华中科技大学计算机学院
《计算机通信与网络》实验报告

班级[redacted] 姓名[redacted] 学号[redacted]

项目	Socket 编程（40%）	数据可靠传输协议设计（20%）	CPT 组网（20%）	平时成绩（20%）	总分
得分					

教师评语：

教师签名：

给分日期：

目　　录

实验二 数据可靠传输协议设计实验

1.1 环境

1.1.1 开发平台

处理器 Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
机带 RAM 16.0 GB (15.8 GB 可用)
系统类型 64 位操作系统, 基于 x64 的处理器
版本 Windows 11 家庭中文版
操作系统版本 22621.963
开发平台：Microsoft Visual Studio 2017
第三方组件：Windows SDK 10.0.17341.0

1.1.2 运行平台

处理器 Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
机带 RAM 16.0 GB (15.8 GB 可用)
系统类型 64 位操作系统, 基于 x64 的处理器
版本 Windows 11 家庭中文版
操作系统版本 22621.963

1.2 系统功能需求

可靠运输层协议实验只考虑单向传输，即：只有发送方发生数据报文，接收方仅仅接收报文并给出确认报文。

要求实现具体协议时，指定编码报文序号的二进制位数（例如 3 位二进制编码报文序号）以及窗口大小（例如大小为 4），报文段序号必须按照指定的二进制位数进行编码。

本实验包括三个级别的内容，具体包括：

实现基于 GBN 的可靠传输协议。

实现基于 SR 的可靠传输协议。

在实现 GBN 协议的基础上，根据 TCP 的可靠数据传输机制实现一个简化版 TCP，要求：

报文段格式、接收方缓冲区大小和 GBN 协议一样保持不变

报文段序号 按照报文段为单位进行编号；

单一的超时计时器，不需要估算 RTT 动态调整定时器 Timeout 参数

支持快速重传和超时重传，重传时只重传最早发送且没被确认的报文段；

确认号为收到的最后一个报文段序号；

不考虑流量控制、拥塞控制。

1.3 系统设计

窗口大小 N 为 4，二进制编码报文序号长度为 3

1.3.1 GBN 协议

1. 特点：
   1) ACK(n): 接收方对序号 n 之前包括 n 在内的所有分组进行确认——累积确认
   2) 对所有已发送但未确认的分组统一设置一个定时器
   3) 超时(n): 重传分组 n 和窗口中所有序号大于 n 的分组
   4) 失序分组:

      丢弃(不缓存)

      重发按序到达的最高序号分组的 ACK
2. 滑动窗口设计：

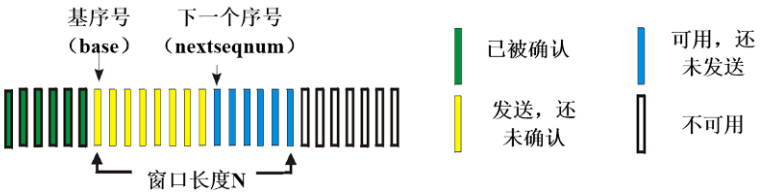


图 1-1

3. 数据包的发送和接收分为四个模块：
   1) 发送方发送分组

      若发送方正在处于等待 ack 状态，说明窗口内的分组都已经发送，但还未收到基序号的 ack，此时不发送下一分组。

      若发送方的窗口内还有未发送的分组，则向接收方发送，并将分组储存，用于超时情况的处理。如果当前发送的分组序号为基序号，启动定时器。

      若发送完，进入等待 Ack 的状态，不再发送分组。
   2) 接收方接收分组，并发送 ack

      若接收方正确接收，把分组交付给上层，向发送方发送对应序号的确认报文；

      若接收的报文序号错误或校验错误，则输出报错信息，发送上次的确认报文。
   3) 发送方接收 ack

发送方若收到正确的基序号的确认报文，关闭基序号的定时器，将窗口前推到下一个未收到的分组，停止等待 ack，此时上层如果调用 send 函数，发送方就可以向接收方发送下一个分组。如果窗口内分组还没有被全部确认，需要对序号为新的基序号分组启动计时器；

若接收的确认报文序号错误或校验错误，则输出报错信息，窗口不滑动。

4) 超时处理: 发送方重传窗口中所有序号大于等于 base 且小于 nextseqnum 的分组

1.3.2 SR 协议

1. 特点
   1) ACK(n): 接收方对接收窗口内所有分组都可以进行确认——单个确认
   2) 对所有已发送但未确认的分组分别设置定时器
   3) 超时(n): 仅重传分组 n
   4) 失序分组:

