哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	可靠数据传输协议-停等协议和 GBN 协议的实现				
姓名	王科龙		院系	计算机科学与技术	
班级	1803104		学号	1180801203	
任课教师	聂兰顺		指导教师	聂兰顺	
实验地点	格物 207		实验时间	2020/11/7	
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					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

（注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）

本次实验的主要目的。

1.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。

2. 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：

概述本次实验的主要内容，包含的实验项等。

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；（选作内容，加分项目，可以当堂完成或课下完成）

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。（选作内容，加分项目，可以当堂完成或课下完成）

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；（选作内容，加分项目，可以当堂完成或课下完成）

4）将所设计的GBN 协议改进为SR 协议。（选作内容，加分项目，可以当堂完成或课下完成）

实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

1) 包格式

```python
def make_pkt(data, seq):
    """
    数据格式
    seq data
    """
    pkt = str(str(seq) + " " + str(data))
    return pkt.encode('utf-8')


def make_ack(seq):
    """
    ack seq
    """
    ack = str(str(seq) + " ack")
    return ack.encode('utf-8')
```

2) 停等协议存在发送方和接受方，同时发送方要接受ack来检查。但是因为要实现双向传输，所以可以将发送方和接受放到一个类里，然后实例化是实例同一个类，但是调用不同的函数实现发送和接受。

```
class StopWait:
    def __init__(self, name, my_ip, my_port, remote_ip, remote_port):...

    def get_data(self):...

    def __send(self, data):...

    def begin_send(self):...

    def begin_file_dis(self):...

    # 接收数据或者返回ack
    def __receive(self):...

    def __receive_file(self, file, rcv_size):...

    def receive_with_loss(self):...

    def begin_receive_with_loss(self):
        while True:
            self.receive_with_loss()

    def begin_receive(self):...

    def begin_receive_file(self):...

    def __timeout(self):...
```

3) 为了防止接受ack和接受数据冲突，导致ack被接受数据的recvfrom接收到，从而无法进行确认，所以将接受ack和接受数据放到了一起，解析出来数据之后看是否是ack，是ack的话就转到ack的处理程序上去。

```
ret_seq = int(message[0])
if len(message) <= 1 or 'ack' != message[1]:  # 正常接受数据 /或者ack
    if is_rcv < 0.3:  # 接受之后不处理数据
        print("refuse pkt", ',content=', message)
        return
    if ret_seq == self.except_seq:
        print(self.name, 'receive expected pkt,content=', message)
        self.except_seq = (self.except_seq + 1) % 2
    else:
        print('NO expected pkt,expect_seq=', self.except_seq, 'content=', message)
    is_ack = random.random()
    ack = make_ack(ret_seq)
    if is_ack < 0.5:  # 有0.5的概率不发送ack报文
        print('Send ACK,content=', ack)
        self.socket.sendto(ack, addr)
    else:
        print('no_ack=', is_ack, 'ack_message=', ack)
else:  # 获取到ack
    print('receive ACK,seq=', ret_seq, 'wait ACK=', self.seq)
    if ret_seq == self.seq:
        self.seq = (self.seq + 1) % 2
        self.wait_ack = False
```

4) 模拟引入丢包，丢包有2种，1是丢失pkt，2是丢失ack。丢失pkt可以通过不接受数据模拟，丢失ack可以通过不发送ack模拟

```
if is_rcv < 0.3:  # 接受之后不处理数据
    print("refuse pkt", ',content=', message)
    return
```

```
is_ack = random.random()
ack = make_ack(ret_seq)
if is_ack < 0.5:  # 有0.5的概率不发送ack报文
    print('Send ACK,content=', ack)
    self.socket.sendto(ack, addr)
else:
    print('no_ack=', is_ack, 'ack_message=', ack)
```

5) 收发文件:由于原本的程序主要是通过在控制台输入文字来交互，并没有考虑收发文件的事情，因此重新写函数来做收发文件。一个要解决的重要问题是，什么时候socket停止接受文件，如果没有标志的话，接受数据的循环无法接受，在python中不关闭文件是无法将数据写入文件中的。

我的主要解决方案是:

Server端发送:

