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-GBN 协议的设计与实现				
姓名	张明远		院系	计算机科学与技术	
班级	2003104		学号	120L030501	
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	2022.10.24	
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：

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。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：

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. 将所设计的 GBN 协议改进为 SR 协议。

实验过程：

一、GBN 和 SR 协议数据分组格式

首先定义协议数据格式，如下图所示

GBN 协议数据分组由序列号，数据，检验和组成。序列号标志着数据的分组序号，数据为报文的数据字段，检验和经由数据和序列号生成，用以保障数据分组无比特差错。

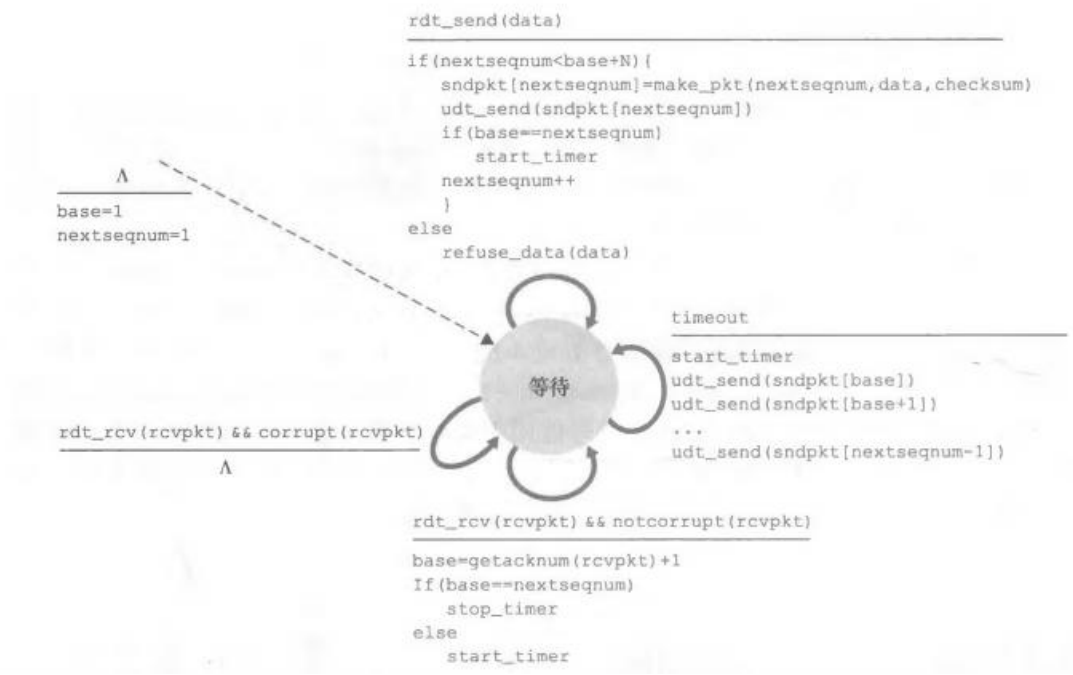
二、确认分组格式

确认分组格式如下图所示

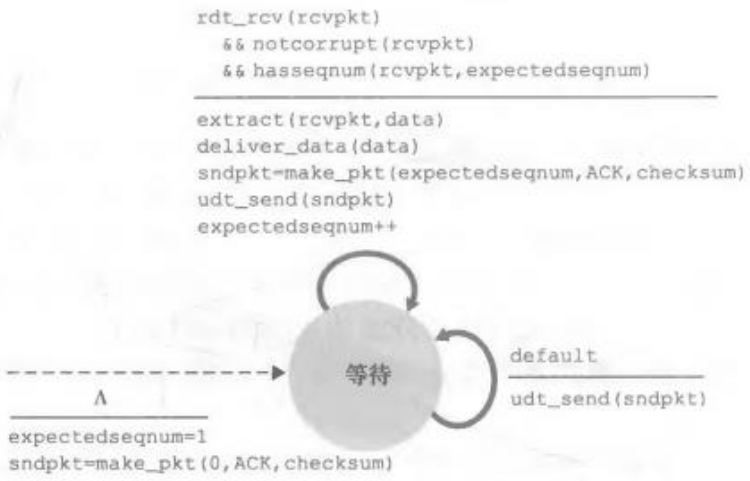
确认分组由序列号，ACK 和检验和组成，序列号标志着下一个期望确认的数据的分组序号，ACK 为报文的确认字段，检验和经由数据和序列号生成，用以保障数据分组无比特差错。

