

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议-停等协议的设计与实现&可靠数据传输协议-GBN 协议的设计与实现 | | | | | |
| 姓名 | 瞿久尧 | | 院系 | 计算学部 | | |
| 班级 | 2037101 | | 学号 | 120L022314 | | |
| 任课教师 | 李全龙 | | 指导教师 | 李全龙 | | |
| 实验地点 | 格物207 | | 实验时间 | 2022.10.14 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| 可靠数据传输协议-停等协议的设计与实现：  理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。  可靠数据传输协议-GBN 协议的设计与实现：  理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于UDP 设计并实现一个 GBN 协议的过程与技术。 |
| 实验内容： |
| 可靠数据传输协议-停等协议的设计与实现：  1) 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服  务器到客户的数据传输）。  2) 模拟引入数据包的丢失，验证所设计协议的有效性。  3) 改进所设计的停等协议，支持双向数据传输；  4）基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。  可靠数据传输协议-GBN 协议的设计与实现：  1) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服  务器到客户的数据传输）。  2) 模拟引入数据包的丢失，验证所设计协议的有效性。  3) 改进所设计的 GBN 协议，支持双向数据传输；  4）将所设计的 GBN 协议改进为 SR 协议。 |
| 实验过程： |
| 注：停等协议实际上为窗口尺寸为1的GBN，故只验收GBN的流程。  1.GBN协议  在程序实现中，双方并行运行，在套接字存活期间，周期地进入主循环，非阻塞地对对方传过来的数据报进行处理，并对无动作循环（即没有接收到ACK的一个循环）进行计数，进行超时判断。  发送方流程图：    接收方流程图：    2.SN协议  SR实现时，需要对单个分组记超时时间，并在超时后重传该单个分组，因此可以对单个窗口置计数器，对循环内没有发生动作的窗口均计数加一。  发送方流程图：    接收方流程图：    3.报文结构规约  发送方和接收方对于报文结构作出了约定，规定报文结构为形似:  [seq\_num,data]的二进制字节流，seq\_num用于标记发送的报文序号。  seq\_num为0的字节流[0,ack\_num(exp\_num-1)]，用于接收方的ACK报文（全双工设计）。  [0 0]报文用于发送方告知数据发送结束，接收方告知已知晓发送结束，双方确认结束后,会结束程序运行。  4.超时计算方法  超时逻辑仅与发送方有关。  GBN：  在一次主循环内，首先执行发送逻辑，判断当前是否有可用窗口，然后非阻塞地接收用户传回的ACK（select方法）：    解析ACK内容，如果在该次循环没有收到ACK，则将超时标记加一。    超时标记超过限制时，触发超时重传，重传所有空中分组.  SR：  不同处在于，SR将全局超时标记改为每个窗口拥有一个单独的超时标记，如果一个已发送窗口在单次循环中没有收到其指定的ACK，则其超时标记加一，当某个窗口超过限制时，重传该窗口的分组。  5.模拟包丢失  事实上，由于在本次实验中采用了本机对本机的socket接口来模拟数据传输，基本不存在自然的丢包行为。  我们可以采用概率发送的方式，来模拟丢包的行为，在每次发送包之前，调用一次Random方法，当取得的数大于n时，才对包进行发送，则可以模拟发生几率为n的包丢失。    6.双向数据传输  本次实验实现了半双工的C/S模式,在一次传输中，双方持有固定的客户端或服务器身份，在一次文件传输结束后，可以重新确定身份，传输下一个文件。 |
| 实验结果： |
| 可靠数据传输协议-GBN 协议的设计与实现：  窗口大小设为4。  发送成功：  IMG_256      模拟丢包：  IMG_256  双向：  IMG_256        SR：  IMG_256 |
| 问题讨论： |
| 对实验过程中的思考问题进行讨论或回答。  无。 |
| 心得体会： |
| 1. 本次实验原定使用全双工的双向数据传输，但有如下的两个问题： 2. 在本地传输的前提下，全双工在一次循环中会浪费两倍于半双工的计算资源，并且由于本地充当客户端和服务器端会出现重复调用同一PORT的错误，双方需要分时地访问PORT，因此需要引入全局的分时，而这实际极大减缓了程序运行(在半双工中由于客户端即时地接收和处理，基本可以忽略其运行时间) 3. 为了防止接口资源流失，PORT应该在调用后及时关闭，然而在全双工中，由于无法及时判定当前PORT是否处于使用状态(报文不是连续不断到达的)，因此不能及时关闭Socket接口，而半双工则可以在一次文件传输结束后立即关闭PORT，直到下一次需要时再打开。 4. 采用超时机制，对于丢包的反应是缓慢的(依靠服务器端的超时)，而且超时时间需要根据链路状态不断地更新，容易因为过早的超时而浪费资源，而且一次丢包就需要进行大量的重发，而且对链路情况不知情的服务器端进行大量重发，很容易又会加重链路的负担。TCP的“快速重传”机制兼顾了二者，是很优秀的发明。 |

