实验报告

课程名称 ： **计算机网络原理**
实验题目 ： **可靠数据传输原理（GBN 或 SR）编程实验**
学号 ： **21281280**
姓名 ： **柯劲帆**
班级 ： **物联网2101班**
指导老师 ： **常晓琳**
报告日期 ： **2024年4月15日**

目录

1. 实验目的

运用各种编程语言实现基于 Go-Back-N 或 SR 的可靠数据传输软件。

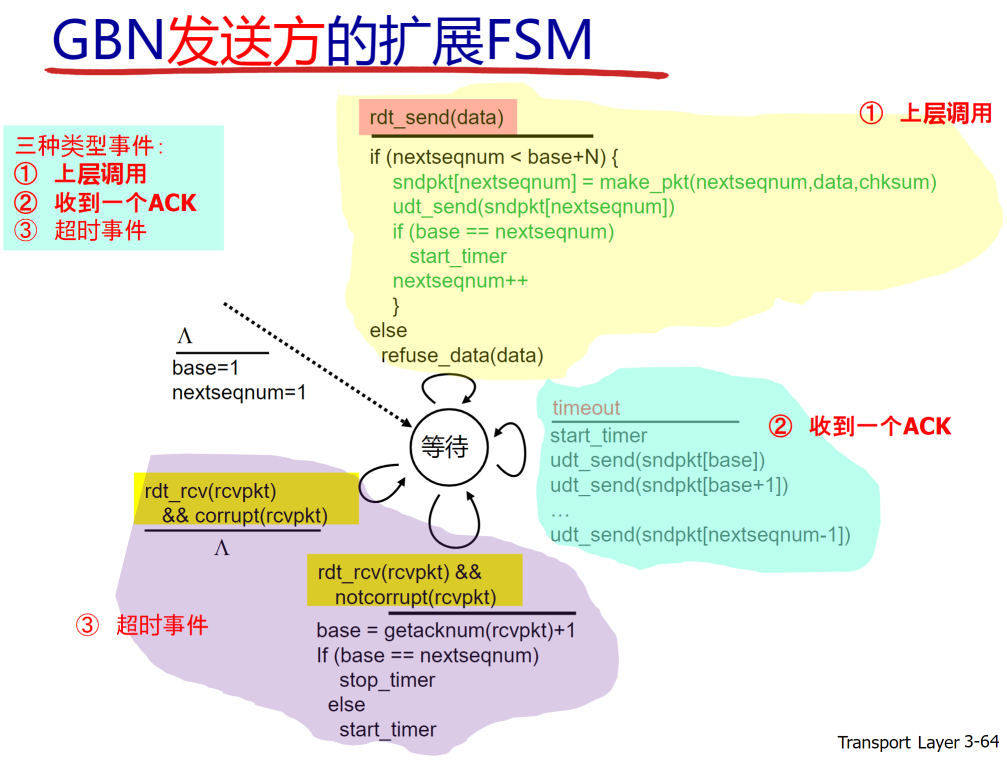
通过本实验，使学生能够对可靠数据传输原理有进一步的理解和掌握。

2. 实验环境

- **OS**：
  - Sender：Deepin (内核5.18.17-amd64-desktop-hwe)
  - Receiver：WSL2 (内核5.15.146.1-microsoft-standard-WSL2)
- **Python**：version 3.11.4

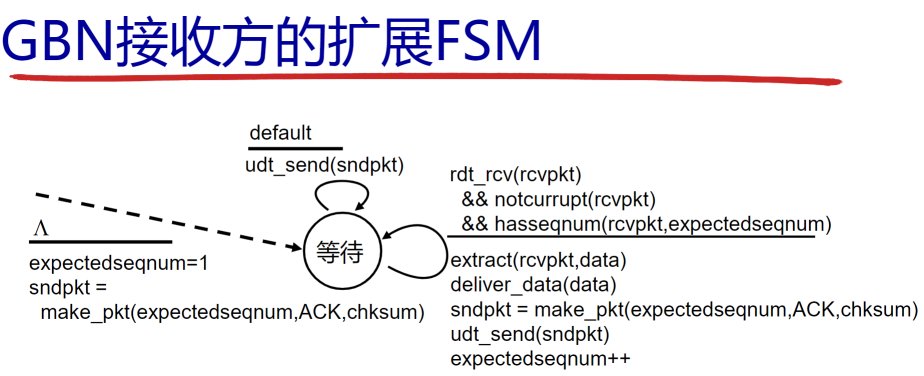
3. 实验原理

3.1. 发送方



1. 初始时，窗口基址`base`=1，下一个序列数据包编号`nextseqnum`=1；
2. 当GBN收到来自上层应用层的调用时：
    1. 如果`nextseqnum`在窗内：
        1. 将数据打包放入待发送数据包队列中；
        2. 将本数据包传给下层网络层发送；
        3. 如果`base`和下一个序列数据包编号`nextseqnum`相等，说明刚刚开始有数据包开始发送，所以开始计时；
        4. 下一个序列数据包编号`nextseqnum`增1；
    2. 否则拒绝上层应用层的调用；
3. 当超时时：
    1. 计时器重新开始计时；
    2. 将`[base, nextseqnum)`之中的所有数据包重发；
4. 当收到下层网络层收到的`ACK`包，且`ACK`包校验和正确时：
    1. 将`base`设置为`ACK`包中的序列编号的下一位；
    2. 如果`base == nextseqnum`，即没有待发送的数据包，关闭计时器；否则，计时器重新开始计时；