三、协议两端程序流程图

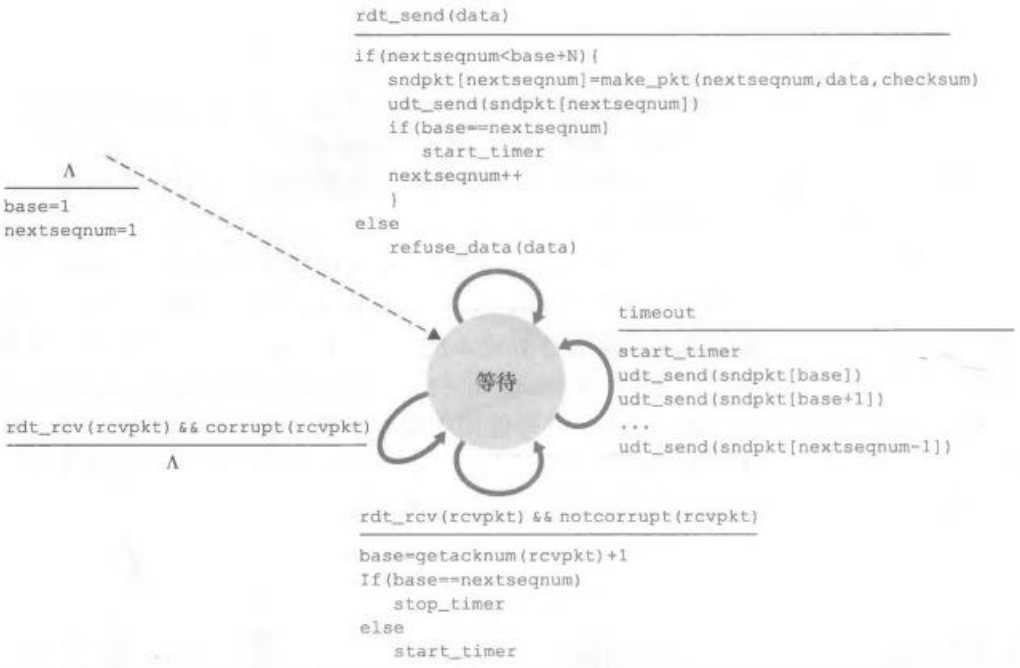
gbn 发送方的流程图



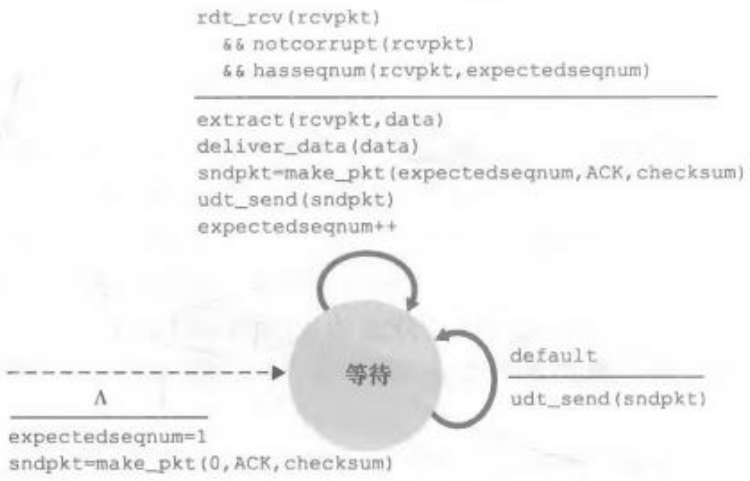
gbn 接受方的流程图



sr 发送方的流程图



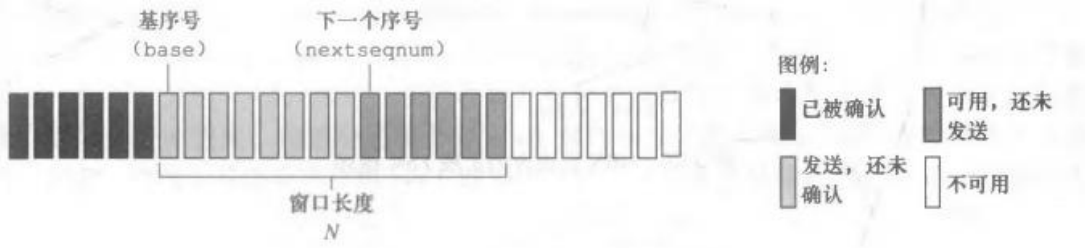
sr 接受方的流程图

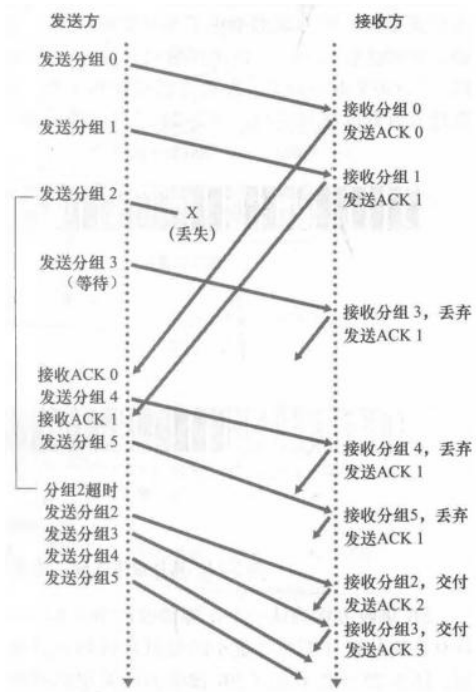


四、协议典型交互过程

与简单停等协议相比，GBN 协议事实上就是把发送窗口的长度设置为 n，而接收窗口的长度仍保持为 1，也就是 GBN 协议不接受乱序到达的数据，如果数据乱序到达，那么在接受处将会决定放弃这一部分数据；而 GBN 协议采用的是累积确认的方式，也就是说如果当前等待接收的数据编号为 1,2,3，而收到了编号为 3 的 ack，那么就表示这三个数据已经都正确接收了。

