哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 计算机网络

课程类型： 必修

实验题目： 可靠数据传输协议-GBN协议的设计与实现

学号：1171000405

姓名：许天骁

一、实验目的

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。

二、实验要求及实验环境

实验内容

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. 利用设计的GBN协议实现文件传输。

实验环境

1. 硬件环境：X64CPU; 4.0Ghz; 16G RAM
2. 软件环境：Windows 64位
3. 开发环境：Python 3.6.8(64位); Visual Stuio Code

三、实验过程及结果

UDP协议

UDP是一个无连接协议，传输数据之前源端和终端不建立连接。在收到发送消息的指令时，直接向目标IP地址发送消息。不像TCP协议一样建立连接。接收消息时，接收端接收指定IP地址发来的消息。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。在本实验中，由于使用的时python。我们将只能使用recvfrom，sendto函数来进行数据收发。根据UDP协议，sendto自动在数据段之前添加了checksum字段，recvfrom会对接收到的数据进行checksum的检查。当检测到错误时，UDP不对错误的数据进行校正。而是简单的扔掉这个数据报。同时sendto函数也不能保证数据准确送达。所以需要滑动窗口协议来完成可靠的数据传输。

滑动窗口协议

在任意时刻，发送方都维持了一个连续的允许发送的帧的序号，称为发送窗口；同时，接收方也维持了一个连续的允许接收的帧的序号，称为接收窗口。发送窗口和接收窗口的序号上下界和大小都不要求相同。不同的滑动窗口协议窗口大小一般不同。发送方窗口内的序列号代表了那些已经被发送，但是还没有被确认的帧，或者是那些可以被发送的帧。接收窗口内的序号表示，准备接收确认的帧。
从滑动窗口的大小和计时的方法来看，可以划分三种滑动窗口协议。分别是停等协议（发送窗口 = 1，接收窗口 = 1）。GBN协议（发送窗口 > 1，接收窗口 = 1）。SR协议（发送窗口 > 1， 接收窗口 > 1）

GBN协议实现

1. 实现单向发送

首先建立起socket连接，启动两次程序后，两个程序分别会绑定一个特定的套接字获得IP地址和端口号。

发送数据：

当程序接收到测试GBN的指令后，自动生成一串字符串。建立新的发送线程。进入线程后，自动根据参数将数据进行分块和编号。接着建立发送窗口。当窗口大小允许时，便向固定的IP地址和端口号发送数据。根据GBN协议的规定，进行超时重发。