附录：程序源代码

**单向:gbn.py**

import random

import select

import socket

class util:

@staticmethod

def mk\_pkt(seqSen, seqRec, msg):

return (str(seqSen) + ' ' + str(seqRec) + ' ' + str(msg)).encode('gb2312', 'ignore')

@staticmethod

def mk\_pkt1(seq, msg):

return (str(seq) + ' ' + str(msg)).encode('gb2312', 'ignore')

class GBN:

def \_\_init\_\_(self,

local\_address: tuple[[str, int]],

remote\_address: tuple[[str, int]],

inputWindow\_size: int = 4,

inputPkg\_size: int = 1024,

hostNameInput: str = ''):

self.hostname = hostNameInput

self.fin\_flag = 0

self.pkg\_size = inputPkg\_size # 单格窗口大小

self.window\_size = inputWindow\_size # 窗口尺寸

self.send\_base = 0 # 发送窗口的最左序号

self.next\_seq = 0 # 当前未被利用的序号

self.time\_count = 0 # 记录当前传输时间

self.time\_out = 5 # 设置超时时间

self.local\_address = local\_address # 设置本地socket地址

self.remote\_address = remote\_address # 设置远程socket地址

self.data = [] # 缓存发送数据

self.read\_path = 'file/gbn/' + self.hostname + '2read.txt' # 需要发送的源文件数据

self.ack\_buf\_size = 1678

self.get\_data\_from\_file()

self.data\_buf\_size = 1678 # 作为客户端接收数据缓存

self.exp\_seq = 0 # 当前期望收到该序号的数据

self.save\_path = 'file/gbn/' + self.hostname + '2save.txt' # 接收数据时，保存数据的地址

self.write\_data\_to\_file('', mode='w')

self.pkt\_loss = 0.1 # 发送数据丢包率

self.ack\_loss = 0.1 # 返回的ack丢包率

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

self.socket.bind(self.local\_address)

self.state\_flag = 0

def send\_data(self):

if self.send\_base == len(self.data): # data数据已全部发送并收到ack

self.socket.sendto(util.mk\_pkt(0, self.exp\_seq - 1, 0), self.remote\_address)

print('服务器: ' + self.hostname + '\t' + '发送完毕!')

return

if self.next\_seq == len(self.data): # data数据已全部发送，无需发送

return

if self.next\_seq - self.send\_base < self.window\_size: # 窗口中仍有可用空间

if random.random() > self.pkt\_loss: # 随机产生丢包行为

self.socket.sendto(util.mk\_pkt(self.next\_seq, self.exp\_seq - 1, self.data[self.next\_seq]),

self.remote\_address)

isLose = 0

else:

isLose = 1

print('服务器: ' + self.hostname + '\t' + '成功发送数据: ' + str(self.next\_seq)

+ ' ack=' + str(self.exp\_seq - 1), end='\n')

if isLose == 1:

print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(self.next\_seq))

self.next\_seq = self.next\_seq + 1

else: # 窗口中无可用空间

print('服务器: ' + self.hostname + '\t' + '窗口已满，暂不发送数据', end='\n')