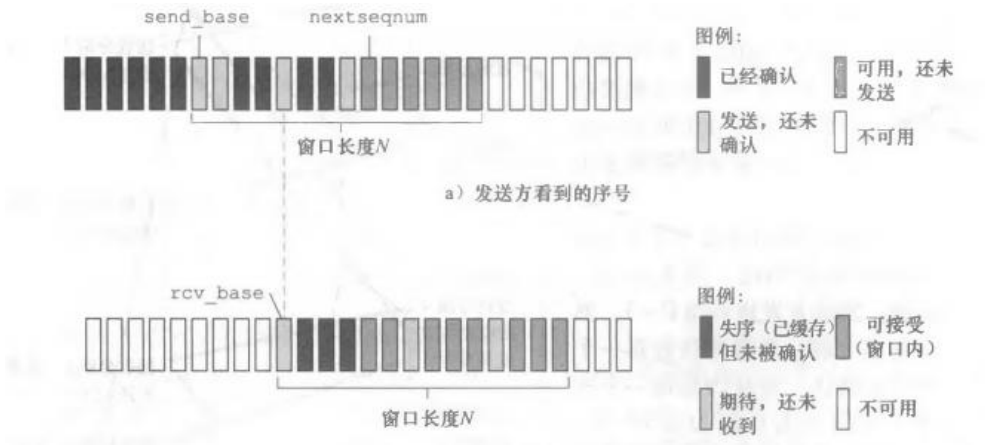
gbn 协议：

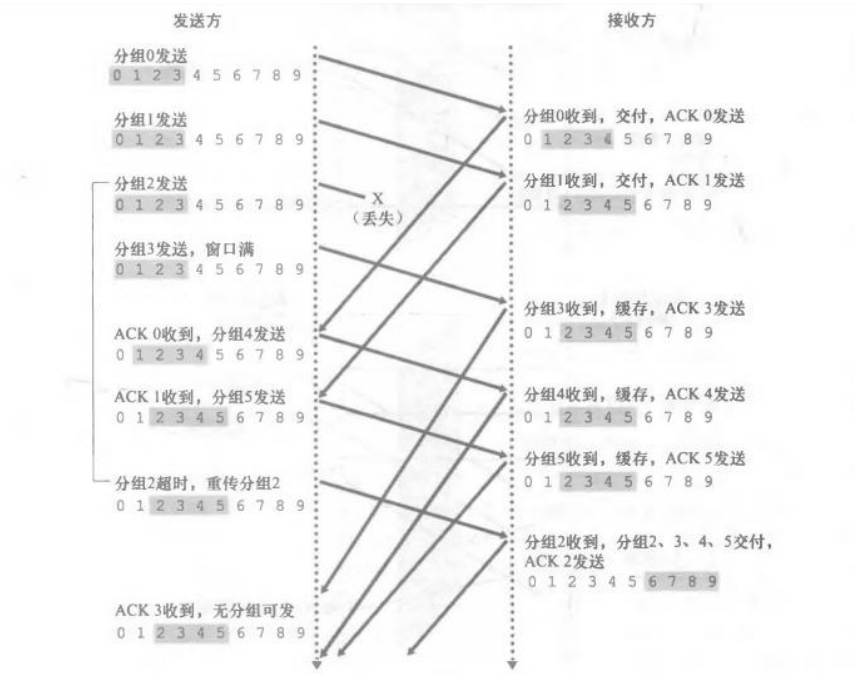




对于 SR 接收方，只要是接收窗口内的帧都可以接收，SR 接收方将确认一个正确接收的帧而不管其是否按序,失序的帧将被缓存,并返回给发送方一个该帧的确认帧,直到所有帧(序号更小的帧)都被收到为止,这时将一批帧顺序交付上层,然后向前移动滑动窗口。

sr 协议：





五、数据分组丢失模拟方法

使用随机函数进行模拟，使得发送方和接受方在对分组作出反应时有一定概率不做反应，这样就可以模拟数据分组以一定的概率丢失的情况

python 实现：

```
# 概率丢包
def loss_pkt():
    return random.randint(0, 9) == 0

#使用举例：
if loss_pkt():
    return
```

六、程序实现的主要类（或函数）及其主要作用

1. 停等协议

当设置服务器端发送窗口的大小为 1 时，GBN 协议就是停-等协议，故停等协议不再说明。

2. GBN 协议

GBN 协议的实现主要为类 rdt_gbn，类的定义如下：

```
class rdt_gbn:
    def __init__(self, address):
        # 对面地址
        self.address_send = None
        # 本机地址
        self.ip = address[0]
        self.port = address[1]
        # socket 设置
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.server_socket.bind(address)
        self.server_socket.settimeout(10)
        self.server_socket.setblocking(False)
        # 滑动窗口设置
        self.base = 0
        self.next_seq_num = 0
        self.window = 10
        self.expect = 0
        self.frag = 30
```