数据的包装如下：

1	2 ⋯ CHUNK_SIZE
编号	数据

第一个字节储存着编号，接着字节储存着数据，每一个数据报的长度由CHUNK_SIZE进行限制。

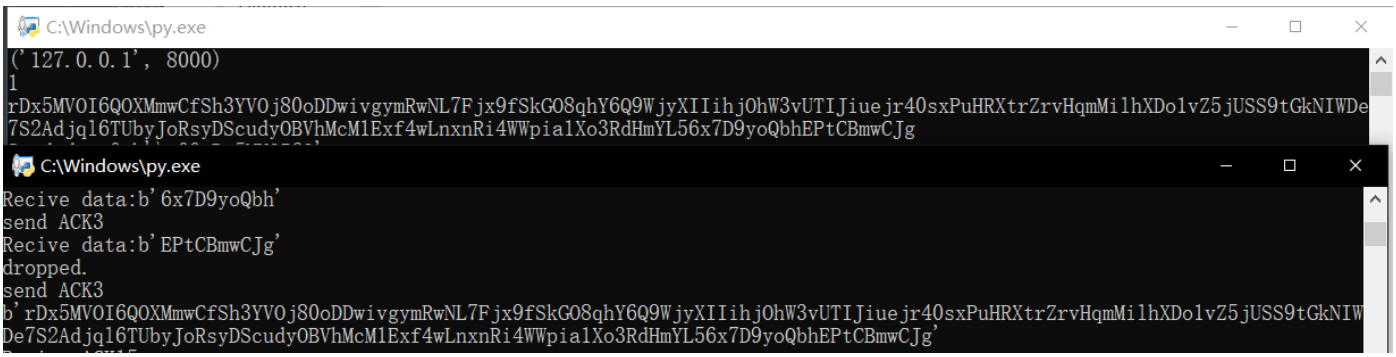
接收数据：

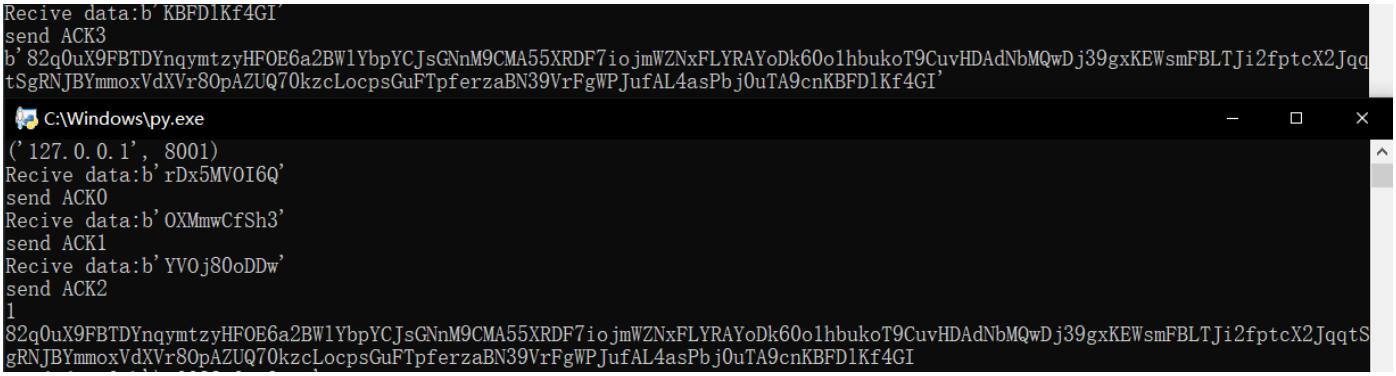
根据启动线程时的设置，在固定的端口号进行监听。收到数据时将数据进行解包，拆分成编号和数据部分。根据GBN协议，若编号与期望接收到的编号相同，则将期望接受编号加1，并且发送一个仅包含一个字节数字的ACK确认信息。表示已经完成接收的最大的数据报的编号。并且使窗口向右滑动。

上图是单向发送时最终的结果。下方的程序作为发送方，上方的程序作为接收方，完成了数据的单向的发送。

2. 实现双向发送

因为要实现全双工的发送，所以不再区分客户端和服务器。发送部分保持不变，需要对接收部分进行修改。若收到的数据只有一个字节则表示接收到的是ACK确认信息。此时接收部分进行服务器的操作。若接收到的数据报包含数据部分。则表示收到的是数据报，此时接收部分进行客户端的操作。

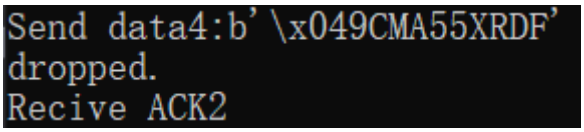




上两图是执行双向发送时的输出情况。端口号为8000的程序首先执行了发送指令。可以对比第二个程序的输出结果是相同的。在上述的发送过程中，端口号为8001的程序也执行了发送指令。可以对比和上方的程序输出结果是相同的。

