2022-2023年春季学期计算机网络与安全

Project实验报告

程礼彬19300740005环境科学（环境管理方向）

1. 实验目的

进一步深入理解传输层协议：滑动窗口协议的基本原理；掌握GBN/SR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，并在此基础上改进实现选择重传（SR）协议。

1. 实验要求

1)基于UDP设计一个简单的GBN协议，实现双向可靠数据传输。

2)在此基础上改进并实现SR协议。

3)模拟引入数据包的丢失，验证所设计协议的有效性。

1. 实验原理

单向数据传输的GBN协议，实质上就是实现为一个C/S应用。

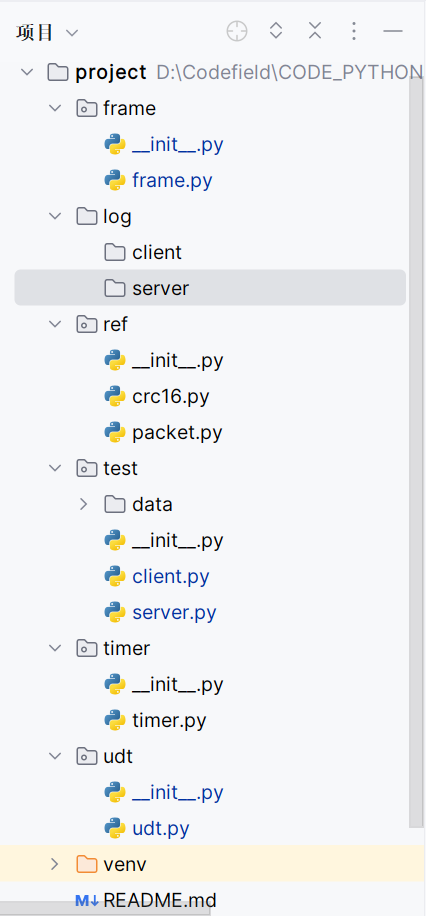
服务器端：使用UDP协议传输数据（比如传输一个文件），等待客户端的请求，接收并处理来自客户端的消息（如数据传输请求），当客户端开始请求数据时进入"伪连接"状态（并不是真正的连接，只是一种类似连接的数据发送的状态），将数据打包成数据报发送，然后等待客户端的ACK信息，同时启动计时器。当收到ACK时，窗口滑动，正常发送下一个数据报，计时器重新计时；若在计时器超时前没有收到ACK，则全部重传窗口内的所以已发送的数据报。

客户端：使用UDP协议向服务器端请求数据，接收服务器端发送的数据报并返回确认信息ACK（注意GBN为累积确认，即若ACK=1和3，表示数据帧2已经正确接收），必须能够模拟ACK丢失直至服务器端超时重传的情况。

单向数据传输的SR协议，实质上也是实现为一个C/S应用，其与GBN协议不同的是在服务器端，当超时发生时不重传窗口内所有已发送的数据报，而是只发送窗口内那些没有收到ACK的数据报；在客户端，接受服务端发来的数据报返回确认信息ACK，只不过是收到哪个序号的数据包就返回哪个序号的ACK，且此时在客户端要有一个接受窗口，当在窗口内数据包接受到，即存下来，然后按照依次的顺序交付给上层协议。

1. 具体实现

代码结构如图所示，log文件夹打印日志，ref文件夹中是一些辅助函数，test文件夹用于测试，传输的图片位于data文件夹中，timer、udt、frame是三个类。



1. FRAME类

FRAME类定义帧格式。

class FRAME:  
 def \_\_init\_\_(self, seq\_num, data=b''):  
 self.seq = seq\_num  
 self.data = data  
 self.crc = crc16.crc16xmodem(data)  
 self.start\_time = -1  
  
 def \_\_str\_\_(self):  
 return "frame"

