期末PJ实验报告

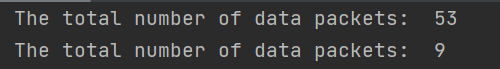
张志豪 20307130085

前言：已经实现了SR协议，因此不再单独阐述如何实现双向传输

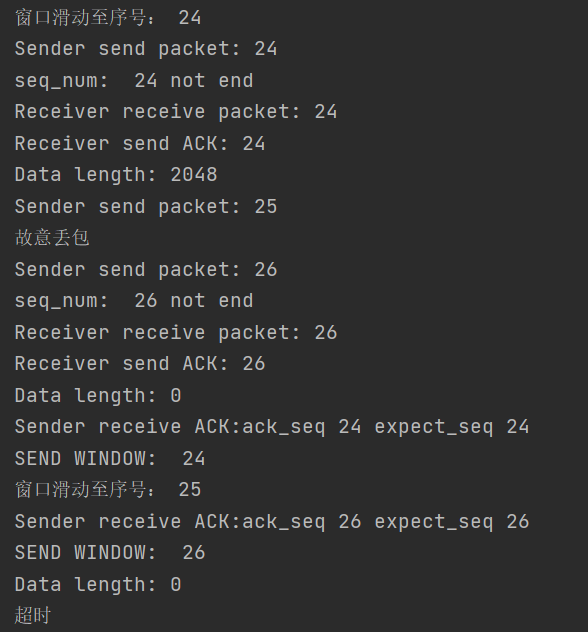
1. SR协议下实验结果

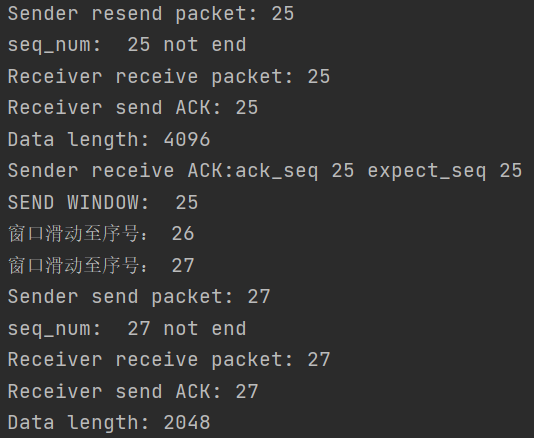
实验全程见附带的录屏

实验中的两个图片一个需要9个分组能完成传输，另一个需要53个分组才能完成传输。



下面找到一处丢包所在位置进行实验结果分析：





根据运行结果可以看出编号为25的分组发生了丢包。本实验中的发送方窗口和接收方窗口大小均设置为3（从0开始），如果按照gbn协议当分组25丢包时应该回退到该窗口的base即分组24，紧接着重发分组24、25、26。

然而在SR协议下，正如上图中所示，25丢包后，发送端接收到24的返回ack，因此窗口滑动到25并在此停顿；发送方继续发送26，收到26的返回ack后发现ack值与窗口值不符，因此得知了分组25发生了超时。接下来重发分组25，发送端接收到分组25返回的ack时窗口滑动到26，又“惊喜地”发现分组26已经被缓存过了，窗口继续滑动至27，继续完成剩余的发送任务。

二、GBN原理

1. 发送端

def GBN\_send(self):  
 self.sender\_socket.settimeout(self.timeout)  
 count = 0  
 while True:  
 if count >= 10:  
 # 连续超时10次，接收方已断开，终止  
 break  
 try:  
 data, address = self.sender\_socket.recvfrom(BUFFER\_SIZE)  
 ack\_seq, expect\_seq = self.analyse\_pkt(data)  
 print('Sender receive ACK:ack\_seq', ack\_seq, "expect\_seq", expect\_seq)  
 print("Send from window: ", ack\_seq)  
 if (self.send\_base == (ack\_seq + 1) % 256):

pass  
 self.send\_base = max(self.send\_base, (ack\_seq + 1) % 256) ##窗口滑动  
 if self.send\_base == self.next\_seq: # 已发送分组确认完毕  
 self.sender\_socket.settimeout(None)  
 return True  
  
 except socket.timeout:  
 # 超时，重发分组. ##回退N步  
 print('超时')  
 for i in range(self.send\_base, self.next\_seq):  
 print('Sender resend packet:', i)  
 self.udp\_send(self.packets[i])  
 self.sender\_socket.settimeout(self.timeout) # reset timer  
 count += 1  
 return False