# 超时处理函数：计时器置0

def handle\_time\_out(self):

print('服务器: ' + self.hostname + '\t' + '数据包: ' + str(self.send\_base) + '超时，开始重传!', end='\n')

self.time\_count = 0 # 超时计次重启

for i in range(self.send\_base, self.next\_seq): # 发送空中的所有分组

if random.random() > self.pkt\_loss: # 概率性重传

self.socket.sendto(util.mk\_pkt(i, self.exp\_seq - 1, self.data[i]), self.remote\_address)

isLose = 0

else:

isLose = 1

print('服务器: ' + self.hostname + '\t' + '数据: ' + str(i)

+ ' ack=' + str(self.exp\_seq - 1) + '已重发!', end='\n')

if isLose == 1:

print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(i), end='\n')

# 从文本中读取数据用于模拟上层协议数据的到来

def get\_data\_from\_file(self):

f = open(self.read\_path, 'r', encoding='gb2312')

while True:

send\_data = f.read(self.pkg\_size)

if len(send\_data) <= 0:

break

self.data.append(send\_data) # 将读取到的数据保存到data数据结构中

# 线程执行函数，不断发送数据并接收ACK报文做相应的处理

def server\_run(self):

self.state\_flag = 1

rcv\_finish = 0

send\_finish = 0

while True:

self.send\_data() # 发送数据逻辑

readable = select.select([self.socket], [], [], 1)[0]

if len(readable) > 0: # 接收到数据

rcv\_data = self.socket.recvfrom(self.data\_buf\_size)[0].decode('gb2312')

rcv\_seq = rcv\_data.split()[0] # 按照格式规约获取数据序号

rcv\_ack = rcv\_data.split()[1]

rcv\_data = rcv\_data.replace(rcv\_seq + ' ' + rcv\_ack + ' ', '') # 按照格式规约获取数据

print('服务器: ' + self.hostname + '\t' + '接收到数据: ' + rcv\_seq + ' ack: ' + rcv\_ack, end='\n')

if int(rcv\_seq) == self.exp\_seq: # 接收到按序数据包

print('服务器: ' + self.hostname + '\t' + '接收到期望序号数据: ' + rcv\_seq, end='\n')

self.write\_data\_to\_file(rcv\_data) # 保存服务器端发送的数据到本地文件中

self.exp\_seq = self.exp\_seq + 1 # 期望数据的序号更新

elif rcv\_seq == '0' and rcv\_data == '0': # 接收到结束包

print('服务器: ' + self.hostname + '\t' + '接收数据结束', end='\n')

rcv\_finish = 1

else:

print('服务器: ' + self.hostname + '\t' + '接收到非期望数据，期望: ' +

str(self.exp\_seq) + '实际:' + str(rcv\_seq), end='\n')

if int(rcv\_ack) < self.send\_base:

self.time\_count += 1

else:

self.send\_base = int(rcv\_ack) + 1 # 滑动窗口的起始序号

self.time\_count = 0

print('服务器: ' + self.hostname + '\t' + 'self.send\_base=' + str(self.send\_base), end='\n')

else: # 未收到包

self.time\_count += 1 # 超时计次+1

if self.time\_count > self.time\_out: # 触发超时重传操作

self.handle\_time\_out()

if self.send\_base == len(self.data): # 判断数据是否传输结束

self.socket.sendto(util.mk\_pkt(0, self.exp\_seq - 1, 0), self.remote\_address) # 发送结束报文

print('服务器: ' + self.hostname + '\t' + '发送完毕', end='\n')

send\_finish = 1

if rcv\_finish == 1 and send\_finish == 1:

print('服务器: ' + self.hostname + '\t' + '发送接收均完毕!!', end='\n')

break

self.state\_flag = 0

# 保存来自服务器的合适的数据

def write\_data\_to\_file(self, data, mode='a'):

with open(self.save\_path, mode, encoding='gb2312') as f:

f.write(data) # 模拟将数据交付到上层

def isHostAlive(self):

return self.state\_flag

def shut\_socket(self):

self.socket.close()

**main.py**

import threading

from time import sleep

import gbn

def run\_gbn(local: tuple[[str, int]],

remote: tuple[[str, int]],

window\_num: int = 4,

window\_size: int = 1024):

threading.Lock()

host\_1 = gbn.GBN(local, remote, window\_num, window\_size)

host\_2 = gbn.GBN(remote, local, window\_num, window\_size)

client = threading.Thread(target=host\_1.client\_run, args=())

server = threading.Thread(target=host\_2.server\_run, args=())

server.start()

client.start()

while host\_1.isHostAlive() or host\_2.isHostAlive():

sleep(0.2)

host\_1.shut\_socket()

host\_2.shut\_socket()

