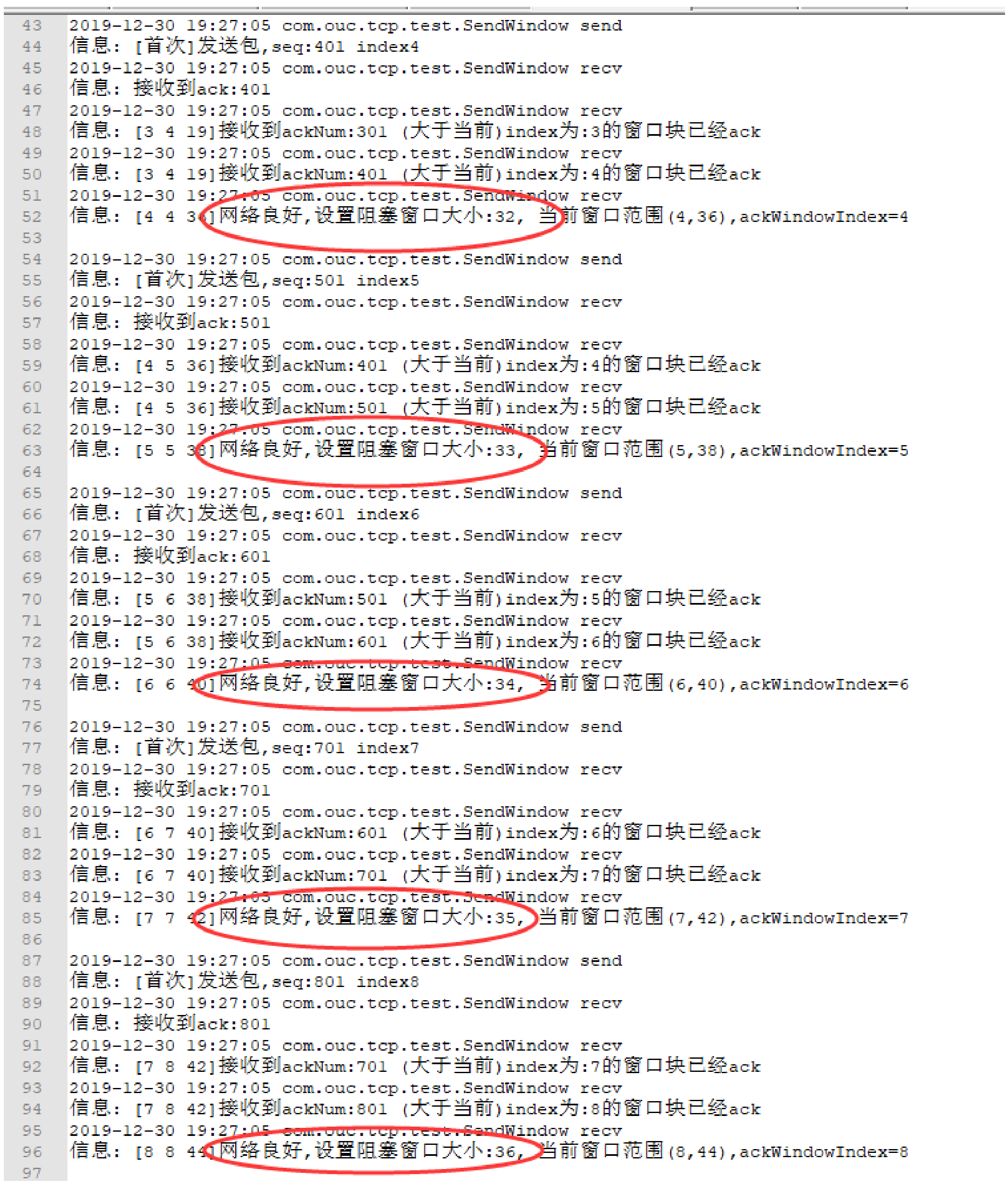
在发送第seq为1001的包后,连续收到了三个ack为901的包(因为首次发送的1001的包延迟了),此时发送端执行快速重传,重新发送1001的包,之后接收到了ack值为1301的包