rdt_gbn通过提供sendall方法和recv方法实现了gbn协议下的双向数据传输。
sendall方法将数据分为片段，再依据滑动窗口的位置进行发送数据
recv方法依据接受窗口对分片数据进行接收，然后进行聚合
python实现：

```
# 发送数据
def sendall(self, data):
    # 初始化
    data = bytes(data, encoding='utf-8')
    self.base = 0
    self.next_seq_num = 0
    self.expect = 0
    # 数据切片
    data_list = []
    while data != b'':
        data_list.append(data[:self.frag])
        data = data[self.frag:]
    # 发送数据
```

```
    while True:
        # 结束本次发送
        if self.base >= len(data_list):
            self.__send(self.base, b'#__END__#', self.address_send)
            break
        # 发送可以发送的数据包
        if self.base == self.next_seq_num:
            # 设置计时器
            start = time.time()
            # 发送数据
            for i in range(self.window):
                self.__send(self.next_seq_num, data_list[self.next_seq_num],
self.address_send)
                self.next_seq_num += 1
                # 限制范围
                if self.next_seq_num >= len(data_list):
                    break
        # 检查超时
        if time.time() - start >= 1:
            self.next_seq_num = self.base
        # 接受 ack
        try:
            message = self.server_socket.recvfrom(1024)
            # 模拟丢包
            if loss_pkt():
                continue
            [seg_last, data, checksum] = analyse_pkt(message[0])
            seg = bytes(str(self.base + 1), encoding='utf-8')
            cs = make_checksum(data)
            # 记录收到的数据包
            if seg_last >= seg and checksum == cs:
                # 更新 base
                self.base = int(seg_last)
                print('已发送' + str(self.base - 1) + ' ', data_list[self.base - 1])
        except:
            pass

# 接受数据
def recv(self):
    self.base = 0
    self.next_seq_num = 0
    self.expect = 0
    recv_list = b''
    while True:
```

```
            data = None
            while data is None:
                data = self.rdt_recv(self.address_send)
            if data == b'#__END__#':
                break
            # 整合数据
            recv_list += data
        return recv_list
```

---

3. SR 协议

GBN 协议的实现主要为类 rdt_gbn，类的定义如下：

```
class rdt_sr:
    def __init__(self, address):
        # 对面地址
        self.address_send = None
        # 本机地址
        self.ip = address[0]
        self.port = address[1]
        # socket 设置
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.server_socket.bind(address)
        self.server_socket.settimeout(10)
        self.server_socket.setblocking(False)
        # 滑动窗口设置
        self.base = 0
        self.next_seq_num = 0
        self.window = 10
        self.expect = 0
        self.time_limit = 1
        self.frag = 30
```

---

rdt_gbn通过提供sendall方法和recv方法实现了gbn协议下的双向数据传输。
sendall方法将数据分为片段，再依据滑动窗口的位置进行发送数据
recv方法依据接受窗口对分片数据进行接收，然后进行聚合
python实现：

```
# 发送数据
def sendall(self, data):
    # 初始化
```

```
        data = bytes(data, encoding='utf-8')
        self.base = 0
        self.next_seq_num = 0
        self.expect = 0
        data_list = []
        timer_dict = {}
        reset_dict = {}
        # 数据切片
        while data != b'':
            data_list.append(data[:self.frag])
            data = data[self.frag:]
        # 发送数据
        while True:
            # 结束本次发送
            if self.base >= len(data_list):
                self.__send(self.base, b'#__END__#', self.address_send)
                break
            # 发送可以发送的数据包
            if self.next_seq_num < self.base + self.window:
                for i in range(self.window):
                    if self.next_seq_num >= len(data_list):
                        continue
                    # 发送数据
                    self.__send(self.next_seq_num, data_list[self.next_seq_num],
self.address_send)
                    # 设置计时器
                    timer_dict[data_list[self.next_seq_num]] = (time.time(), self.next_seq_num)
                    self.next_seq_num += 1
                    # 限制范围
                    if self.next_seq_num >= len(data_list):
                        break
            # 检查超时
            for key, timer in timer_dict.items():
                if timer[0] is None:
                    reset_dict[key] = timer
                    continue
                if time.time() - timer[0] >= self.time_limit:
                    self.__send(timer[1], key, self.address_send)
                    reset_dict[key] = (time.time(), timer[1])
            # 修改计时器
            for key, timer in reset_dict.items():
                if timer[0] is None:
                    del timer_dict[key]
                timer_dict[key] = timer
```

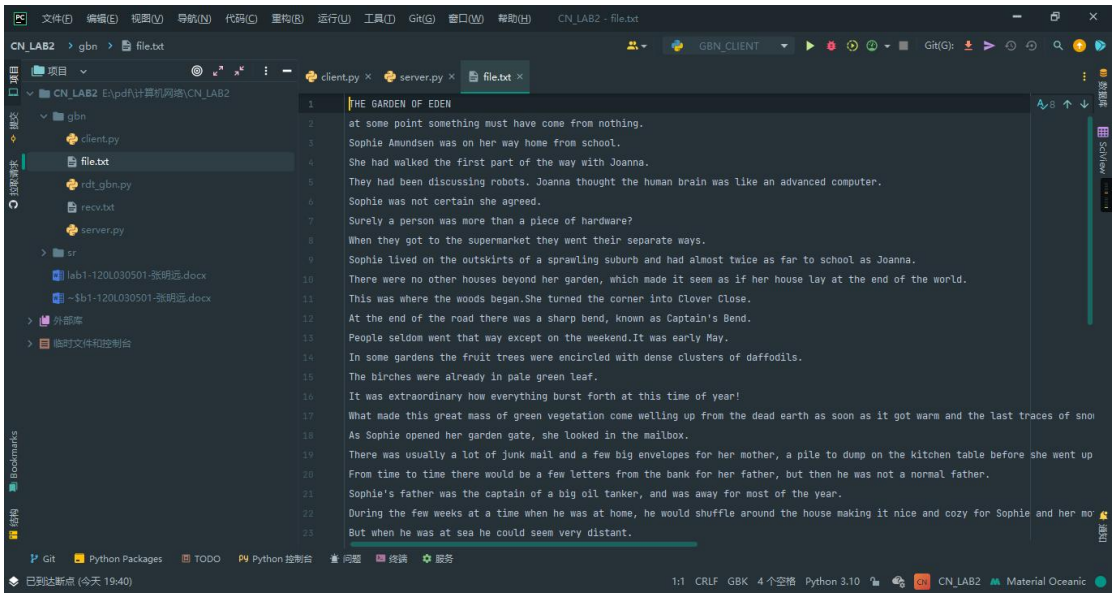
```
            # 接受 ack
            try:
                message = self.server_socket.recvfrom(1024)
                # 模拟丢包
                if loss_pkt():
                    continue
                [seg_last, data, checksum] = analyse_pkt(message[0])
                cs = make_checksum(data)
                seg = int(seg_last) - 1
                # 记录收到的数据包
                if seg >= self.base and seg < self.next_seq_num:
                    timer_dict[data_list[seg]] = (None, timer_dict[data_list[seg]][1])
                    print('已发送' + str(seg) + ' ', data_list[seg])
                # 更新 base
                pos = 0
                for i in range(self.base, self.next_seq_num):
                    if timer_dict[data_list[i]][0] is None:
                        if i == self.base + pos:
                            pos += 1
                self.base += pos
            except:
                pass

# 接受数据
def recv(self):
    self.base = 0
    self.next_seq_num = 0
    self.expect = 0
    recv_list = {}
    while True:
        data = None
        while data is None:
            data, pos = self.rdt_recv(self.address_send)
        if data == b'#__END__#':
            break
        recv_list[pos] = data
    # 整合数据
    result = b''
    for i in range(len(recv_list)):
        result += recv_list[i]
    return result

-----------------------------------------------------------------------------------------
```