#print('停等协议:\n')

#run\_gbn(('127.0.0.1', 5000), ('127.0.0.1', 5001), 1, 10)

print('gbn:\n')

run\_gbn(('127.0.0.1', 5000), ('127.0.0.1', 5001), 4, 10)

**双向gbn.py**

import random

import select

import socket

class util:

@staticmethod

def mk\_pkt(seqSen, seqRec, msg):

return (str(seqSen) + ' ' + str(seqRec) + ' ' + str(msg)).encode('gb2312', 'ignore')

@staticmethod

def mk\_pkt1(seq, msg):

return (str(seq) + ' ' + str(msg)).encode('gb2312', 'ignore')

class GBN:

def \_\_init\_\_(self,

local\_address: tuple[[str, int]],

remote\_address: tuple[[str, int]],

inputWindow\_size: int = 4,

inputPkg\_size: int = 1024,

hostNameInput: str = ''):

self.hostname = hostNameInput

self.fin\_flag = 0

self.pkg\_size = inputPkg\_size # 单格窗口大小

self.window\_size = inputWindow\_size # 窗口尺寸

self.send\_base = 0 # 发送窗口的最左序号

self.next\_seq = 0 # 当前未被利用的序号

self.time\_count = 0 # 记录当前传输时间

self.time\_out = 5 # 设置超时时间

self.local\_address = local\_address # 设置本地socket地址

self.remote\_address = remote\_address # 设置远程socket地址

self.data = [] # 缓存发送数据

self.read\_path = 'file/gbn/' + self.hostname + '2read.txt' # 需要发送的源文件数据

self.ack\_buf\_size = 1678

self.get\_data\_from\_file()

self.data\_buf\_size = 1678 # 作为客户端接收数据缓存

self.exp\_seq = 0 # 当前期望收到该序号的数据

self.save\_path = 'file/gbn/' + self.hostname + '2save.txt' # 接收数据时，保存数据的地址

self.write\_data\_to\_file('', mode='w')

self.pkt\_loss = 0.1 # 发送数据丢包率

self.ack\_loss = 0.1 # 返回的ack丢包率

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

self.socket.bind(self.local\_address)

self.state\_flag = 0

def send\_data(self):

if self.send\_base == len(self.data): # data数据已全部发送并收到ack

self.socket.sendto(util.mk\_pkt(0, self.exp\_seq - 1, 0), self.remote\_address)

print('服务器: ' + self.hostname + '\t' + '发送完毕!')

return

if self.next\_seq == len(self.data): # data数据已全部发送，无需发送

return

if self.next\_seq - self.send\_base < self.window\_size: # 窗口中仍有可用空间

if random.random() > self.pkt\_loss: # 随机产生丢包行为

self.socket.sendto(util.mk\_pkt(self.next\_seq, self.exp\_seq - 1, self.data[self.next\_seq]),

self.remote\_address)

isLose = 0

else:

isLose = 1

print('服务器: ' + self.hostname + '\t' + '成功发送数据: ' + str(self.next\_seq)

+ ' ack=' + str(self.exp\_seq - 1), end='\n')

if isLose == 1:

print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(self.next\_seq))

self.next\_seq = self.next\_seq + 1

else: # 窗口中无可用空间

print('服务器: ' + self.hostname + '\t' + '窗口已满，暂不发送数据', end='\n')

# 超时处理函数：计时器置0

def handle\_time\_out(self):

print('服务器: ' + self.hostname + '\t' + '数据包: ' + str(self.send\_base) + '超时，开始重传!', end='\n')

self.time\_count = 0 # 超时计次重启

for i in range(self.send\_base, self.next\_seq): # 发送空中的所有分组

if random.random() > self.pkt\_loss: # 概率性重传

self.socket.sendto(util.mk\_pkt(i, self.exp\_seq - 1, self.data[i]), self.remote\_address)

isLose = 0

else:

isLose = 1

print('服务器: ' + self.hostname + '\t' + '数据: ' + str(i)