      缓存

      对失序分组进行选择性确认
2. 滑动窗口设计：

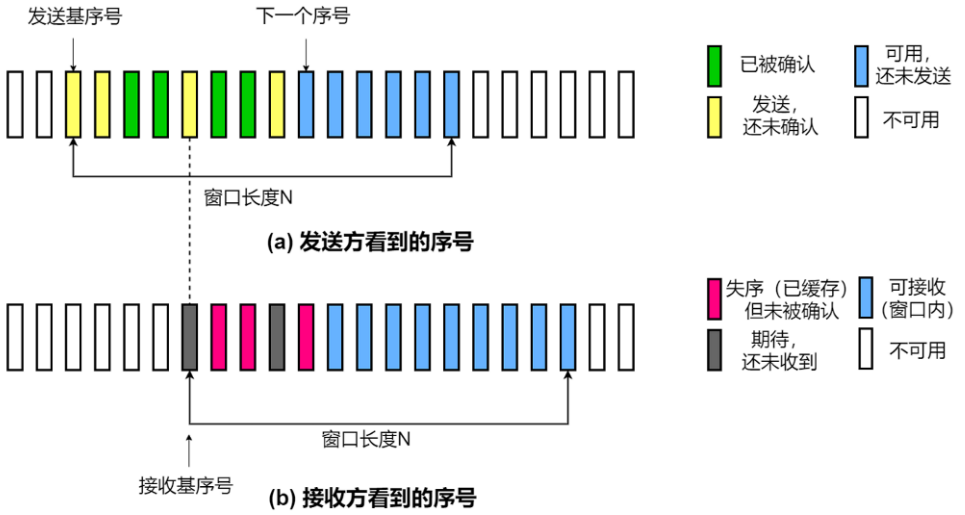


图 1-2

3. 数据包的发送和接收分为四个模块：
   1) 发送方发送分组

      若发送方正在处于等待 ack 状态，说明窗口内的分组都已经发送，但还未收到基序号的 ack，此时不发送下一分组。

      若发送方窗口内还有未发送的数据包，则向接收方按顺序发送，并将数据包储存，用于超时情况的处理。对每个发送但还未确认的数据包启动计时器。

      若窗口已满，进入等待 Ack 的状态。不再发送下一个分组。

   2) 接收方接收分组，并发送 ack

      若接收方正确接收分组，向发送方发送确认报文。接收到的分组共有 3 种情况：

a) 报文序号等于基序号：将该分组以及之前缓存的连续的失序分组一起交付给上层，将窗口前推到下一个未收到的分组

b) 报文序号大于基序号且在期望接收的下一序号前：分组失序，将其缓存到缓存区

c) 报文序号小于基序号：过时报文，不做处理；

如果接收的报文序号大于期望接收的下一序号或校验错误，则输出报错信息，不发送 ack。

3) 发送方接收 ack

发送方若收到正确的确认报文。接收到的 ack 共有 3 种情况：

a) 确认号为基序号：关闭该序号的定时器，根据记录的已经收到 ack 的序号，将窗口前推到下一个未收到的分组，停止等待 ack。

b) 确认号大于基序号：关闭该序号的定时器，记录序号，窗口不滑动

c) 确认号小于基序号或该确认号已记录：重复 Ack，不做处理

如果接收的报文序号错误或校验错误，则输出报错信息。

4) 超时处理: 重置定时器，发送方重传超时的分组

1.3.3 TCP 快速重传

作用：在超时到来之前重传报文段。

在 GBN 的基础上，如果发送方收到的 ack 序号等于上一个 ack 的序号，则开始计数。如果连续收到 3 个重复的冗余 ack，就认为该序号之后的报文段丢失，将下一序号的报文重发。

过程如图 1-3：

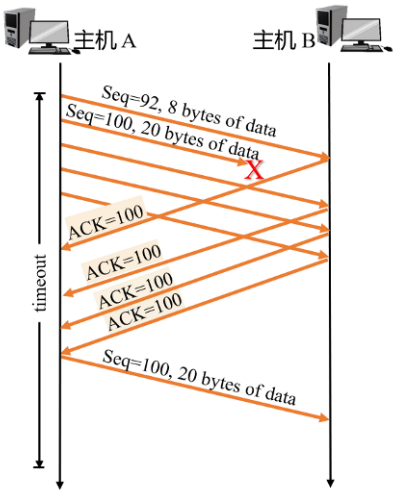


图 1-3

1.4 系统实现

系统的总体流程已在系统设计的模块划分中叙述，这里不再赘述，只列出数据结构和一些比较重要的细节，以及整体的流程图。

1.4.1 GBN 协议

定义 Seqlength=8，为两个窗口大小，所有用于储存或标记的数组都分配 Seqlength 大小的内存，足够保存运行过程中所有需要的数据。

1. 发送方数据结构设计：

   base：基序号，即最早的未确认的分组的序号，初始 0

   expectSeqNum：下一个发送序号，初始 0

   waitingState：发送方是否处于等待 Ack 的状态，初始 false

   packetWaitingAck[Seqlength]：已发送并等待 Ack 的分组（用于超时情况的处理）

2. 接收方数据结构设计：

   expectSeqNum：期待收到的下一个报文序号，初始 0

   lastAckPkt：上次发送的确认报文

3. 发送方 send 函数

   如果上层调用 send 函数时，发送方正在等待 ack，则返回 false，不继续发送分组；否则返回 true。

   对于每一个分组 packet，acknum 字段忽略，seqnum 赋值为 expectSeqNum。分组存储在 packetWaitingAck 数组中，下标为 expectSeqNum%Seqlength，用于超时处理。分组通过 sendToNetworkLayer 函数发送到接收方。