(1) Server端发送’file’表示接下来要发送文件

(2) Server端读取文件的大小和名字发给Client，让Client利用文件大小来确定是否读完的所有的数据

(3) 发送真实的文件内容

```
def begin_file_dis(self):
    path = input('请输入要发送的文件路径\n')
    filesize_bytes = os.path.getsize(path)
    self.__send('file')
    time.sleep(2)
    self.__send(path + " " + str(filesize_bytes))
    time.sleep(2)
    with open(path, 'rb') as file:
        while True:
            data = file.read(1024).decode()
            if len(data) <= 0: break
            self.__send(data)
            time.sleep(2)
```

由于一般的文件收发都是基于TCP的，需要保证不能有乱序分组出现，但是这个实验要求基于UDP，因此通过使线程睡眠来保证可以准时收到，按序发送

缺点：由于make_pkt只能打包字符串数据，因此无法收发图片

Client接受：

```python
def begin_receive_file(self):
    while True:
        order_mes = self.__receive()
        if order_mes == 'file':
            break
    print('order=', order_mes)
    while True:
        message = self.__receive().split()
        if len(message) == 2:
            break
    print('message=', message)
    paths = message[0].split('.')
    new_file_name = ''
    for i in range(len(paths) - 1):
        if i == len(paths) - 2:
            new_file_name = new_file_name + paths[i] + '(new).'
        else:
            new_file_name = new_file_name + paths[i] + '.'
    new_file_name = new_file_name + paths[len(paths) - 1]
    file_size = int(message[1])
    rcv_size = 0
    file = open(new_file_name, 'wb')
    print(new_file_name)
    print('rcv_size=', rcv_size)
    while rcv_size < file_size:
        rcv_size = self.__receive_file(file, rcv_size)
    file.close()
```

6) Gbn类的设计思路和停等类相同，只不过遵循的逻辑是gbn的收发逻辑。
发送方通过移动窗口来控制已经发送未确认的分组，保证这些分组收到ack或者是超时重发所有已发送但是未确认的分组。

```
def __send(self, data):
    readable, writeable, errors = select.select([], [self.socket, ], [], 1)
    if len(writeable) > 0:
        if self.next_seq_num < MAX_SEQ_NUM:  # 还有可以使用的序号
            pass
        else:
            if self.base >= MAX_SEQ_NUM:  # 序号内的所有分组都已经被接收
                self.base = 0
                self.next_seq_num = 0
            else:
                print('序号用尽,请等待')
                # self.timeout()
                return
        if self.next_seq_num < self.base + self.n:  # 窗口内还有序号
            pkt = make_pkt(data, self.next_seq_num)
            self.socket.sendto(pkt, (self.remote_ip, self.remote_port))
            self.send_flag = True
            print(self.name, 'Send,pkt seq=', self.next_seq_num, 'base=', self.base, 'next_seq_num=',
                  self.next_seq_num, 'timer=', self.timer)
            self.send_cache[
                self.next_seq_num % self.n] = pkt  # 如果窗口大小是5，->0,1,2,3,4 当next_seq_num=5时，这个数
            if self.next_seq_num == self.base:  # 没有未确认的数据，重启定时器
                self.timer = 0
            self.next_seq_num = self.next_seq_num + 1
        else:
            print('窗口满！')
```

接受方拥有一个expect_seq_num，只有发送的序号和expect_seq_num对上了才会接受数据

```
readable, writeable, errors = select.select([self.socket, ], [], [], 1)

if len(readable) > 0:
    is_rcv = random.random()
    byte, addr = self.socket.recvfrom(1024)
    message = byte.decode().split()
    ret_seq = int(message[0])
    if len(message) <= 1 or 'ack' != message[1]:  # 分开接受 ack 和接受数据
        if is_rcv < 0.3:  # 接受之后不处理数据
            print("refuse_pkt", ',content=', message)
            return
        # print(self.name, 'receive data,seq=', ret_seq)
        if ret_seq == self.except_seq_num:
            self.ack = make_ack(self.except_seq_num)  # 构造返回的 ack
            print(self.name, ' receive expected data,seq=',
self.except_seq_num, ',content=\"', message, '\"')
            self.except_seq_num = (self.except_seq_num + 1) % MAX_SEQ_NUM
        else:
            print('NO expected data,content=', message)
        is_ack = random.random()
        if is_ack < 0.5:  # 有 0.5 的概率不发送 ack 报文
            self.socket.sendto(self.ack, (self.remote_ip,
self.remote_port))
        else:
            print('no_ack=', is_ack, 'ack_message=', self.ack)
```

```
    else:
        self.base = ret_seq + 1
        print(self.name, 'receive ACK,seq=', ret_seq, 'base=', self.base,
'next_seq_num=',
              self.next_seq_num)
        if self.base == self.next_seq_num:
            self.timer = -1  # 收到ack之后后面没有未确认的数据，停止计时器
        else:
            self.timer = 0  # 收到ack之后，后面还有未确认的数据，重启计时器
else:
    if self.next_seq_num > self.base:  # 读不到资源(ack)并且存在未确认的数据
        self.timer = self.timer + 1
        if self.timer > MAX_TIMER:
            self.timeout()
```