实验结果：

本环节采用的验证方法为文件传输，，采用文件为一篇文本，如下图：

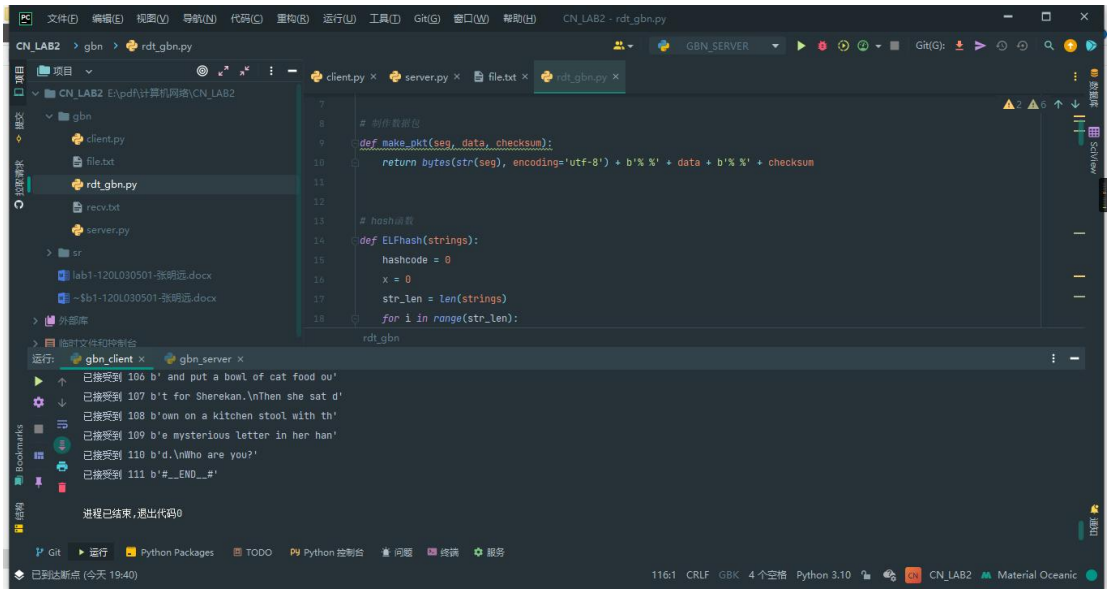


由服务器端程序，向客户端程序发送这段文本，然后客户端程序将其保存在本地。

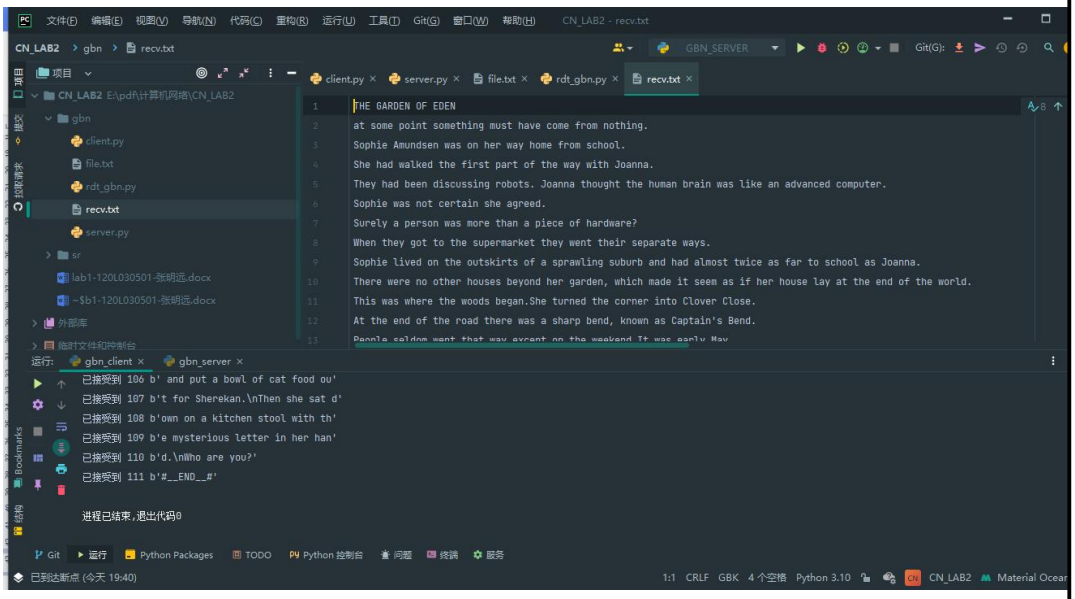
1. GBN：

由于停等协议就是 GBN 协议中发送窗口长度设置为 1，因此停等协议与 GBN 协议验证过程类似，统一演示如下：

下图为gbn协议运行结果，可以看到文本被完整的传送到了客户端

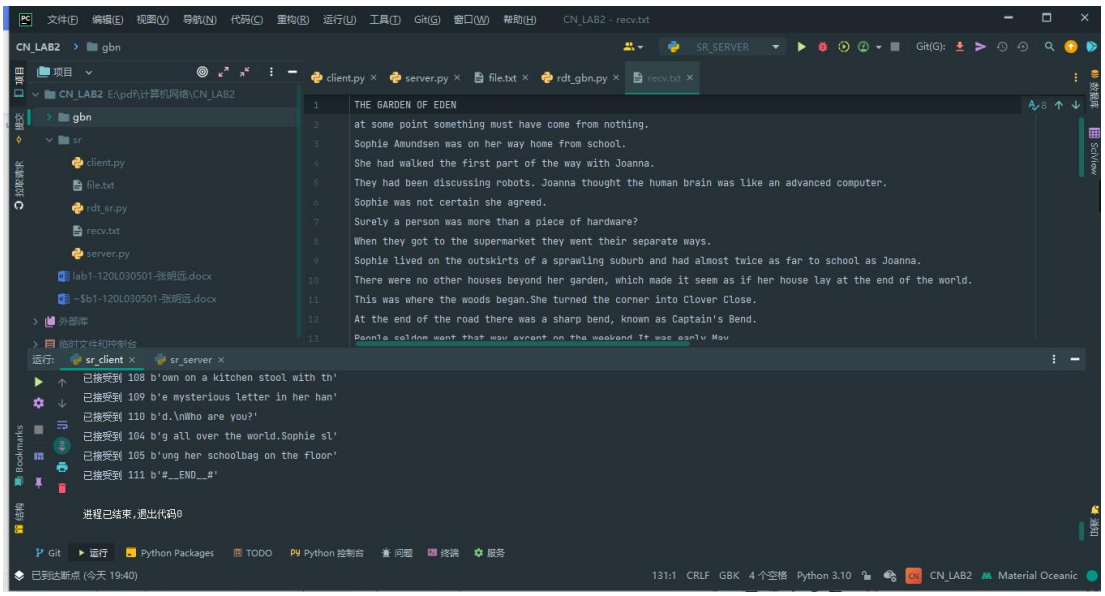


下为客户端生成的本地文件，可见与原文件相同：

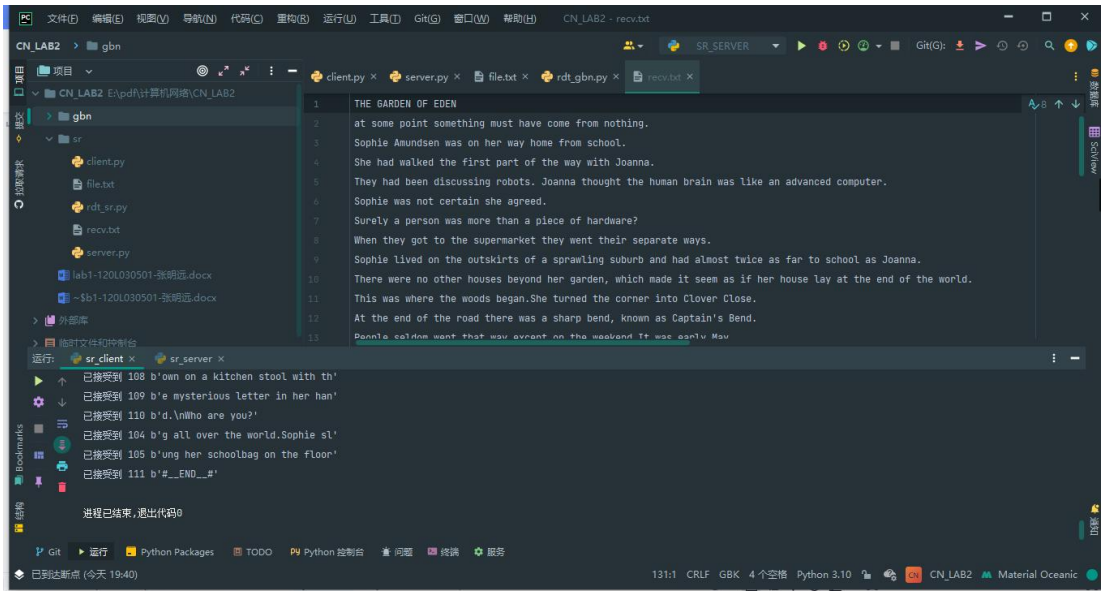


2. **SR：**：

下图为sr协议运行结果，可以看到文本被完整的传送到了客户端



下为客户端生成的本地文件，可见与原文件相同：



问题讨论：

- 由于 UDP 协议是无连接的传输协议，无法保证可靠的数据传输，因此如果在上层使用的时候需要上层来实现可靠的数据传输的部分。而且和 TCP 协议不同，UDP 是一种无连接的协议，因此在传输数据之前并不需要三次挥手建立连接，尽管在本次实验中一部分代码确实使用了类似于连接的方式实现，但是需要注意的是那只是逻辑上的连接，并不存在真实的连接过程。
- SR 协议的出现主要就是为了解决在使用 GBN 的过程中的一定量的资源浪费的情况，事实上 SR 协议、GBN 协议、停等协议这三者在本质上的区别就是接收窗口和发送窗口大小的区别，至于其他的一些区别都是一些小的区别。
- 需要注意的是因为 SR 协议为每一个发送窗口的数据都设置了一个计时器，每次都重传超时的部分。在实际实现的过程中每次只需要比对发送窗口最低位是否超时，如果超时则重

传，如果没有超时则表示发送窗口中没有超时的数据。

心得体会：
通过本次实验对于基于 UDP 协议的可靠数据传输有了更加深刻的认识，掌握了停等协议，GBN 协议以及 SR 协议，同时对于 socket 编程也有了更深入的了解。