   如果 base=expectSeqNum，说明窗口里的分组都还未被发送，对序号 base 的 packet 启动定时器。

4. 接收方 receive 函数

   对于每一个正确收到的 packet，需要将其中有效的内容通过 memcpy 函数复制到 Message 中，再通过函数 delivertoAppLayer 交付到上层。

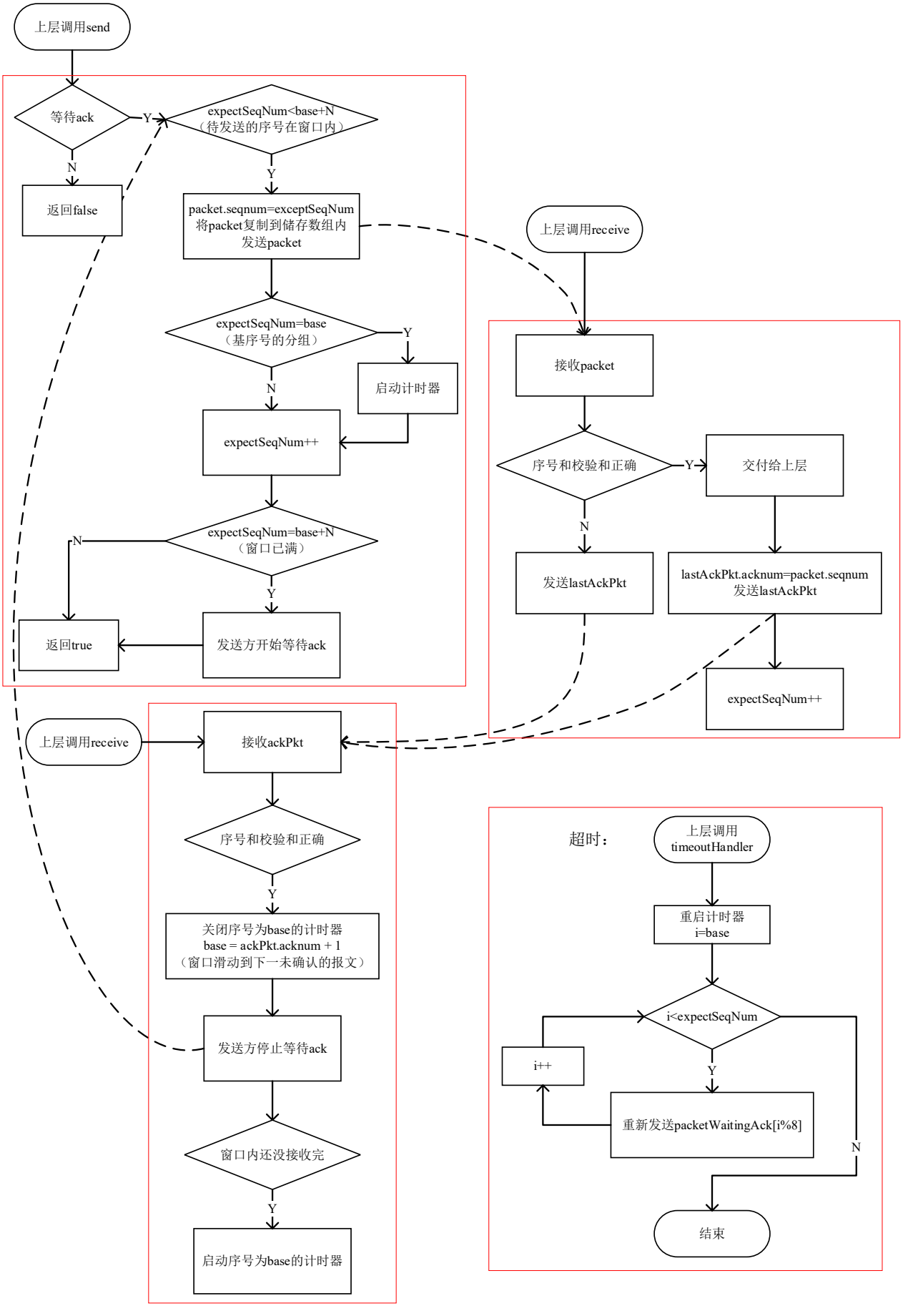
   对于每一个要发送的确认报文 lastAckPkt，acknum 赋值为正确收到的分组的 seqnum。

   如果没有正确接收，直接发送 lastAckPkt，内容与上一次发送的完全相同

5. 发送方 receive 函数

   窗口滑动后，如果窗口内的数据包还没有被全部确认，即 base!=expectSeqNum，需要将新的基序号的分组启动计时器。每一个基序号分组都需要启动计时器。

6. 流程图：左侧为发送方，右侧为超时处理和接收方。红色方框内是四个模块的流程，虚线表示各个函数之间的关联


上层调用send
等待ack
Y
N
返回false
expectSeqNum<base+N
（待发送的序号在窗口内）
Y
packet.seqnum=exceptSeqNum
将packet复制到储存数组内
发送packet
expectSeqNum=base
（基序号的分组）
Y
N
启动计时器
expectSeqNum++
expectSeqNum=base+N
（窗口已满）
N
Y
返回true
发送方开始等待ack
上层调用receive
接收packet
序号和校验和正确
Y
交付给上层
N
发送lastAckPkt
lastAckPkt.acknum=packet.seqnum
发送lastAckPkt
expectSeqNum++
上层调用receive
接收ackPkt
序号和校验和正确
Y
关闭序号为base的计时器
base = ackPkt.acknum + 1
（窗口滑动到下一未确认的报文）
发送方停止等待ack
窗口内还没接收完
Y
启动序号为base的计时器
超时：
上层调用
timeoutHandler
重启计时器
i=base
i<expectSeqNum
Y
i++
重新发送packetWaitingAck[i%8]
N
结束