1. timer类

timer类实现计时器和超时重传。

class timer:  
 TIMER\_STOP = -1\_TIMER = {}  
   
 def \_\_init\_\_(self, \_interval):  
 self.start\_time = self.TIMER\_STOP  
 self.interval = \_interval  
 def satrt(self, seq):  
 self.\_TIMER[seq] = time.time()  
  
 def get\_time(self):  
 return time.time()  
  
 def overtime(self,seq):  
 if seq >= len(self.\_TIMER):  
 seq -= 1  
 if time.time() - self.\_TIMER[seq] > self.interval:  
 return True  
 else:  
 return False

1. UDT类

UDT类用于定义发送、接收帧的行为。

class UDT:  
 def \_\_init\_\_(self, lost, err):  
 random.seed(time.time())

self.LOST\_PROB = lost

self.ERR\_PROB = errdef send(self, packet, sock, addr):  
 if random.random() < self.ERR\_PROB:  
 packet = self.make\_error(packet)  
 if random.random() > self.LOST\_PROB:  
 sock.sendto(packet, addr)  
 return  
def recv(self, sock):  
 packet, addr = sock.recvfrom(1024)  
 return packet, addr  
  
 def sendack(self, ack, sock, addr):  
 ack\_bytes = ack.to\_bytes(4, byteorder = 'little', signed = True)  
 if random.random() > self.LOST\_PROB:  
 sock.sendto(ack\_bytes, addr)  
 return  
  
 def recvack(self, sock):  
 ack\_bytes, addr = sock.recvfrom(1024)  
 ack = int.from\_bytes(ack\_bytes, byteorder = 'little', signed = True)  
 return ack, addr  
  
 def make\_error(self, packet):  
 ErrData = b''  
 for i in range(len(packet) - 8):  
 byte = random.randint(65, 121)  
 ErrData = ErrData + byte.to\_bytes(1, byteorder = 'little', signed = True)  
 return packet[0:8] + ErrData

（4）reference

1、crc16.py

crc16.py是引入的库，用于实现计算crc和校验功能。

2、packet.py

packet.py实现组帧和提取帧中数据的功能。

（5）client.py

调用client.py开启进程1。

import socket  
from ref import crc16, packet  
from udt import udt  
import os  
import time  
import threading  
  
  
def receive(sock, filename, IP\_PORT):  
 UDTER = udt.UDT(0.005, 0.005)  
 file = open(filename, "wb")  
 log\_filename = IP\_PORT[0] + "\_" + str(IP\_PORT[1]) + "\_" + "log.txt"  
 log\_file = open("../log/client/" + log\_filename, "a+")  
 log\_file.write("-------------------------------\n")  
 frame\_expected = 0  
 log\_file.write("Receiving %s...\n" % filename)  
  
 while True:  
 pdu, addr = UDTER.recv(sock)  
 if not pdu:  
 break  
 seq\_num, crc\_num, data = packet.extract(pdu)  
   
 print('Got frame', seq\_num)  
  
 crc\_expected = crc16.crc16xmodem(data)  
 if crc\_expected != crc\_num:  
 log\_file.write("%s: Receive frame = %d, STATUS = DataErr, FRAME\_EXPECTED = %d from %s\n" \  
 %(time.ctime(), seq\_num, frame\_expected, str(addr)))  
 print("data with error")  
 continue  
  
 if seq\_num == frame\_expected:  
 print('Got expected packet')  
 log\_file.write("%s: Receive frame = %d,STATUS = OK, FRAME\_EXPECTED = %d from %s\n" \  
 %(time.ctime(), seq\_num, frame\_expected, str(addr)))  
 print('Sending ACK', frame\_expected)  
 UDTER.sendack(frame\_expected, sock, addr)  
 frame\_expected += 1  
 file.write(data)  
   
 else:  
 print('Got unexpected packet')  
 log\_file.write("%s: Receive frame = %d, STATUS = NoErr, FRAME\_EXPECTED = %d from %s\n" \  
 %(time.ctime(), seq\_num, frame\_expected, str(addr)))  
 print('Sending ACK', frame\_expected - 1)  
 UDTER.sendack(frame\_expected - 1, sock, addr)  
  
 print("over")  
 log\_file.write("Receive succeed\n")  
 log\_file.write("-------------------------------\n\n\n")  
 log\_file.close()  
 file.close()  
  
  
def main():  
 IP = ""  
 PORT = 808  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 IP\_PORT = (IP, PORT)  
 sock.bind(IP\_PORT)  
  