发送端主要完成三个事件：（1）上层的调用，发送方检查窗口是否已满，如果窗口未满，则产生一个分组并发送；若窗口已满，发送方发送的data为0，隐式指示上层该窗口已满。（2）收到一个ACK后，对ACK编号的确认方式为累积确认（即必须收到n和所有n以前的返回ACK）。（3）对于超时的处理，顾名思义需要“回退n步”，重新发送所有已发送但还未确认的分组。

1. 接收端

def GBN\_receive(self):  
 *"""  
 接收方等待接受数据包  
 """* self.receiver\_socket.settimeout(self.timeout)  
 while True:  
 try:  
 data, address = self.receiver\_socket.recvfrom(BUFFER\_SIZE)  
 self.target = address  
 seq\_num, flag, checksum, data = self.analyse\_pkt(data)  
 print('Receiver receive packet:', seq\_num)  
 # 收到期望数据包且未出错  
 if seq\_num == self.expect\_seq and getChecksum(data) == checksum:  
 self.expect\_seq = (self.expect\_seq + 1) % 256  
 ack\_pkt = self.make\_pkt(seq\_num, seq\_num)  
 self.udp\_send(ack\_pkt)  
 if flag: # 最后一个数据包  
 return data, True # 向上层递交数据块  
 else:  
 return data, False  
 else:  
 ack\_pkt = self.make\_pkt((self.expect\_seq - 1) % 256, self.expect\_seq) # 重复确认，让客户回退N步重传  
 self.udp\_send(ack\_pkt)  
 return bytes('', encoding='utf-8'), False  
 except socket.timeout:  
 return bytes('', encoding='utf-8'), False

如果一个序号为n的分组被正确接收到，并且上次交付给上层的数据序号为n-1，则这个分组传输成功，返回该ACK；其它情况下均丢掉该分组，并且为最近接收到的分组重新发送ACK。

1. SR在GBN基础上的改进

1、发送端

received\_ack[ack\_seq] = 1 # 记录收到的ACK  
 if self.send\_base == (ack\_seq + 1) % 256: # 收到重复确认, 此处应当立即重发  
 pass  
 if self.send\_base == ack\_seq:  
 # 如果发送方的窗口未滑动，且没收到窗口中的某个片段重复确认，SR协议下将窗口滑动到此片段  
 while 1:  
 if received\_ack[self.send\_base] == 1: # 滑动窗口  
 self.send\_base = (self.send\_base + 1) % 256  
 print("窗口滑动至序号：", self.send\_base)  
 else: # 停在未返回ack的片段处  
 break  
 if self.send\_base == self.next\_seq: # 已发送分组确认完毕  
 self.sender\_socket.settimeout(None)  
 return True  
  
 except socket.timeout:  
 # 超时，重发分组.  
 print('超时')  
 for i in range(self.send\_base, self.send\_base + self.window\_size):  
 # 检测重发范围比GBN方法中的要更大，因为窗口base之后windows\_size大小的片段都可能已被传输  
 if already\_sent[i] == 1 and received\_ack[i] == 0:  
 # 如果分组已发送但是没有收到对应编号的ack  
 print('Sender resend packet:', i)  
 self.udp\_send(self.packets[i])  
 self.sender\_socket.settimeout(self.timeout) # reset timer  
 count += 1  
return False