1.4.2 SR 协议

1. 发送方数据结构设计：

   base：发送方基序号，即最早的未确认的分组的序号，初始 0

   expectSeqNum：下一个发送序号，初始 0

   waitingState：是否处于等待 Ack 的状态，初始 false

   packetWaitingAck[Seqlength]：已发送并等待 Ack 的数据包

   AckFlag[Seqlength]：发送的分组是否已收到正确的 Ack 的标记，下标与 packetWaitingAck 对应

2. 接收方数据结构设计：

   base：接收方基序号，即最早的未收到的分组的序号，初始 0

   expectSeqNum：下一个接收序号，初始 N

   lastAckPkt：确认报文

   ReceivedPacket[Seqlength]：接收方的缓存区

   packetFlag [Seqlength]：是否已缓存失序的报文的标记，下标与 ReceivedPacket 对应

3. 发送方 send 函数

   与 GBN 不同的是，需要对所有已发送但未确认的分组分别设置定时器。

4. 接收方 receive 函数

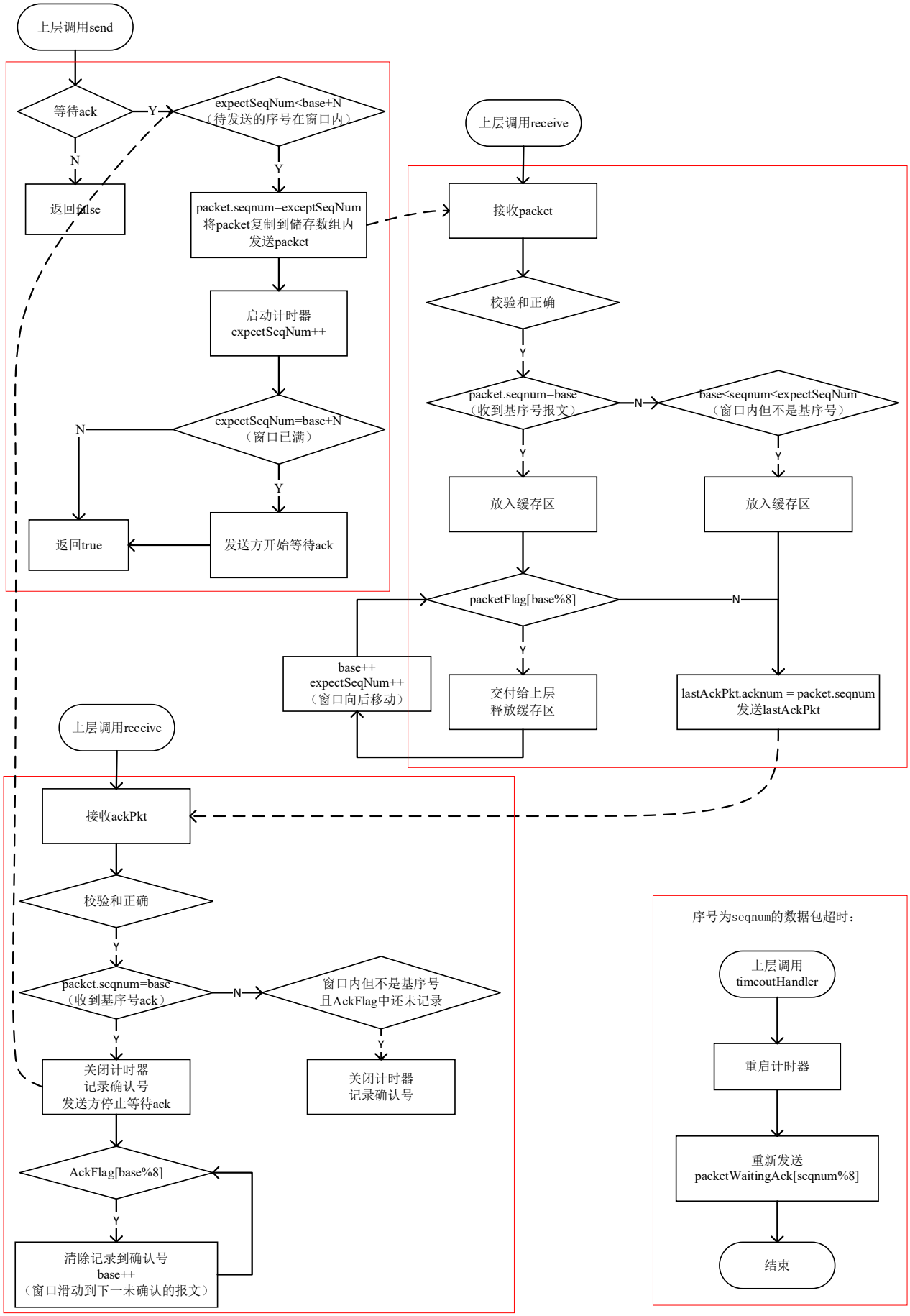
   对于每一个接收到的窗口内的 packet，将其储存到 ReceivedPacket 数组即缓存区中，下标为 packet.seqnum%Seqlength，并把相同下标的 packetFlag 赋值为 true，用于标记已缓存的分组；收到基序号分组则和所有连续的失序分组一起交付到上层，然后把所有交付的序号对应的 packetFlag 赋值为 false，释放缓存区。

