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）

一、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。

二、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。

三、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。

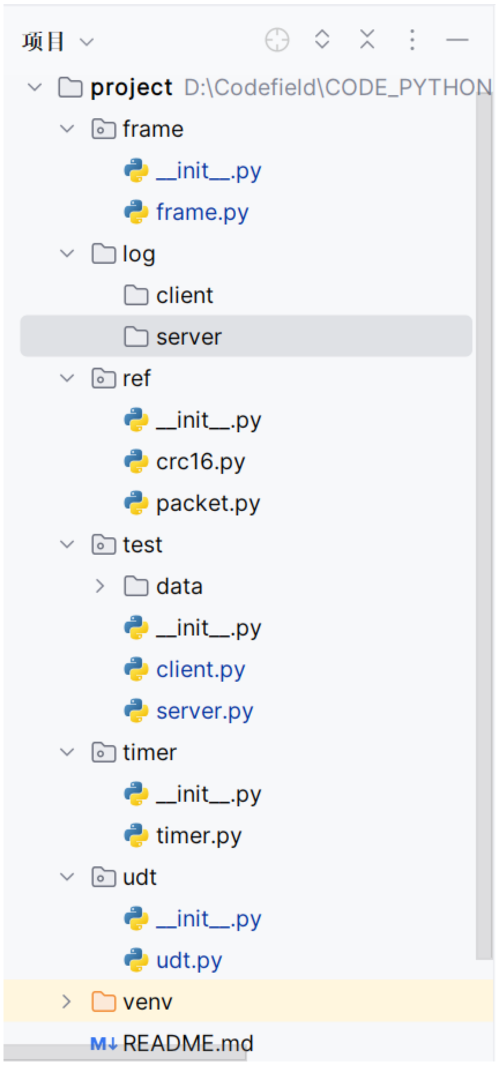
服务器端：使用 UDP 协议传输数据（比如传输一个文件），等待客户端的请求，接收并处理来自客户端的消息（如数据传输请求），当客户端开始请求数据时进入"伪连接"状态（并不是真正的连接，只是一种类似连接的数据发送的状态），将数据打包成数据报发送，然后等待客户端的 ACK 信息，同时启动计时器。当收到 ACK 时，窗口滑动，正常发送下一个数据报，计时器重新计时；若在计时器超时前没有收到 ACK，则全部重传窗口内的所以已发送的数据报。

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。

四、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。


```
项目
project D:\Codefield\CODE_PYTHON
    frame
        __init__.py
        frame.py
    log
        client
        server
    ref
        __init__.py
        crc16.py
        packet.py
    test
        data
        __init__.py
        client.py
        server.py
    timer
        __init__.py
        timer.py
    udt
        __init__.py
        udt.py
    venv
    README.md
```

（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"
```

（2） 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
```

（3） 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 = err

    def 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

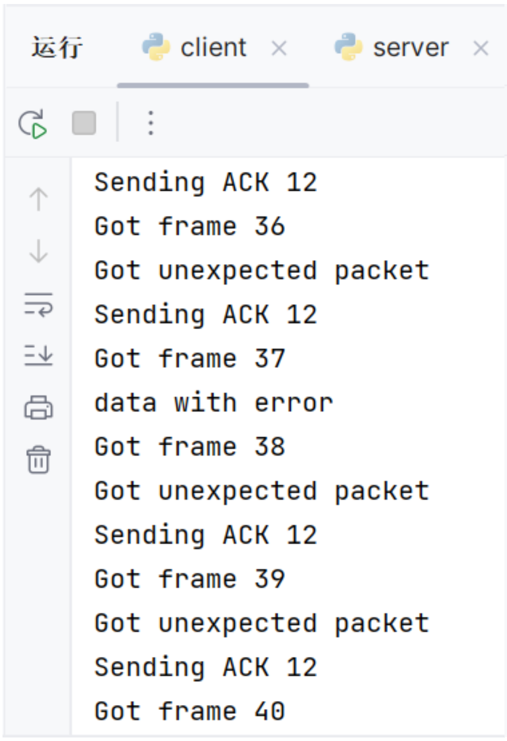
调用 server.py 调用进程 2，发送 test/data/server 文件夹中的图片 data.jpg。

```
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)
```