 LIENT\_DIR = os.path.dirname(\_\_file\_\_) + '/data/client'  
 filename = LIENT\_DIR + "/copy.jpg"  
  
 lock = threading.Lock()  
 lock.acquire()  
 lock.release()  
 receive\_thread = threading.Thread(target = receive, args = (sock, filename, IP\_PORT))  
 receive\_thread.start()  
 receive\_thread.join()  
  
  
if \_\_name\_\_=='\_\_main\_\_':  
 main()

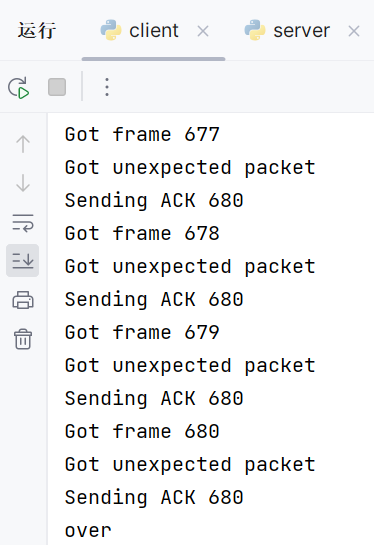
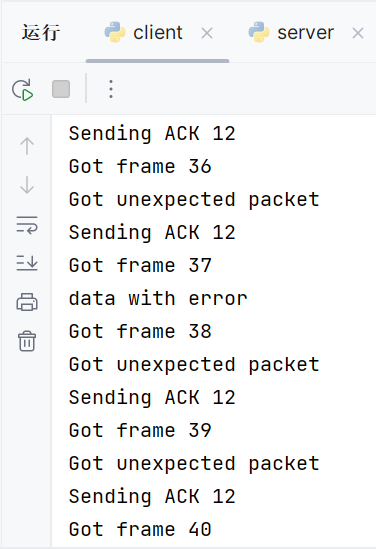
（6）server.py

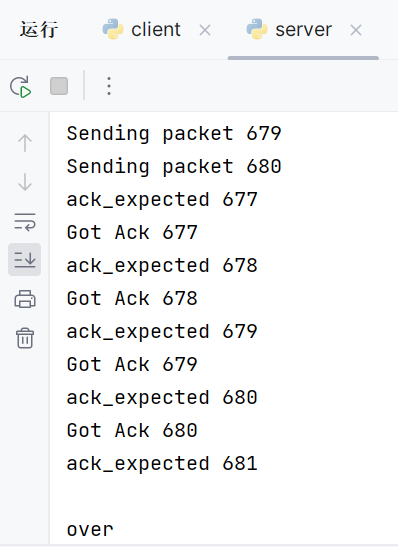
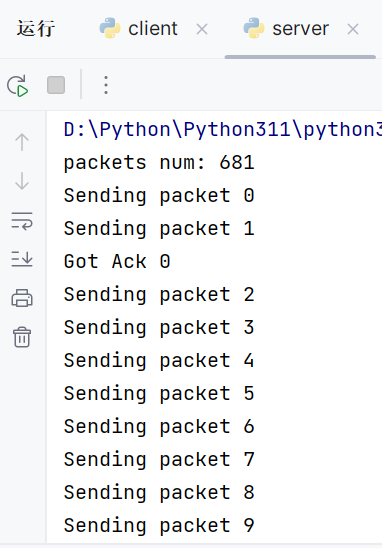
调用server.py调用进程2，发送test/data/server文件夹中的图片data.jpg。