   对于所有接收到的校验和正确的 packet，都需要向发送方发送确认报文，即使是过时报文。

5. 发送方 receive 函数

   对于每一个接收到的窗口内的 ackPkt，关闭该序号计时器，并将下标为 ackPkt.acknum%Seqlength 的 AckFlag 置 true，作为已收到 ack 的标记。当收到基序号的 ack 时，将下一未确认序号之前的 AckFlag 全部置 false，删去标记。

6. 流程图：左侧为发送方，右侧为超时处理和接收方。


上层调用send
等待ack
Y
N
expectSeqNum<base+N
（待发送的序号在窗口内）
Y
返回false
packet.seqnum=exceptSeqNum
将packet复制到储存数组内
发送packet
启动计时器
expectSeqNum++
expectSeqNum=base+N
（窗口已满）
N
Y
返回true
发送方开始等待ack
上层调用receive
接收packet
校验和正确
Y
packet.seqnum=base
（收到基序号报文）
N
base<seqnum<expectSeqNum
（窗口内但不是基序号）
Y
Y
放入缓存区
放入缓存区
packetFlag[base%8]
N
Y
base++
expectSeqNum++
（窗口向后移动）
交付给上层
释放缓存区
lastAckPkt.acknum = packet.seqnum
发送lastAckPkt
上层调用receive
接收ackPkt
校验和正确
Y
packet.seqnum=base
（收到基序号ack）
N
窗口内但不是基序号
且AckFlag中还未记录
Y
Y
关闭计时器
记录确认号
发送方停止等待ack
关闭计时器
记录确认号
AckFlag[base%8]
Y
清除记录到确认号
base++
（窗口滑动到下一未确认的报文）
序号为seqnum的数据包超时：
上层调用
timeoutHandler
重启计时器
重新发送
packetWaitingAck[seqnum%8]
结束