+ ' ack=' + str(self.exp\_seq - 1) + '已重发!', end='\n')

if isLose == 1:

print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(i), end='\n')

# 从文本中读取数据用于模拟上层协议数据的到来

def get\_data\_from\_file(self):

f = open(self.read\_path, 'r', encoding='gb2312')

while True:

send\_data = f.read(self.pkg\_size)

if len(send\_data) <= 0:

break

self.data.append(send\_data) # 将读取到的数据保存到data数据结构中

# 线程执行函数，不断发送数据并接收ACK报文做相应的处理

def server\_run(self):

self.state\_flag = 1

rcv\_finish = 0

send\_finish = 0

while True:

self.send\_data() # 发送数据逻辑

readable = select.select([self.socket], [], [], 1)[0]

if len(readable) > 0: # 接收到数据

rcv\_data = self.socket.recvfrom(self.data\_buf\_size)[0].decode('gb2312')

rcv\_seq = rcv\_data.split()[0] # 按照格式规约获取数据序号

rcv\_ack = rcv\_data.split()[1]

rcv\_data = rcv\_data.replace(rcv\_seq + ' ' + rcv\_ack + ' ', '') # 按照格式规约获取数据

print('服务器: ' + self.hostname + '\t' + '接收到数据: ' + rcv\_seq + ' ack: ' + rcv\_ack, end='\n')

if int(rcv\_seq) == self.exp\_seq: # 接收到按序数据包

print('服务器: ' + self.hostname + '\t' + '接收到期望序号数据: ' + rcv\_seq, end='\n')

self.write\_data\_to\_file(rcv\_data) # 保存服务器端发送的数据到本地文件中

self.exp\_seq = self.exp\_seq + 1 # 期望数据的序号更新

elif rcv\_seq == '0' and rcv\_data == '0': # 接收到结束包

print('服务器: ' + self.hostname + '\t' + '接收数据结束', end='\n')

rcv\_finish = 1

else:

print('服务器: ' + self.hostname + '\t' + '接收到非期望数据，期望: ' +

str(self.exp\_seq) + '实际:' + str(rcv\_seq), end='\n')

if int(rcv\_ack) < self.send\_base:

self.time\_count += 1

else:

self.send\_base = int(rcv\_ack) + 1 # 滑动窗口的起始序号

self.time\_count = 0

print('服务器: ' + self.hostname + '\t' + 'self.send\_base=' + str(self.send\_base), end='\n')

else: # 未收到包

self.time\_count += 1 # 超时计次+1

if self.time\_count > self.time\_out: # 触发超时重传操作

self.handle\_time\_out()

if self.send\_base == len(self.data): # 判断数据是否传输结束

self.socket.sendto(util.mk\_pkt(0, self.exp\_seq - 1, 0), self.remote\_address) # 发送结束报文

print('服务器: ' + self.hostname + '\t' + '发送完毕', end='\n')

send\_finish = 1

if rcv\_finish == 1 and send\_finish == 1:

print('服务器: ' + self.hostname + '\t' + '发送接收均完毕!!', end='\n')

break

self.state\_flag = 0

# 保存来自服务器的合适的数据

def write\_data\_to\_file(self, data, mode='a'):

with open(self.save\_path, mode, encoding='gb2312') as f:

f.write(data) # 模拟将数据交付到上层

def isHostAlive(self):

return self.state\_flag

def shut\_socket(self):

self.socket.close()

**sr.py**

import random

import select

import socket

import gbn

class SR:

def \_\_init\_\_(self,

local\_address: tuple[[str, int]],

remote\_address: tuple[[str, int]],

inputWindow\_size: int = 4,

inputPkg\_size: int = 1024):

self.pkg\_size = inputPkg\_size # 单格窗口大小

self.send\_window\_size = inputWindow\_size # 窗口尺寸

self.send\_base = 0 # 最小的被发送的分组序号

self.next\_seq = 0 # 当前未被利用的序号

self.time\_out = 5 # 设置超时时间

self.local\_address = local\_address # 设置本地socket地址

self.remote\_address = remote\_address # 设置远程socket地址

self.data = [] # 缓存发送数据

self.read\_path = 'file/sr/2read.txt' # 需要发送的源文件数据

self.ack\_buf\_size = 10

self.get\_data\_from\_file()