import socket  
from udt import udt  
import \_thread  
from timer import timer  
import os  
from ref import crc16, packet  
import time  
import threading  
  
interval = 1  
expected\_ack = 0 *#累计确认，只用维护一个*packets\_num = 0  
send\_timer = timer.timer(interval)  
log\_filename = ""  
mutex = \_thread.allocate\_lock()  
UDTER = udt.UDT(0.005, 0.005)  
  
  
def send(sock, filename, IP\_PORT, RECEIVER\_ADDR):  
 global UDTER  
 global mutex  
 global expected\_ack  
 global packets\_num  
 global send\_timer  
 global log\_filename  
  
 *# log printing* log\_filename = IP\_PORT[0] + "\_" + str(IP\_PORT[1]) + "\_" + "log.txt"  
 log\_file = open("../log/server/" + log\_filename, "a+")  
 file = open(filename,"rb")  
 log\_file.write("-------------------------------\n")  
 log\_file.write("%s send %s to %s\n" % (IP\_PORT[0] + " " + str(IP\_PORT[1]), filename, RECEIVER\_ADDR[0] + " " + str(RECEIVER\_ADDR[1])))  
  
 packets = []  
 seq\_num = 0  
 while True:  
 data = file.read(512) *#data size* if not data:  
 break  
 crc\_num = crc16.crc16xmodem(data) *# calculate crc* pdu = packet.make(seq\_num, crc\_num, data) *# make packet* packets.append(pdu)  
 seq\_num += 1  
 packets\_num = len(packets)  
 log\_file.write("total %d packets(512bytes)\n" % packets\_num)  
 print('packets num:', packets\_num)  
 window\_size = 200  
 next\_frame\_to\_send = 0  
  
 *#start receive ack thread* THREAD = threading.Thread(target = receive,args = (sock, ))  
 THREAD.start()  
 overtime\_flag = 0  
 scale = 50  
 start = time.perf\_counter()  
 pre = start  
 while expected\_ack < len(packets):  
 mutex.acquire()  
 while next\_frame\_to\_send < expected\_ack + window\_size:  
 if next\_frame\_to\_send >= len(packets):  
 break  
 print('Sending packet', next\_frame\_to\_send)  
 if overtime\_flag == 0:  
 log\_file.write("%s: Send frame = %d, STATUS = New, ACKed = %d to %s\n" % (time.ctime(), next\_frame\_to\_send, expected\_ack, str(RECEIVER\_ADDR)))  
 elif overtime\_flag == 1:  
 log\_file.write("%s: Send frame = %d, STATUS = TO, ACKed = %d to %s\n" % (time.ctime(), next\_frame\_to\_send, expected\_ack, str(RECEIVER\_ADDR)))  
 send\_timer.satrt(next\_frame\_to\_send)  
 UDTER.send(packets[next\_frame\_to\_send], sock, RECEIVER\_ADDR)  
 next\_frame\_to\_send += 1  
 overtime\_flag = 0  
 if send\_timer.overtime(expected\_ack):  
 print("overtime")  
 overtime\_flag = 1  
 next\_frame\_to\_send = expected\_ack  
  
 *# print result* if (time.perf\_counter() - pre) > 1:  
 pre = time.perf\_counter()  
 param = int(packets\_num / 50)  
 i = int(next\_frame\_to\_send / param)  
 a = '\*' \* i  
 b = '.' \* (scale - i)  
 c = min((i / scale) \* 100, 100)  
 dur = pre - start  
 print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c, a, b, dur), end='')  
 mutex.release()  
 print("\nover")  
 UDTER.send(packet.make\_empty(), sock, RECEIVER\_ADDR)  
 log\_file.write("send succeed\n")   
 log\_file.write("-------------------------------\n\n\n")  
 file.close()  
 log\_file.close()  
  