1.4.3 TCP 快速重传

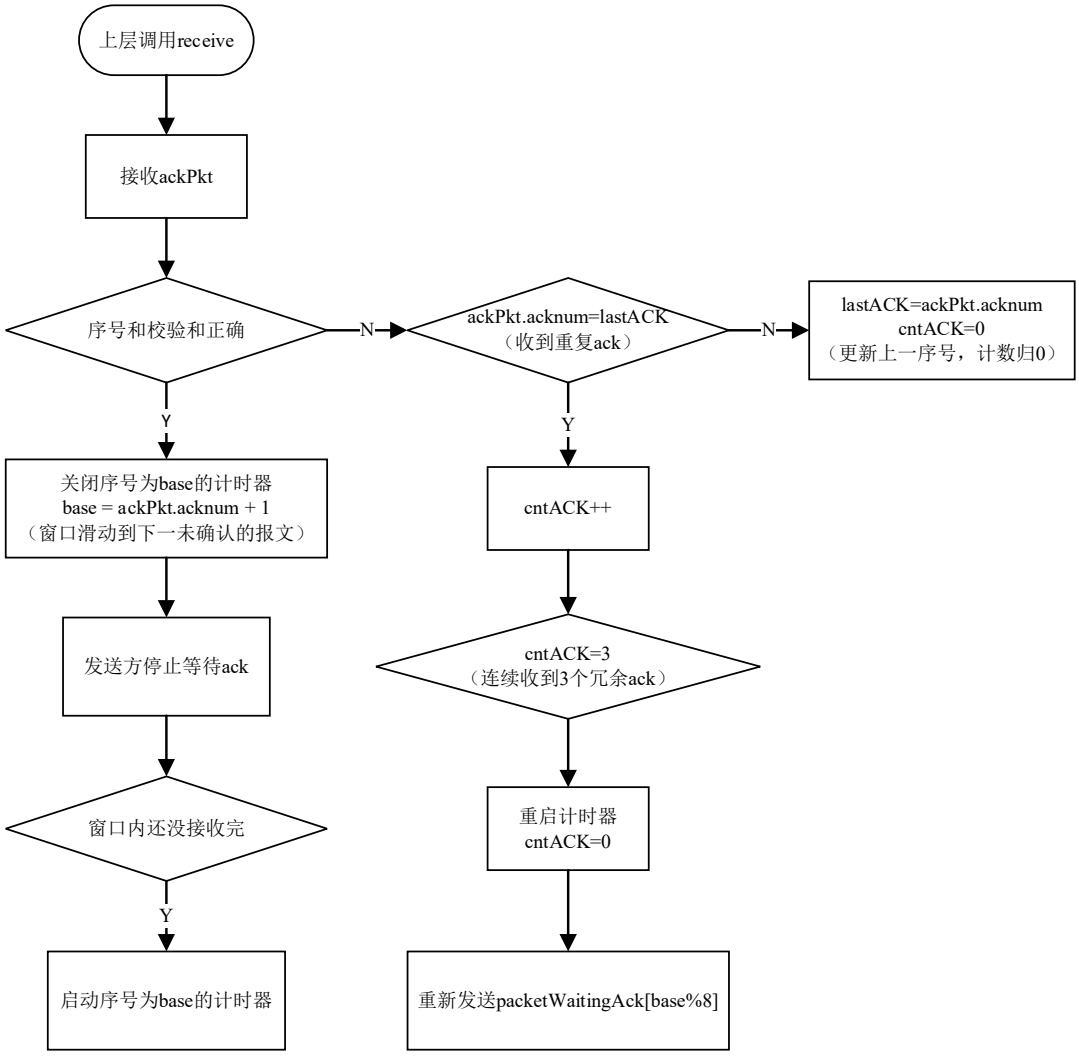
1. 发送方数据结构设计在 GBN 的基础上加了两个变量：

   lastACK：上个 Ack 的序号，初始 0

   cntACK：连续冗余 Ack 的个数，初始 0
2. 发送方数据结构设计与 GBN 相同：
3. 发送方 send 函数与 GBN 一致
4. 接收方 receive 函数与 GBN 一致
5. 发送方 receive 函数

   如果没有收到冗余 ack，则更新 lastACK，计数清零，否则计数加一。

   计数到 3，就重新发送序号为 lastACK+1 的数据包，计数清零
6. 流程图：因为发送方 send 和接收方 receive 函数与 GBN 完全相同，只显示发送方 receive 的流程



1.5 系统测试及结果说明

通过给定的检查脚本来测试代码

1.5.1 GBN 协议

运行 check_win10 次，内容均相同：

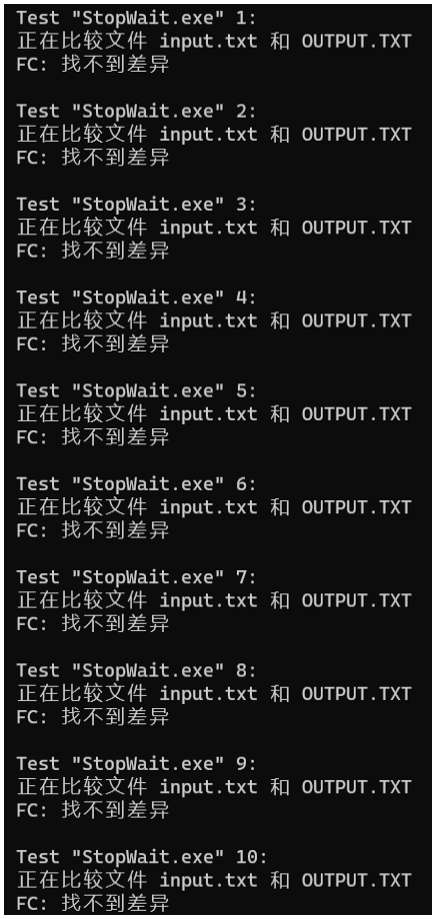


图 1-4

输出结果分析：

1 号分组丢包，接收方在收到正确的 1 号分组之前，始终发送 0 号的确认报文

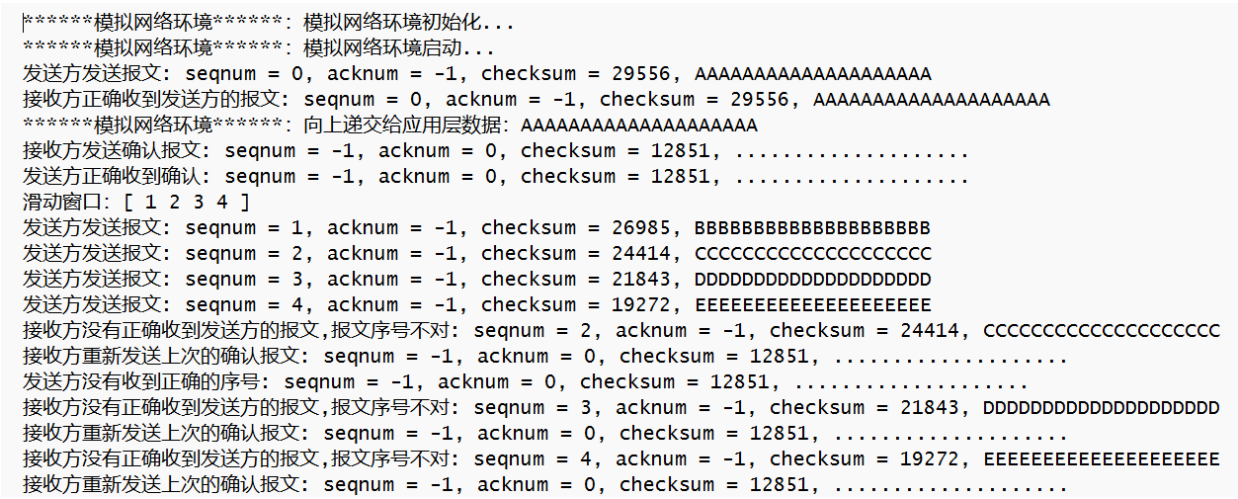


图 1-5

1 号分组的计数器超时，此时下一发送序号为 5，将从序号 1~4 的分组重发。

```
发送方没有收到正确的序号: seqnum = -1, acknum = 0, checksum = 12851, ....................
发送方定时器时间到，重发报文: seqnum = 1, acknum = -1, checksum = 26985, BBBBBBBBBBBBBBBBBBBB
发送方定时器时间到，重发报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
发送方定时器时间到，重发报文: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDD
发送方定时器时间到，重发报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEEE
```