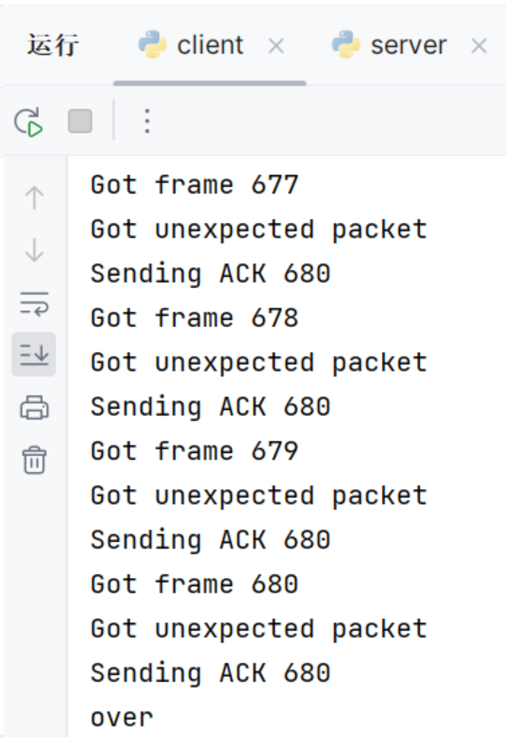
```
            print('Got Ack', ack)
            if ack >= expected_ack:
                mutex.acquire()
                expected_ack = ack + 1
                print('ack_expected', expected_ack)
                mutex.release()
            if expected_ack >= packets_num:
                break


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()
```

五、实验结果

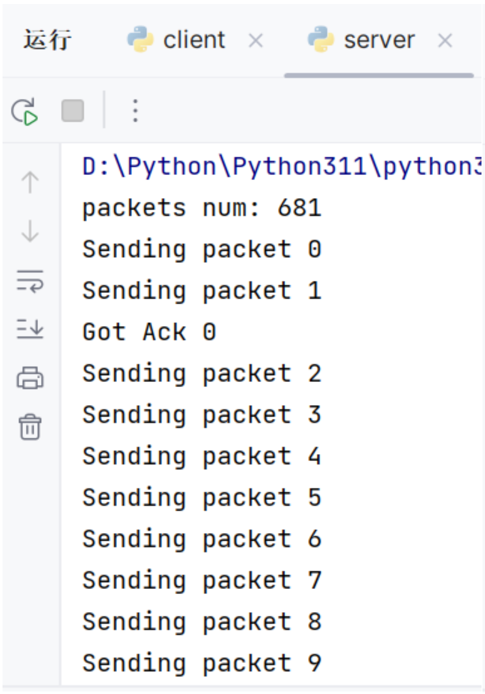
这里将丢失概率设为 0.5%，可以得到以下实验结果：



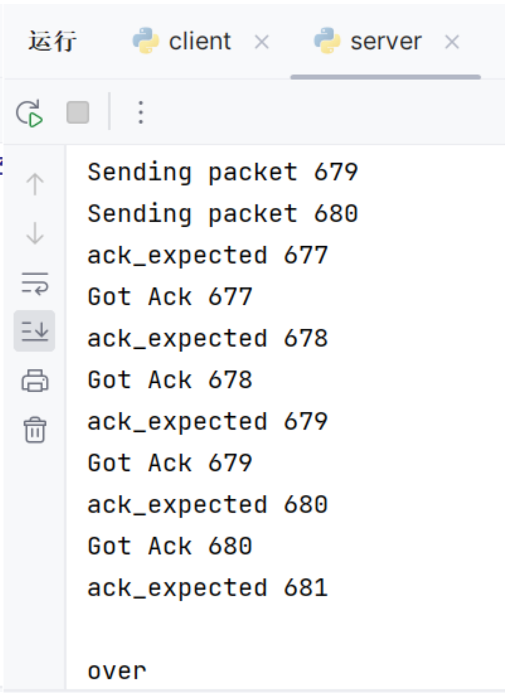
```
Sending ACK 12
Got frame 36
Got unexpected packet
Sending ACK 12
Got frame 37
data with error
Got frame 38
Got unexpected packet
Sending ACK 12
Got frame 39
Got unexpected packet
Sending ACK 12
Got frame 40
```



```
Got frame 677
Got unexpected packet
Sending ACK 680
Got frame 678
Got unexpected packet
Sending ACK 680
Got frame 679
Got unexpected packet
Sending ACK 680
Got frame 680
Got unexpected packet
Sending ACK 680
over
```