每种情况已在注释中标明，总体来说差异体现在两点：（1）新增两个标志，用来记录序号为n的分组是否已返回ACK和是否已被发送（2）窗口滑动操作，只要收到编号为n的返回ACK，就立即滑动到n+1（3）超时处理，检测重发范围比GBN方法中的要更大，因为窗口base之后windows\_size大小的片段都可能已被传输。

2、接收端

if seq\_num == self.expect\_seq and getChecksum(data) == checksum:  
 self.expect\_seq = (self.expect\_seq + 1) % 256  
 ack\_pkt = self.make\_pkt(seq\_num, seq\_num)  
 self.udp\_send(ack\_pkt)  
 for i in range(self.expect\_seq, self.expect\_seq + self.window\_size):  
 # 只要是发过来的data就都缓存起来  
 if received\_data[i] == 1:  
 data = data + buffer[i]  
 self.expect\_seq = (self.expect\_seq + 1) % 256 # 滑动接收窗口  
 else: # 一旦遇到没发过来的data就停止缓存  
 break  
 if flag: # 最后一个数据包  
 return data, True # 向上层递交数据块  
 else:  
 return data, False  
elif seq\_num < self.expect\_seq and getChecksum(data) == checksum:  
 # 不管该分组是否已被确认，都必须生成一个ack  
 ack\_pkt = self.make\_pkt(seq\_num, seq\_num)  
 self.udp\_send(ack\_pkt)  
 return bytes('', encoding='utf-8'), False  
elif self.expect\_seq < seq\_num < self.expect\_seq + self.window\_size and getChecksum(  
 data) == checksum:  
 # 如果收到的分组在接收方的窗口内，一个选择ack被会送给发送方  
 received\_data[seq\_num] = 1  
 buffer[seq\_num] = data  
 ack\_pkt = self.make\_pkt(seq\_num, seq\_num)  
 self.udp\_send(ack\_pkt)  
 return bytes('', encoding='utf-8'), False  
else: # 忽略改分组  
 return bytes('', encoding='utf-8'), False

SR接收端新增缓存，所有已发送但没有收到ACK的数据都会加入buffer，超时解决后buffer会更新。SR接收方确认一个正确接收的分组不考虑是否按序，失序的分组将被缓存直到所有丢失分组都被收到为止，这时才可以将一批分组按序交付给上层。

四、双向传输的实现

fpSend\_Client = open(os.path.dirname(\_\_file\_\_) + '/client/girl.jpg', 'rb')  
fpSend\_Server = open(os.path.dirname(\_\_file\_\_) + '/server/cartoon.jpg', 'rb')  
fpReceive\_Client = open(os.path.dirname(\_\_file\_\_) + '/client/' + str(int(time.time())) + '.jpg', 'ab')  
fpReceive\_Server = open(os.path.dirname(\_\_file\_\_) + '/server/' + str(int(time.time())) + '.jpg', 'ab')  
  
Socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
Socket2 = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
sender\_client = GBNSender(Socket, ('127.0.0.1', 8888))  
sender\_server = GBNSender(Socket2, ('127.0.0.1', 7777))  
  
receiverSocket\_Client = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
receiverSocket\_Server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
receiverSocket\_Client.bind(('', 7777))  
receiverSocket\_Server.bind(('', 8888))  
  
Client\_Send = threading.Thread(target=Send\_SR, args=(sender\_client, fpSend\_Client,))  
Server\_Send = threading.Thread(target=Send\_SR, args=(sender\_server, fpSend\_Server,))  
Client\_Receive = threading.Thread(target=Receive\_SR, args=(GBNReceiver(receiverSocket\_Client), fpReceive\_Client,))  
Server\_Receive = threading.Thread(target=Receive\_SR, args=(GBNReceiver(receiverSocket\_Server), fpReceive\_Server,))  
Client\_Send.start()  
Server\_Send.start()  
Client\_Receive.start()  
Server\_Receive.start()

使用线程函数完成双向的同时传输。由于需要同时进行，因此需要开两个套接字，两个传输方向各自的client端和server端使用相同的端口号。

运行结果也证明是双向同时传输的：