7）sr协议不再超时之后发送所有的未确认分组，我的程序里超时之后只发送第一个未确认的分组，同时对于乱序收到的分组，会将其先缓存下来。每收到一个分组之后，接受方的rcv_base就会移动到第一个目前没有收到分组的序号处，同时将这中间乱序收到没有上交的数据一并上交。

```
# 更新rcv_base
for i in range(self.n):
    if self.rcv_log[self.rcv_base % self.n] == 1:
        # 上交数据
        print('deliver data,seq=', self.rcv_base, 'content=',
              self.rcv_cache[self.rcv_base % self.n])
        self.rcv_log[self.rcv_base % self.n] = 0  #
        if self.rcv_base < MAX_SEQ_NUM:
            self.rcv_base = self.rcv_base + 1
            if self.rcv_base == MAX_SEQ_NUM:  # 只会收到0~MAX_SEQ_NUM序号的字段
                self.rcv_base = 0
    else:
        break
```

实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

1)文件收发

sw_client sw_server

```
E:\桌面文件\文档\大三秋季学期\计算机网络\la
请输入要发送的文件路径
../waste.txt
Sender Send Seq= 0
receive ACK,seq= 0 wait ACK= 0
Sender Send Seq= 1
receive ACK,seq= 1 wait ACK= 1
Sender Send Seq= 0
receive ACK,seq= 0 wait ACK= 0
Sender Send Seq= 1
receive ACK,seq= 1 wait ACK= 1
```

sw_client sw_server

```
client1,请输入要发送的信息:
client1 receive expected pkt,content= ['0', 'file']
Send ACK,ack is b'0 ack'
order= file
client1 receive expected pkt,content= ['1', '../waste.txt', '1152']
Send ACK,ack is b'1 ack'
message= ['../waste.txt', '1152']
../waste(new).txt
rcv_size= 0
client1 receive expected pkt,content= ['0', 'gbn', 'class:', '1.', 'd
rcv_size= 1024
Send ACK,ack is b'0 ack'
client1 receive expected pkt,content= ['1', '+', '1', 'if', 'self.tim
rcv_size= 1152
Send ACK,ack is b'1 ack'
```

waste(new).txt

```
gbn class:
1.
  def check_ack(self):
        while True:
            self.__rcv_ack()  # 检查ack
2.
 def __rcv_ack(self):
        # 接受ack  非阻塞方式
        readable, writeable, errors = select.
        if self.send_flag:
            if len(readable) > 0:
                byte, addr = self.socket.recv
                ack_message = byte.decode()
                if 'ack' in ack_message:
                    ack_message = ack_message
                    ret_seq = int(ack_message
                    print(self.name, 'receive
                          self.next_seq_num)
```

waste.txt

```
gbn class:
1.
  def check_ack(self):
        while True:
            self.__rcv_ack()  # 检查ack
2.
 def __rcv_ack(self):
        # 接受ack  非阻塞方式
        readable, writeable, errors = select.select
        if self.send_flag:
            if len(readable) > 0:
                byte, addr = self.socket.recvfrom(1
                ack_message = byte.decode()
                if 'ack' in ack_message:
                    ack_message = ack_message.split
                    ret_seq = int(ack_message[0])
                    print(self.name, 'receive ACK,s
                          self.next_seq_num)
```