self.rcv\_window\_size = inputWindow\_size # 接受窗口尺寸

self.data\_buf\_size = 1678 # 作为客户端接收数据缓存

self.save\_path = 'file/sr/2save.txt' # 接收数据时，保存数据的地址

self.write\_data\_to\_file('', mode='w')

self.pkt\_loss = 0.1 # 发送数据丢包率

self.ack\_loss = 0 # 返回的ack丢包率

self.time\_counts = {} # 存储窗口中每个发出序号的时间

self.ack\_seqs = {} # 储存窗口中每个序号的ack情况

self.rcv\_base = 0 # 最小的需要接收的数据的分组序号

self.rcv\_data = {} # 缓存失序的接收数据

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

self.socket.bind(self.local\_address)

self.state\_flag = 0

# 若仍剩余窗口空间，则构造数据报发送；否则拒绝发送数据

def send\_data(self):

if self.next\_seq == len(self.data): # 判断是否还有缓存数据可以发送

print('服务器:发送完毕，等待确认')

return

if self.next\_seq < self.send\_base + self.send\_window\_size: # 窗口中仍有可用空间

if random.random() > self.pkt\_loss:

self.socket.sendto(gbn.util.mk\_pkt1(self.next\_seq, self.data[self.next\_seq]),

self.remote\_address)

self.time\_counts[self.next\_seq] = 0 # 设置计时器

self.ack\_seqs[self.next\_seq] = False # 设置为未接受确认包

print('服务器:发送数据' + str(self.next\_seq))

self.next\_seq += 1

else: # 窗口中无可用空间

print('服务器:窗口已满，暂不发送数据')

# 超时处理函数：计时器置0，设为未接受ACK，同时发送该序列号数据

def handle\_time\_out(self, time\_out\_seq):

print('超时重传:' + str(time\_out\_seq))

self.time\_counts[time\_out\_seq] = 0 # 重新定时

if random.random() > self.pkt\_loss: # 随机发送数据包

self.socket.sendto(gbn.util.mk\_pkt1(time\_out\_seq, self.data[time\_out\_seq]),

self.remote\_address)

# 从文件中读取数据，并存储到data属性里

def get\_data\_from\_file(self):

f = open(self.read\_path, 'r', encoding='gb2312')

while True:

send\_data = f.read(self.pkg\_size) # 一次读取1024个字节（如果有这么多）

if len(send\_data) <= 0:

break

self.data.append(send\_data) # 将读取到的数据保存到data数据结构中

# 滑动窗口，用于接收到最小的ack后调用

def slide\_send\_window(self):

while self.ack\_seqs.get(self.send\_base): # 一直滑动到未接收到ACK的分组序号处

del self.ack\_seqs[self.send\_base] # 从dict数据结构中删除此关键字

del self.time\_counts[self.send\_base] # 从dict数据结构中删除此关键字

self.send\_base = self.send\_base + 1 # 滑动窗口

print('服务器:窗口滑动到' + str(self.send\_base))

def server\_run(self):

self.state\_flag = 1

while True:

self.send\_data() # 发送数据

readable = select.select([self.socket], [], [], 0.1)[0] # 非阻塞方法

if len(readable) > 0: # 接收ACK数据

rcv\_ack = self.socket.recvfrom(self.ack\_buf\_size)[0].decode('gb2312').split()[0]

if self.send\_base <= int(rcv\_ack) < self.next\_seq: # 收到ack，则标记为已确认且超时计数为0

print('服务器:收到有用ACK' + rcv\_ack)

self.ack\_seqs[int(rcv\_ack)] = True # 设为已接受

if self.send\_base == int(rcv\_ack): # 收到的ack为最小的窗口序号

self.slide\_send\_window() # 则滑动窗口

else:

print('服务器:收到无用ACK' + rcv\_ack)

for seq in self.time\_counts.keys(): # 每个未接收的分组的时长都加1

if not self.ack\_seqs[seq]: # 若未收到ACK

self.time\_counts[seq] += 1 # 则计次+1

if self.time\_counts[seq] > self.time\_out: # 触发超时操作

self.handle\_time\_out(seq) # 超时处理

if self.send\_base == len(self.data): # 数据传输结束

self.socket.sendto(gbn.util.mk\_pkt1(0, 0), self.remote\_address) # 发送传输结束包

print('服务器:数据传输结束')

break

self.state\_flag = 0

# 保存来自服务器的合适的数据

def write\_data\_to\_file(self, data, mode='a'):

with open(self.save\_path, mode, encoding='gb2312') as f:

f.write(data)