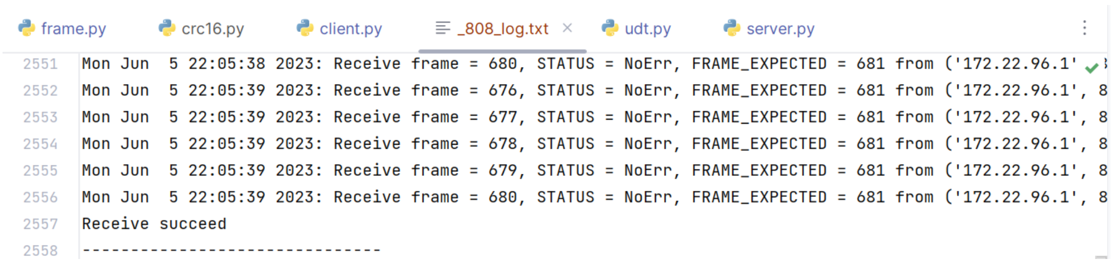


```
D:\Python\Python311\python3
packets num: 681
Sending packet 0
Sending packet 1
Got Ack 0
Sending packet 2
Sending packet 3
Sending packet 4
Sending packet 5
Sending packet 6
Sending packet 7
Sending packet 8
Sending packet 9
```



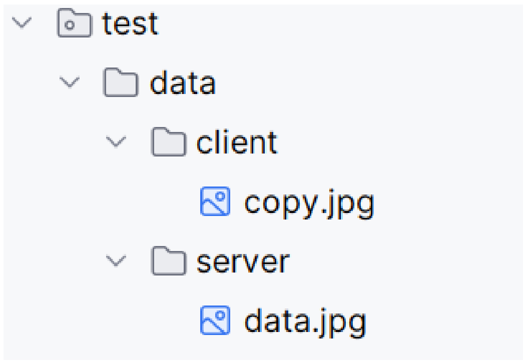
```
Sending packet 679
Sending packet 680
ack_expected 677
Got Ack 677
ack_expected 678
Got Ack 678
ack_expected 679
Got Ack 679
ack_expected 680
Got Ack 680
ack_expected 681

over
```

日志打印结果：



```
Mon Jun  5 22:05:38 2023: Receive frame = 680, STATUS = NoErr, FRAME_EXPECTED = 681 from ('172.22.96.1'
Mon Jun  5 22:05:39 2023: Receive frame = 676, STATUS = NoErr, FRAME_EXPECTED = 681 from ('172.22.96.1', 8
Mon Jun  5 22:05:39 2023: Receive frame = 677, STATUS = NoErr, FRAME_EXPECTED = 681 from ('172.22.96.1', 8
Mon Jun  5 22:05:39 2023: Receive frame = 678, STATUS = NoErr, FRAME_EXPECTED = 681 from ('172.22.96.1', 8
Mon Jun  5 22:05:39 2023: Receive frame = 679, STATUS = NoErr, FRAME_EXPECTED = 681 from ('172.22.96.1', 8
Mon Jun  5 22:05:39 2023: Receive frame = 680, STATUS = NoErr, FRAME_EXPECTED = 681 from ('172.22.96.1', 8
Receive succeed
------------------------------
```

文件传输结果：


```
test
  data
    client
      copy.jpg
    server
      data.jpg
```

六、参考资料

1、https://zzbloc.top/archives/computer-networking-lab2-gbn

2、《计算机网络——自顶向下方法》（第七版）

3、https://blog.csdn.net/weixin_55697913/article/details/130454752

4、https://blog.csdn.net/weixin_43877853/article/details/123789260

5、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