图 1-6

1 号分组的计数器超时，此时下一发送序号为 5，将从序号 1~4 的分组重发。

正确收到 1 号分组后，窗口开始向右滑动

```
接收方正确收到发送方的报文: seqnum = 1, acknum = -1, checksum = 26985, BBBBBBBBBBBBBBBBBBBB
******模拟网络环境******: 向上递交给应用层数据: BBBBBBBBBBBBBBBBBBBB
接收方发送确认报文: seqnum = -1, acknum = 1, checksum = 12850, ....................
接收方正确收到发送方的报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
******模拟网络环境******: 向上递交给应用层数据: CCCCCCCCCCCCCCCCCCCC
接收方发送确认报文: seqnum = -1, acknum = 2, checksum = 12849, ....................
发送方正确收到确认: seqnum = -1, acknum = 1, checksum = 12850, ....................
滑动窗口: [ 2 3 4 5 ]
发送方正确收到确认: seqnum = -1, acknum = 2, checksum = 12849, ....................
滑动窗口: [ 3 4 5 6 ]
```

图 1-7

1.5.2 SR 协议

运行 check_win10 次，内容均相同：

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

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

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

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

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

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

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

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

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

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

图 1-8

输出结果分析：

9 号分组校验错误，将窗口内的正确接收的 10、11、12 号的失序分组缓存。可以看到此时 10 号分组已被确认。

```
发送方发送报文: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJ
序列9发送方启动计时器
发送方发送报文: seqnum = 10, acknum = -1, checksum = 3846, KKKKKKKKKKKKKKKKKKK
序列10发送方启动计时器
发送方发送报文: seqnum = 11, acknum = -1, checksum = 1275, LLLLLLLLLLLLLLLLLLL
序列11发送方启动计时器
发送方发送报文: seqnum = 12, acknum = -1, checksum = 64239, MMMMMMMMMMMMMMMMMMM
序列12发送方启动计时器
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 9, acknum = -1, checksum = 6417, KJJJJJJJJJJJJJJJJJJJ
接收方已缓存发送方的报文: seqnum = 10, acknum = -1, checksum = 3846, KKKKKKKKKKKKKKKKKKK
接收方发送确认报文: seqnum = -1, acknum = 10, checksum = 12841, ....................
接收方已缓存发送方的报文: seqnum = 11, acknum = -1, checksum = 1275, LLLLLLLLLLLLLLLLLLL
接收方发送确认报文: seqnum = -1, acknum = 11, checksum = 12840, ....................
发送方正确收到确认: seqnum = -1, acknum = 10, checksum = 12841, ....................
接收方已缓存发送方的报文: seqnum = 12, acknum = -1, checksum = 64239, MMMMMMMMMMMMMMMMMMM
接收方发送确认报文: seqnum = -1, acknum = 12, checksum = 12839, ....................
```

图 1-9

对于超时的分组，只重发其本身。

```
发送方定时器时间到，重发报文: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJ
发送方定时器时间到，重发报文: seqnum = 11, acknum = -1, checksum = 1275, LLLLLLLLLLLLLLLLLLL
发送方定时器时间到，重发报文: seqnum = 12, acknum = -1, checksum = 64239, MMMMMMMMMMMMMMMMMMM
```

图 1-10

等到基序号 9 的分组接收成功，把失序的分组一起递交到上层。

```
发送方正确收到确认: seqnum = -1, acknum = 12, checksum = 12839, ....................
接收方正确收到发送方的报文: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJ
******模拟网络环境******: 向上递交给应用层数据: JJJJJJJJJJJJJJJJJJJ
******模拟网络环境******: 向上递交给应用层数据: KKKKKKKKKKKKKKKKKKK
******模拟网络环境******: 向上递交给应用层数据: LLLLLLLLLLLLLLLLLLL
******模拟网络环境******: 向上递交给应用层数据: MMMMMMMMMMMMMMMMMMM
```

图 1-11

发送方正确收到基序号 9 的确认报文后，因为 10 号报文之前已经确认，窗口向右滑动到 11。

```
接收方发送确认报文: seqnum = -1, acknum = 9, checksum = 12842, ....................
接收方正确收到已确认的过时报文: seqnum = 11, acknum = -1, checksum = 1275, LLLLLLLLLLLLLLLLLLL
接收方发送确认报文: seqnum = -1, acknum = 11, checksum = 12840, ....................
发送方正确收到确认: seqnum = -1, acknum = 9, checksum = 12842, ....................
滑动窗口: [ 11 12 13 14 ]
```