慢开始证明



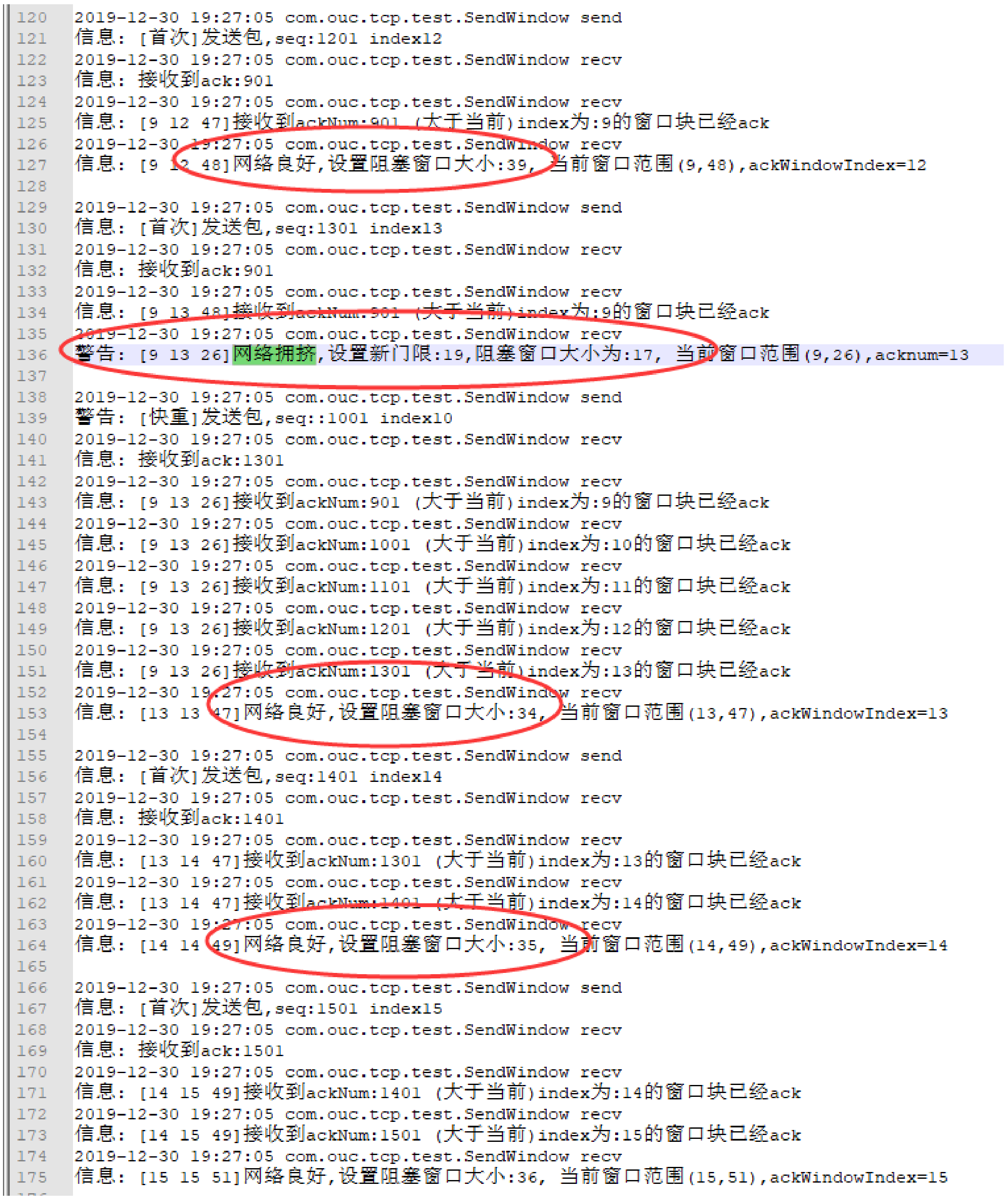
```
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1 index0
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:1
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [0 0 1]接收到ackNum:1 (大于当前)index为:0的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [0 0 2]网络良好,设置阻塞窗口大小:2, 当前窗口范围(0,2),ackWindowIndex=0

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:101 index1
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:101
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [0 1 2]接收到ackNum:1 (大于当前)index为:0的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [0 1 2]接收到ackNum:101 (大于当前)index为:1的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [1 1 5]网络良好,设置阻塞窗口大小:4, 当前窗口范围(1,5),ackWindowIndex=1

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:201 index2
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:201
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [1 2 5]接收到ackNum:101 (大于当前)index为:1的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [1 2 5]接收到ackNum:201 (大于当前)index为:2的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [2 2 10]网络良好,设置阻塞窗口大小:8, 当前窗口范围(2,10),ackWindowIndex=2

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:301 index3
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:301
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [2 3 10]接收到ackNum:201 (大于当前)index为:2的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [2 3 10]接收到ackNum:301 (大于当前)index为:3的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [3 3 19]网络良好,设置阻塞窗口大小:16, 当前窗口范围(3,19),ackWindowIndex=3

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:401 index4
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:401
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [3 4 19]接收到ackNum:301 (大于当前)index为:3的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [3 4 19]接收到ackNum:401 (大于当前)index为:4的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [4 4 36]网络良好,设置阻塞窗口大小:32, 当前窗口范围(4,36),ackWindowIndex=4
```