3. 引入丢包和超时

   丢包：在此实验中，所有的数据发送均启动了新的线程进行发送。若需要丢包则在发送之前，进行随机操作。保证以一定的概率不执行发送代码，完成丢包操作。

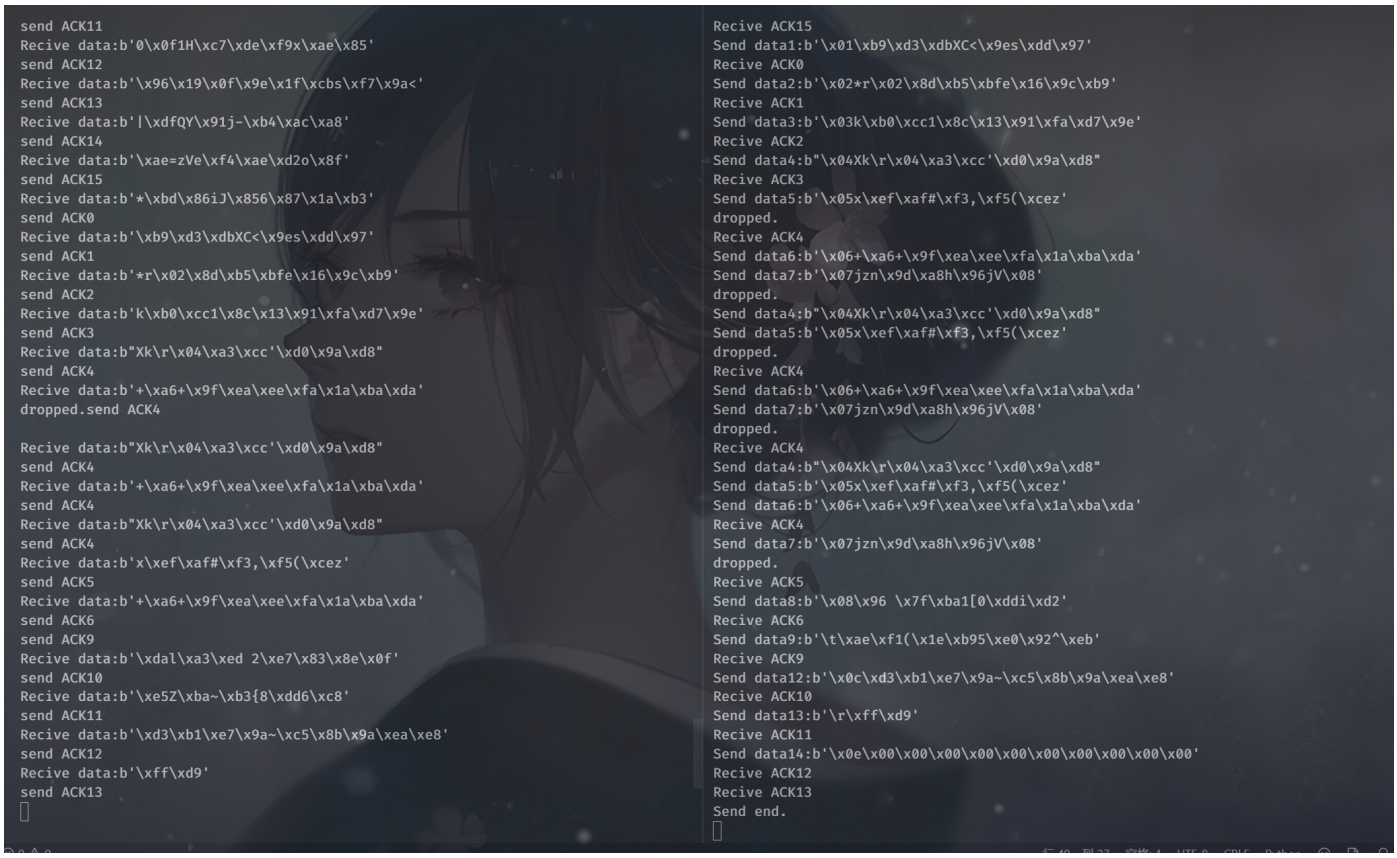


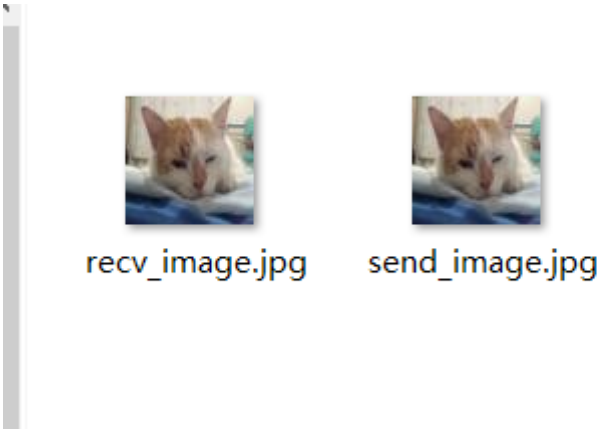
   发送任何一个数据报都有可能发送丢包，如果程序输出了dropped表示模拟出了一次丢包。

   超时：同样是在发送之前，进行随即操作。保证以一定的概率等待一段时间后再执行发送代码。完成超时操作。

4. 传输文件

   传输文件与发送数据类似。将需要传送的文件放入指定文件夹中。此实验中利用图片进行测试。利用二进制形式打开文件，得到字段然后进行发送。传输完成前的最后一部分过程和传输结果如下图所示。





四、实验心得

1. 本次实验深入了解了滑动窗口协议，和UDP数据报的发送原理。更加深入了解了数据的包装和发送的整体过程。
2. 熟练掌握了python多线程的使用和多线程数据共享的操作。