图 1-12

1.5.3 TCP 快速重传

运行 check_win10 次，内容均相同：

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

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

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

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

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

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

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

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

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

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

图 1-13

输出结果分析：

发送方已经收到了正确的 8 号 ack

```
发送方正确收到确认: seqnum = -1, acknum = 8, checksum = 12843, .....................
滑动窗口: [ 9 10 11 12 ]
```

图 1-14

因为 9 号分组的数据校验错误，接收方一直发送 8 号的 ack。

```
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 5, acknum = -1, checksum = 16701, FFFFFFFFFFFFFFFFFFF
接收方重新发送上次的确认报文: seqnum = -1, acknum = 8, checksum = 12843, .....................
发送方没有收到正确的序号: seqnum = -1, acknum = 8, checksum = 12843, .....................
发送方发送报文: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJJ
发送方启动计时器
发送方发送报文: seqnum = 10, acknum = -1, checksum = 3846, KKKKKKKKKKKKKKKKKKK
发送方发送报文: seqnum = 11, acknum = -1, checksum = 1275, LLLLLLLLLLLLLLLLLLL
发送方发送报文: seqnum = 12, acknum = -1, checksum = 64239, MMMMMMMMMMMMMMMMMMM
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 9, acknum = -1, checksum = 6417, KJJJJJJJJJJJJJJJJJJJ
接收方重新发送上次的确认报文: seqnum = -1, acknum = 8, checksum = 12843, .....................
发送方没有收到正确的序号: seqnum = -1, acknum = 8, checksum = 12843, .....................
```

图 1-15

连续收到 3 次序号重复的确认报文后，快速重传 9 号报文，接收到正确的 9 号 ack 后，窗口向右滑动

```
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 9, acknum = -1, checksum = 6417, KJJJJJJJJJJJJJJJJJJ
接收方重新发送上次的确认报文: seqnum = -1, acknum = 8, checksum = 12843, ....................
发送方没有收到正确的序号: seqnum = -1, acknum = 8, checksum = 12843, ....................
------收到1个冗余的ACK------
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 11, acknum = -1, checksum = 1275, LLLLLLLLLLLLLLLLLLL
接收方重新发送上次的确认报文: seqnum = -1, acknum = 8, checksum = 12843, ....................
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 12, acknum = -1, checksum = 64239, MMMMMMMMMMMMMMMMMMM
接收方重新发送上次的确认报文: seqnum = -1, acknum = 8, checksum = 12843, ....................
发送方定时器时间到，重发报文: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJ
发送方没有收到正确的序号: seqnum = -1, acknum = 8, checksum = 12843, ....................
------收到2个冗余的ACK------
接收方正确收到发送方的报文: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJ
******模拟网络环境******: 向上递交给应用层数据: JJJJJJJJJJJJJJJJJJJ
接收方发送确认报文: seqnum = -1, acknum = 9, checksum = 12842, ....................
发送方没有收到正确的序号: seqnum = -1, acknum = 8, checksum = 12843, ....................
------收到了3个冗余的ACK，快速重传序号9------
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 9, acknum = -1, checksum = 6417, JJJJJJJJJJJJJJJJJJJ
接收方重新发送上次的确认报文: seqnum = -1, acknum = 9, checksum = 12842, ....................
发送方正确收到确认: seqnum = -1, acknum = 9, checksum = 12842, ....................
滑动窗口: [ 10 11 12 13 ]
```

图 1-16

1.6 其它需要说明的问题

无

1.7 参考文献

[1]James,F.Kurose,Keith,W.Ross.计算机网络(原书第 7 版)自顶向下方法[M].北京:机械工业出版社,2018:152-187

[2]模块二 可靠数据传输协议设计(修订版 V2020)[Z]

心得体会与建议

2.1 心得体会

在学习这门课之前，我对计算机网络没有什么了解，只觉得网络是大多数人生活中必不可少的部分，尤其是网不好感到很暴躁的时候，更能体会到它的重要性。学了理论课程后之后，了解了很多计算机网络中的原理，计算机网络中的结构层次以及协议都非常复杂也非常严谨，简直无法想象最先搭建这个系统的人花了多少精力和时间。一个网页打开的过程很复杂，需要用到各种协议，发送各种请求，我在准备考试的时候都背了很久，但我们在使用网络的过程中，等待的时间却可能只有一秒钟，真是非常神奇的事情。但由于各种硬件和软件上的限制，网络也是非常脆弱的，如果一个网页打不开，可能的原因也有无数个。

实验课是对理论课的加深和巩固。实验一和实验二的过程中，看了任务书感觉很难很复杂，代码量很大，但是好在提供了可使用的代码。先理解老师提供的代码和框架，再在其基础上完成实验的要求，难度就降低了很多。实验三的组网实验是很有趣的实验，模拟现实网络并实现各种通信要求，配置成功后非常有成就感。实验检查时助教也很认真负责，这门实验课算是收获颇多。

2.2 建议

理论课多一些对代码的理解的内容，尤其是 socket 编程和运输层协议，理论课和实验课的衔接可能不太到位，实验一和实验二刚开始的时候确实是一头雾水，完全是在自学。