附源代码：
rdt_gbn.py

```
# coding:gbk
import random
import socket
import time
import select


# 制作数据包
def make_pkt(seg, data, checksum):
    return bytes(str(seg), encoding='utf-8') + b'% %' + data + b'% %' + checksum


# hash 函数
def ELFhash(strings):
    hashcode = 0
    x = 0
    str_len = len(strings)
    for i in range(str_len):

        hashcode = (hashcode << 4) + ord(strings[i])
        x = hashcode & 0xF000000000000000
        if x:
            hashcode ^= (x >> 56)
            hashcode &= ~x
    return hashcode


# 生成检验和
```

```
def make_checksum(data):
    return bytes(str(ELFhash(str(data, encoding='utf-8'))), encoding='utf-8')


# 解析数据包
def analyse_pkt(message):
    al = message.split(b'% %')
    return al


# 概率丢包
def loss_pkt():
    return random.randint(0, 9) == 1


# gbn 协议
class rdt_gbn:
    def __init__(self, address):
        # 对面地址
        self.address_send = None
        # 本机地址
        self.ip = address[0]
        self.port = address[1]
        # socket 设置
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.server_socket.bind(address)
        self.server_socket.settimeout(10)
        self.server_socket.setblocking(False)
        # 滑动窗口设置
        self.base = 0
        self.next_seq_num = 0
        self.window = 10
        self.expect = 0
        self.frag = 30

    # 发送数据包
    def __send(self, total, data, address):
        checksum = make_checksum(data)
        sndpkt = make_pkt(total, data, checksum)
        self.server_socket.sendto(sndpkt, address)

    # 设置对面地址
    def set_add_sen_to(self, address):
        self.address_send = address
```

```
# 接受单个数据
def rdt_recv(self, address):
    while True:
        try:
            message = self.server_socket.recvfrom(1024)
            # 丢包
            if loss_pkt():
                return None
            # 解析
            [seg_last, data, checksum] = analyse_pkt(message[0])
            seg = bytes(str(self.expect), encoding='utf-8')
            cs = make_checksum(data)
            # 接收数据并发送 ack
            if seg_last == seg and checksum == cs:
                self.expect += 1
                print('已接受到', str(seg_last, encoding='utf-8'), data)
                self.__send(self.expect, b'ACK', address)
                return data
            if seg_last < seg and checksum == cs:
                self.__send(self.expect, b'ACK', address)
            self.__send(self.expect, b'ACK', address)
        except:
            pass

# 发送数据
def sendall(self, data):
    # 初始化
    data = bytes(data, encoding='utf-8')
    self.base = 0
    self.next_seq_num = 0
    self.expect = 0
    # 数据切片
    data_list = []
    while data != b'':
        data_list.append(data[:self.frag])
        data = data[self.frag:]
    # 发送数据
    while True:
        # 结束本次发送
        if self.base >= len(data_list):
            self.__send(self.base, b'#__END__#', self.address_send)
            break
        # 发送可以发送的数据包
```

```
                if self.base == self.next_seq_num:
                    # 设置计时器
                    start = time.time()
                    # 发送数据
                    for i in range(self.window):
                        self.__send(self.next_seq_num, data_list[self.next_seq_num],
self.address_send)
                        self.next_seq_num += 1
                        # 限制范围
                        if self.next_seq_num >= len(data_list):
                            break
                # 检查超时
                if time.time() - start >= 1:
                    self.next_seq_num = self.base
                # 接受 ack
                try:
                    message = self.server_socket.recvfrom(1024)
                    # 模拟丢包
                    if loss_pkt():
                        continue
                    [seg_last, data, checksum] = analyse_pkt(message[0])
                    seg = bytes(str(self.base + 1), encoding='utf-8')
                    cs = make_checksum(data)
                    # 记录收到的数据包
                    if seg_last >= seg and checksum == cs:
                        # 更新 base
                        self.base = int(seg_last)
                        print('已发送' + str(self.base - 1) + ' ', data_list[self.base - 1])
                except:
                    pass

    # 接受数据
    def recv(self):
        self.base = 0
        self.next_seq_num = 0
        self.expect = 0
        recv_list = b''
        while True:
            data = None
            while data is None:
                data = self.rdt_recv(self.address_send)
            if data == b'#__END__#':
                break
            # 整合数据
```

```
                recv_list += data
            return recv_list

rdt_sr.py
#coding:gbk
import random
import socket
import time
import select


# 制作数据包
def make_pkt(seg, data, checksum):
    return bytes(str(seg), encoding='utf-8') + b'% %' + data + b'% %' + checksum

# hash 函数
def ELFhash(strings):
    hashcode = 0
    x = 0
    str_len = len(strings)
    for i in range(str_len):

        hashcode = (hashcode << 4) + ord(strings[i])
        x = hashcode & 0xF000000000000000
        if x:
            hashcode ^= (x >> 56)