五、源代码

```python
import socket
import random
import string
import time
from threading import Lock, Thread
import traceback

S1_ADDR = ('127.0.0.1', 8000)
S2_ADDR = ('127.0.0.1', 8001)

DATA_LENGTH = 200
CHUNK_SIZE = 50
WINDOW = 5
RECV_SIZE = 1024
TIMEOUT = 0.5
MAX_SIZE = 16
TIME_OUT_RATE = 0.1
PACKET_LOSS_RATE = 0.1
EOF = bytes([0]) * CHUNK_SIZE
base = 0
timer = 0
lock = Lock()
def make_data(length):
    return ''.join(random.choices(string.ascii_letters + string.digits, k=length)).encode('utf-8

def make_file():
    with open('计算机网络\GBN\image\send_image.jpg', 'rb') as f:
        image_data = f.read()
    return image_data

def make_ack(num):
    return (num).to_bytes(1, byteorder='little')

def divide_data(base_data, chunk_size):
    length = len(base_data)
    re = []
    i = 0
    while i * chunk_size < length:
        if i + chunk_size < length:
            re.append((i % MAX_SIZE).to_bytes(1, byteorder='little') + base_data[i * chunk_size:
            i += 1
        else:
            re.append((i % MAX_SIZE).to_bytes(1, byteorder='little') + base_data[i:])
            break
    eof = (i % MAX_SIZE).to_bytes(1, byteorder='little') + bytes([0]) * chunk_size
    re.append(eof)
    return re

def get_data(sdata):
    num = int.from_bytes(sdata[:1], byteorder='little')
    if len(sdata) == 1:
```

```
        return (num, None)
    else:
        data = sdata[1:]
    return (num, data)

def send(conn, data, tar):
    if random.random() >= PACKET_LOSS_RATE:
        conn.sendto(data, tar)
    else:
        print('dropped.')
    return

def sendto(conn, rawdata, tar):
    data_piece = divide_data(rawdata, CHUNK_SIZE)
    global base
    global timer
    next_seq_num = 0
    tot_num = len(data_piece) - 1
    win = WINDOW
    next_num = 0
    while True:
        lock.acquire()
        if (next_seq_num - base + 1 + MAX_SIZE) % MAX_SIZE < win and next_num <= tot_num:
            send_thread = Thread(target=send, args=(conn, data_piece[next_num], tar))
            if(random.random() <= TIME_OUT_RATE):
                time.sleep(1.5)
            print('Send data%s:%s'%(next_seq_num, data_piece[next_num]))
            send_thread.start()
            if base == next_seq_num:
                timer = time.perf_counter()
            next_seq_num = (next_seq_num + 1) % MAX_SIZE
            next_num += 1
        if time.perf_counter() - timer > TIMEOUT:
            timer = time.perf_counter()
            gap = (next_seq_num - base + MAX_SIZE) % MAX_SIZE
            base_num = next_num - gap
            for i in range(gap):
                send_thread = Thread(target=send, args=(conn, data_piece[base_num + i], tar))
                if(random.random() <= TIME_OUT_RATE):
                    time.sleep(1.5)
                print('Send data%s:%s'%((base_num + i) % MAX_SIZE, data_piece[base_num + i]))
                send_thread.start()
        if next_num == tot_num + 1 and (next_seq_num - base + MAX_SIZE) % MAX_SIZE == 2:
            lock.release()
            break
        lock.release()
        time.sleep(0.05)
    print('Send end.')

def recvfrom(conn, size, tar):
    global base
```

```
    global timer
    expect_seq_num = 0
    re = b''
    while True:
        rawdata, addr = conn.recvfrom(size)
        num, data = get_data(rawdata)
        if data == EOF:
            with open('计算机网络\GBN\image\\recv_image.png', 'wb') as f:
                f.write(re)
        elif data == None:
            print('Recive ACK%d'%(num))
            ack = num
            lock.acquire()
            if (ack - base + 1 + MAX_SIZE) % MAX_SIZE <= WINDOW:
                base = ack
                timer = time.perf_counter()
            lock.release()
        else:
            print('Recive data:%s'%(data))
            if num == expect_seq_num:
                expect_seq_num = (expect_seq_num + 1) % MAX_SIZE
                re += data
            ack = make_ack((expect_seq_num + MAX_SIZE - 1) % MAX_SIZE)
            send_thread = Thread(target=send, args=(conn, ack, tar))
            send_thread.start()
            print('send ACK%d'%(int.from_bytes(ack, byteorder='little')))
        time.sleep(0.05)


def listen(conn, tar_addr):
    recv_th = Thread(target=recvfrom, args=(conn, RECV_SIZE, tar_addr))
    recv_th.start()
    while True:
        arg = input()
        if arg == 'time':
            now_time = time.time()
            time_local = time.localtime(now_time)
            data = time.strftime("%Y-%m-%d %H:%M:%S", time_local)
            print(data)
            data = data.encode('utf-8')
            send_th = Thread(target=sendto, args=(conn, data, tar_addr))
            send_th.start()
        if arg == 'quit':
            break
        if arg == 'testgbn':
            data = make_data(DATA_LENGTH)
            print(data.decode('utf-8'))
            send_th = Thread(target=sendto, args=(conn, data, tar_addr))
            send_th.start()
        if arg == '1':
            data = make_file()
```

```
            send_th = Thread(target=sendto, args=(conn, data, tar_addr))
            send_th.start()
        time.sleep(0.5)

def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.bind(S1_ADDR)
        print(S1_ADDR)
        listen(s, S2_ADDR)
    except Exception:
        s.bind(S2_ADDR)
        print(S2_ADDR)
        listen(s, S1_ADDR)
    except KeyboardInterrupt:
        print('sys exit')
    finally:
        s.close()

if __name__ == '__main__':
    try:
        main()
    except Exception as e:
        print("error exit")
        traceback.print_exc()
    finally:
        print('end server')
```