加法增大证明



```
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:401 index4
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:401
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [3 4 19]接收到ackNum:301 (大于当前)index为:3的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [3 4 19]接收到ackNum:401 (大于当前)index为:4的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [4 4 36]网络良好,设置阻塞窗口大小:32, 当前窗口范围(4,36),ackWindowIndex=4

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:501 index5
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:501
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [4 5 36]接收到ackNum:401 (大于当前)index为:4的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [4 5 36]接收到ackNum:501 (大于当前)index为:5的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [5 5 38]网络良好,设置阻塞窗口大小:33, 当前窗口范围(5,38),ackWindowIndex=5

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:601 index6
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:601
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [5 6 38]接收到ackNum:501 (大于当前)index为:5的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [5 6 38]接收到ackNum:601 (大于当前)index为:6的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [6 6 40]网络良好,设置阻塞窗口大小:34, 当前窗口范围(6,40),ackWindowIndex=6

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:701 index7
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:701
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [6 7 40]接收到ackNum:601 (大于当前)index为:6的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [6 7 40]接收到ackNum:701 (大于当前)index为:7的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [7 7 42]网络良好,设置阻塞窗口大小:35,当前窗口范围(7,42),ackWindowIndex=7

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:801 index8
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:801
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [7 8 42]接收到ackNum:701 (大于当前)index为:7的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [7 8 42]接收到ackNum:801 (大于当前)index为:8的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [8 8 44]网络良好,设置阻塞窗口大小:36,当前窗口范围(8,44),ackWindowIndex=8
```

拥塞避免\乘法减小证明



```
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1201 index12
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:901
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 12 47]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 12 48]网络良好,设置阻塞窗口大小:39, 当前窗口范围(9,48),ackWindowIndex=12

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1301 index13
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:901
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 48]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
警告: [9 13 26]网络拥挤,设置新门限:19,阻塞窗口大小为:17, 当前窗口范围(9,26),acknum=13

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
警告: [快重]发送包,seq::1001 index10
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:1301
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 26]接收到ackNum:901 (大于当前)index为:9的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 26]接收到ackNum:1001 (大于当前)index为:10的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 26]接收到ackNum:1101 (大于当前)index为:11的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 26]接收到ackNum:1201 (大于当前)index为:12的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [9 13 26]接收到ackNum:1301 (大于当前)index为:13的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [13 13 47]网络良好,设置阻塞窗口大小:34, 当前窗口范围(13,47),ackWindowIndex=13

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1401 index14
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:1401
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [13 14 47]接收到ackNum:1301 (大于当前)index为:13的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [13 14 47]接收到ackNum:1401 (大于当前)index为:14的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [14 14 49]网络良好,设置阻塞窗口大小:35, 当前窗口范围(14,49),ackWindowIndex=14

2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow send
信息: [首次]发送包,seq:1501 index15
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: 接收到ack:1501
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [14 15 49]接收到ackNum:1401 (大于当前)index为:14的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [14 15 49]接收到ackNum:1501 (大于当前)index为:15的窗口块已经ack
2019-12-30 19:27:05 com.ouc.tcp.test.SendWindow recv
信息: [15 15 51]网络良好,设置阻塞窗口大小:36, 当前窗口范围(15,51),ackWindowIndex=15
```

检测到网络拥挤时, 新门限的值为原来的窗口大小的1/2(原来窗口大小为39,故新门限为19);新的窗口大小设置为原来的门限大小+1(原来的门限大小为16,即新窗口大小为17).

下次接收到ack包且网络良好时,由于17小于门限19,故指数增大,新窗口大小为34