            hashcode &= ~x
    return hashcode


# 生成检验和
def make_checksum(data):
    return bytes(str(ELFhash(str(data,encoding='utf-8'))),encoding='utf-8')


# 解析数据包
def analyse_pkt(message):
    al = message.split(b'% %')
    return al


# 概率丢包
def loss_pkt():
    return random.randint(0, 9) == 0
```

```
# sr 协议
class rdt_sr:
    def __init__(self, address):
        # 对面地址
        self.address_send = None
        # 本机地址
        self.ip = address[0]
        self.port = address[1]
        # socket 设置
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.server_socket.bind(address)
        self.server_socket.settimeout(10)
        self.server_socket.setblocking(False)
        # 滑动窗口设置
        self.base = 0
        self.next_seq_num = 0
        self.window = 10
        self.expect = 0
        self.time_limit = 1
        self.frag = 30

    # 发送数据包
    def __send(self, total, data, address):
        checksum = make_checksum(data)
        sndpkt = make_pkt(total, data, checksum)
        self.server_socket.sendto(sndpkt, address)

    #设置对面地址
    def set_add_sen_to(self, address):
        self.address_send = address

    # 接受单个数据
    def rdt_recv(self, address):
        while True:
            try:
                message = self.server_socket.recvfrom(1024)
                # 丢包
                if loss_pkt():
                    return None, None
                # 解析
                [seg_last, data, checksum] = analyse_pkt(message[0])
                seg = str(seg_last, encoding='utf-8')
```

```
                    cs = make_checksum(data)
                    # 接收数据并发送 ack
                    if checksum == cs:
                        print('已接受到', seg, data)
                        self.__send(int(seg) + 1, b'ACK', address)
                        return data, int(str(seg_last, encoding='utf-8'))
                except:
                    pass

    # 发送数据
    def sendall(self, data):
        # 初始化
        data = bytes(data, encoding='utf-8')
        self.base = 0
        self.next_seq_num = 0
        self.expect = 0
        data_list = []
        timer_dict = {}
        reset_dict = {}
        # 数据切片
        while data != b'':
            data_list.append(data[:self.frag])
            data = data[self.frag:]
        # 发送数据
        while True:
            # 结束本次发送
            if self.base >= len(data_list):
                self.__send(self.base, b'#__END__#', self.address_send)
                break
            # 发送可以发送的数据包
            if self.next_seq_num < self.base + self.window:
                for i in range(self.window):
                    if self.next_seq_num >= len(data_list):
                        continue
                    # 发送数据
                    self.__send(self.next_seq_num, data_list[self.next_seq_num],
self.address_send)
                    # 设置计时器
                    timer_dict[data_list[self.next_seq_num]] = (time.time(),
self.next_seq_num)
                    self.next_seq_num += 1
                    # 限制范围
                    if self.next_seq_num >= len(data_list):
                        break
```

```
            # 检查超时
            for key, timer in timer_dict.items():
                if timer[0] is None:
                    reset_dict[key] = timer
                    continue
                if time.time() - timer[0] >= self.time_limit:
                    self.__send(timer[1], key, self.address_send)
                    reset_dict[key] = (time.time(), timer[1])
            # 修改计时器
            for key, timer in reset_dict.items():
                if timer[0] is None:
                    del timer_dict[key]
                timer_dict[key] = timer
            # 接受 ack
            try:
                message = self.server_socket.recvfrom(1024)
                # 模拟丢包
                if loss_pkt():
                    continue
                [seg_last, data, checksum] = analyse_pkt(message[0])
                cs = make_checksum(data)
                seg = int(seg_last) - 1
                # 记录收到的数据包
                if seg >= self.base and seg < self.next_seq_num:
                    timer_dict[data_list[seg]] = (None, timer_dict[data_list[seg]][1])
                    print('已发送' + str(seg) + ' ', data_list[seg])
                # 更新 base
                pos = 0
                for i in range(self.base, self.next_seq_num):
                    if timer_dict[data_list[i]][0] is None:
                        if i == self.base + pos:
                            pos += 1
                self.base += pos
            except:
                pass

    # 接受数据
    def recv(self):
        self.base = 0
        self.next_seq_num = 0
        self.expect = 0
        recv_list = {}
        while True:
            data = None
```

```
            while data is None:
                data, pos = self.rdt_recv(self.address_send)
            if data == b'#__END__#':
                break
            recv_list[pos] = data
        # 整合数据
        result = b''
        for i in range(len(recv_list)):
            result += recv_list[i]
        return result
```