# 主要执行函数，不断接收服务器发送的数据，若失序则保存到缓存；若按序则滑动窗口；否则丢弃

def client\_run(self):

self.state\_flag = 1

while True:

readable = select.select([self.socket], [], [], 0.1)[0] # 非阻塞接受

if len(readable) > 0:

rcv\_data = self.socket.recvfrom(self.data\_buf\_size)[0].decode('gb2312')

rcv\_seq = rcv\_data.split()[0] # 提取数据包序号

rcv\_data = rcv\_data.replace(rcv\_seq + ' ', '') # 提取数据包数据

if rcv\_seq == '0' and rcv\_data == '0': # 收到传输数据结束的标志

print('客户端:传输数据结束')

break

print('客户端:收到数据' + rcv\_seq)

if self.rcv\_base - self.rcv\_window\_size <= int(

rcv\_seq) < self.rcv\_base + self.rcv\_window\_size:

if self.rcv\_base <= int(rcv\_seq) < self.rcv\_base + self.rcv\_window\_size: # 窗口内

self.rcv\_data[int(rcv\_seq)] = rcv\_data # 失序的数据到来:缓存+发送ack

if int(rcv\_seq) == self.rcv\_base: # 按序数据的到来:滑动窗口并交付数据(清除对应的缓冲区)

self.slide\_rcv\_window()

if random.random() >= self.ack\_loss:

self.socket.sendto(gbn.util.mk\_pkt1(int(rcv\_seq), 0), self.remote\_address)

print('客户端:发送ACK' + rcv\_seq)

self.state\_flag = 0

# 滑动接收窗口:滑动rcv\_base，向上层交付数据，并清除已交付数据的缓存

def slide\_rcv\_window(self):

while self.rcv\_data.get(self.rcv\_base) is not None: # 循环直到出现未接受的数据包

self.write\_data\_to\_file(self.rcv\_data.get(self.rcv\_base)) # 写入文件

del self.rcv\_data[self.rcv\_base] # 清除该缓存

self.rcv\_base = self.rcv\_base + 1 # 滑动窗口

print('客户端:窗口滑动到' + str(self.rcv\_base))

def isHostAlive(self):

return self.state\_flag

def shut\_socket(self):

self.socket.close()

**main.py**

import threading

from time import sleep

import gbn

import sr

def run\_gbn(local: tuple[[str, int]],

remote: tuple[[str, int]],

window\_num: int = 4,

window\_size: int = 1024):

threading.Lock()

host\_1 = gbn.GBN(local, remote, window\_num, window\_size, 'h1')

host\_2 = gbn.GBN(remote, local, window\_num, window\_size, 'h2')

server1 = threading.Thread(target=host\_1.server\_run, args=())

server2 = threading.Thread(target=host\_2.server\_run, args=())

server1.start()

server2.start()

while host\_1.isHostAlive() or host\_2.isHostAlive():

sleep(0.2)

host\_1.shut\_socket()

host\_2.shut\_socket()

def run\_sr(local: tuple[[str, int]],

remote: tuple[[str, int]],

window\_num: int = 4,

window\_size: int = 1024):

host\_1 = sr.SR(local, remote, window\_num, window\_size)

host\_2 = sr.SR(remote, local, window\_num, window\_size)

threading.Thread(target=host\_1.server\_run).start()

threading.Thread(target=host\_2.client\_run).start()

while host\_1.isHostAlive() or host\_2.isHostAlive():

sleep(0.2)

host\_1.shut\_socket()

host\_2.shut\_socket()

#print('gbn:\n')

#run\_gbn(('127.0.0.1', 5000), ('127.0.0.1', 5001), 4, 10)

print('sr:\n')

run\_sr(('127.0.0.1', 5000), ('127.0.0.1', 5001), 4, 10)