def receive(sock):  
 global mutex  
 global expected\_ack  
 global packets\_num  
   
 while True:  
 ack, \_ = UDTER.recvack(sock)  
 print('Got Ack', ack)  
 if ack >= expected\_ack:  
 mutex.acquire()  
 expected\_ack = ack + 1  
 print('ack\_expected', expected\_ack)  
 mutex.release()  
 if expected\_ack >= packets\_num:  
 break  
  
  
def main():  
 hostname = socket.gethostname()  
 IP = socket.gethostbyname(hostname)  
 PORT = 809  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 IP\_PORT = (IP, PORT)  
 sock.bind(IP\_PORT)  
  
 lock = threading.Lock()  
 lock.acquire()  
 LIENT\_DIR = os.path.dirname(\_\_file\_\_) + '/data/server'  
 filename = LIENT\_DIR + "/data.jpg"  
 RECEIVER\_IP = socket.gethostbyname(hostname)  
 RECEIVER\_PORT = 808  
 RECEIVER\_IP\_PORT = (RECEIVER\_IP, RECEIVER\_PORT)  
 lock.release()  
  
 send\_thread = threading.Thread(target=send, args=(sock, filename, IP\_PORT, RECEIVER\_IP\_PORT))  
 send\_thread.start()  
 send\_thread.join()  
  
  
if \_\_name\_\_=='\_\_main\_\_':  
 main()

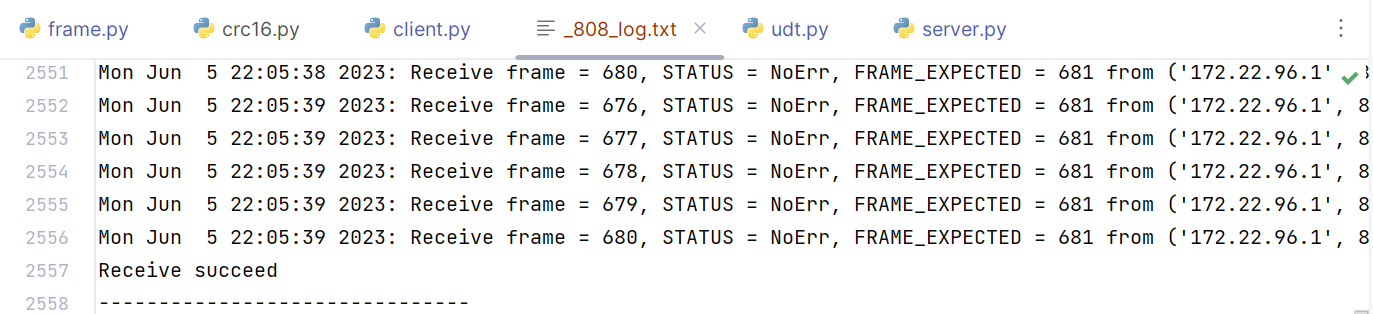
1. 实验结果

这里将丢失概率设为0.5%，可以得到以下实验结果：

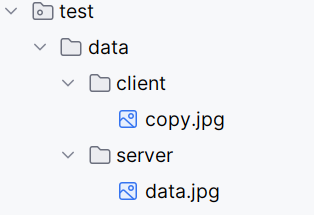




日志打印结果：



文件传输结果：



1. 参考资料
2. <https://zzbloc.top/archives/computer-networking-lab2-gbn>
3. 《计算机网络——自顶向下方法》（第七版）
4. <https://blog.csdn.net/weixin_55697913/article/details/130454752>
5. <https://blog.csdn.net/weixin_43877853/article/details/123789260>
6. <https://github.com/bicongwang/hit-computer_network#%E5%8F%AF%E9%9D%A0%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE%E7%9A%84%E5%AE%9E%E7%8E%B0>