2) gbn双向通信和丢包演示

丢包

```
client1,请输入要发送的信息:
n
client1 Send,pkt seq= 2 base= 2 next_seq_num= 2 timer= -1
client1,请输入要发送的信息:
n
client1 Send,pkt seq= 3 base= 2 next_seq_num= 3 timer= 2
client1,请输入要发送的信息:
Sender Time Out,seq= 2
client1 receive ACK,seq= 2 base= 3 next_seq_num= 4
client1 receive ACK,seq= 3 base= 4 next_seq_num= 4
n
```

```
server,请输入要发送的信息:
server  receive expected data,seq= 0 ,content=" ['0', 'n'] "
server  receive expected data,seq= 1 ,content=" ['1', 'n'] "
server  receive expected data,seq= 2 ,content=" ['2', 'n'] "
no_ack= 0.9982002530515434 ack_message= b'2 ack'
refuse_pkt ,content= ['3', 'n']
NO expected data,content= ['2', 'n']
server  receive expected data,seq= 3 ,content=" ['3', 'n'] "
refuse_pkt ,content= ['4', 'n']
NO expected data,content= ['5', 'n']
server  receive expected data,seq= 4 ,content=" ['4', 'n'] "
server  receive expected data,seq= 5 ,content=" ['5', 'n'] "
```

同时server也是可以向client发送数据的

```
server,请输入要发送的信息:
你好
server Send,pkt seq= 1 base= 1 next_seq_num= 1 timer= -1
server receive ACK,seq= 1 base= 2 next_seq_num= 2
```

```
client1  receive expected data,seq= 1 ,content=" ['1', '你好'] "
```

3)sr协议效果展示：

发送端快速发送，但是由于丢包使得窗口满：

```
窗口满!
client1,请输入要发送的信息:
Sender Time Out,seq= 0
client1 receive ACK,seq= 0 base= 2 next_seq_num= 5
Sender Time Out,seq= 2
Sender Time Out,seq= 2
client1 receive ACK,seq= 2 base= 4 next_seq_num= 5
Sender Time Out,seq= 4
client1 receive ACK,seq= 4 base= 5 next_seq_num= 5
```

接受方：

```
server,请输入要发送的信息:
server  receive data in window,seq= 0 rcv_base= 0 rcv_end= 5 ,content=" ['0', 'ni'] "
deliver data,seq= 0 content= ['0', 'ni']
no_ack= 0.5670637206542852 ack_message= b'0 ack'
server  receive data in window,seq= 1 rcv_base= 1 rcv_end= 6 ,content=" ['1', 'ni'] "
deliver data,seq= 1 content= ['1', 'ni']
Send ACK,content= b'1 ack'
refuse pkt ,content= ['2', 'n']
server  receive data in window,seq= 3 rcv_base= 2 rcv_end= 7 ,content=" ['3', 'n'] "
Send ACK,content= b'3 ack'
refuse pkt ,content= ['4', 'n']
refuse pkt ,content= ['0', 'ni']
Send ACK,content= b'0 ack'
refuse pkt ,content= ['2', 'n']
server  receive data in window,seq= 2 rcv_base= 2 rcv_end= 7 ,content=" ['2', 'n'] "
deliver data,seq= 2 content= ['2', 'n']
deliver data,seq= 3 content= ['3', 'n']
Send ACK,content= b'2 ack'
server  receive data in window,seq= 4 rcv_base= 4 rcv_end= 9 ,content=" ['4', 'n'] "
deliver data,seq= 4 content= ['4', 'n']
Send ACK,content= b'4 ack'
```

由于server没有返回0ack，使得发送方0分组超时，在这之前，接受方又缓存了1号分组和3号分组，因为0，1号分组都已经接受，可以进行提交，没有收到2号分组，在发送方2分组超时重发，接受方收到2号分组之后，将2号和3号分组一起上交。rcv_base移动到4

问题讨论：

对实验过程中的思考问题进行讨论或回答。

1. 线程的阻塞问题，由于recvfrom函数默认是阻塞的，如果没有从系统内核中拿到数据，就会一直阻塞其他进程，导致发送进程也无法进行发送。
   解决方法是通过selcet函数去查看socket句柄是否有变化，有变化，存在可读取数据之后在去读取数据。

```
readable, writeable, errors = select.select([self.socket, ], [], [], 1)
if len(readable) > 0:
```

心得体会：

结合实验过程和结果给出实验的体会和收获。

要想完整地写完一份代码，要求要又尽量少地错误，一份伪代码或者是流程图之类的东西是必要的，这些可以准确的描述要实现的代码逻辑，将这些东西只放在脑子里是不可靠的，有时候忘记写了一句导致了bug，然后这个bug就有可能花费很长时间去寻找，但是其实上是可以避免的。

通过对于停等协议，gbn协议和sr协议的实现，我对可靠传输数据原理有了更加深刻的认识。