3.2. 接收方



1. 初始时，期望的序列数据包编号`expectedseqnum`=1；
2. 当收到下层网络层的数据包，且校验和正确时：
    1. 解包获得数据；
    2. 如果数据中的`seqnum`等于`expectedseqnum`：
        1. 将数据发给上层应用层；
        2. 将`expectedseqnum`打包成数据包发给下层网络层；
        3. `expectedseqnum`增1；
    3. 否则直接丢弃数据包；

4. 实验过程

4.1. 编写代码

4.1.1.发送方

定义一个数据包类`Packet`，包含数据和`seqnum`。

```
class Packet:
    def __init__(self, data:str, seq_num:int) -> None:
        self.data = data
        self.seq_num = seq_num
```

调用该类的初始化方法就是在模拟数据打包过程`make_pkt()`。本实验忽略校验和的模拟。

定义上层应用层类`ApplicationLayer`，用于提供数据。

```
class ApplicationLayer:
    def __init__(self, data_len:int=5000) -> None:
        self.data_len = data_len
        self.data_to_send = ["data{:0>4d}".format(i) for i in range(data_len)]
```

定义下层网络层类`NetworkLayer`，模拟网络层的不可靠传输（但随机丢包在GBN类中模拟实现）。

```
class NetworkLayer:
    def __init__(self, host:str, port:int) -> None:
```

```
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.bind((host, port))
        self.socket.listen(1)
        print("等待下层的不可靠传输连接。")
        self.client_socket, address = self.socket.accept()
        print("下层的不可靠传输连接成功。")
        self.client_socket.setblocking(False)

    def udt_send(self, data:str):
        self.client_socket.send(data.encode())

    def udt_rcv(self):
        try:
            return self.client_socket.recv(4).decode("utf-8")
        except BlockingIOError:
            return None

    def close(self):
        self.client_socket.close()
        self.socket.close()
```

使用`socket`实现发送方和接收方的连接。

我规定了接收方返回的`ACK`包只有4字节，发送方每次接受4字节，否则容易发生粘包问题。

定义GBN发送方类`Sender`实现GBN算法。

- 初始化

```
def __init__(
        self,
        window_size:int,
        max_seq_num:int,
        timeout_ms:2000,
        networkLayer:NetworkLayer,
    ) -> None:
    self.window_size = window_size
    self.max_seq_num = max_seq_num
    self.packet_list:list[Packet] = [None] * (self.max_seq_num + 1)
    self.base_num = 1
    self.next_seq_num = 1
    self.timeout_ms = timeout_ms
    self.networkLayer = networkLayer
    self.timer = None
```

- 上层调用

```
def rdt_send(self, data:str) -> bool:
    if self.next_seq_num > max_seq_num:
        return False
    if self.next_seq_num >= self.base_num + self.window_size:
        return False

    self.packet_list[self.next_seq_num] = Packet(data,
self.next_seq_num)
```

```
        self.udt_send(
            self.packet_list[self.next_seq_num].data,
            self.packet_list[self.next_seq_num].seq_num
        )
        if self.base_num == self.next_seq_num:
            self.timer = time.time()
        self.next_seq_num += 1
        return True
```

- 收到下层的包

```
def rdt_rcv(self, ack_index:int):
    print(f"收到ACK={ack_index}, ", end="")
    if (ack_index < self.base_num):
        print(f"(ACK={ack_index}) < (base={self.base_num}), ACK失效丢弃。")
        return
    self.base_num = ack_index + 1
    self.timer = time.time()
    if self.base_num == self.next_seq_num:
        print(f"将base_num设置为下一个序列编号。")
        self.timer = None
    else:
        print(f"将base_num设置为{self.packet_list[self.base_num].seq_num}。")
```

- 向下层发送数据

```
def udt_send(self, data:str, index:int):
    index_data = '{:0>3d} '.format(index) + data
    print(f"发送data=\"{index_data}\"", end="")
    if random.random() > 0.25:
        self.networkLayer.udt_send(index_data)
    else:
        print(", 此包丢失。", end="")
    print()
```

- 定时器

```
def is_timeout(self) -> bool:
    if self.timer is None:
        return False
    return time.time() - self.timer >= 0.001 * self.timeout_ms
```

- 回退N步

```python
def gbn(self):
    self.timer = time.time()
    seq_index = self.base_num
    while seq_index < self.next_seq_num:
        self.udt_send(
            self.packet_list[seq_index].data,
            self.packet_list[seq_index].seq_num
        )
        seq_index += 1
```

- 显示回退N步的包

```python
def show_gbn(self) -> list[int]:
    show = []
    seq_index = self.base_num
    while seq_index < self.next_seq_num:
        show.append(self.packet_list[seq_index].seq_num)
        seq_index += 1
    return show
```

- 获取`ACK`包的序列号

```python
def get_ack_num(self, ack_str:str) -> int:
    return int(ack_str)
```

编写`main`逻辑。

```python
if __name__ == "__main__":
    max_seq_num = 20
    networkLayer = NetworkLayer(host="0.0.0.0", port=23666)
    applicationLayer = ApplicationLayer(max_seq_num)
    sender = Sender(
        window_size=4,
        max_seq_num=max_seq_num,
        timeout_ms=2000,
        networkLayer=networkLayer,
    )
    input("按回车键开始传输：")

    pkg_list = applicationLayer.data_to_send
    index = 1
    while index <= max_seq_num:
        time.sleep(1)
        data = pkg_list[index - 1]
        status = sender.rdt_send(data)
        if status:
            index += 1

        ack_str = networkLayer.udt_rcv()
        if ack_str is not None:
            ack_num = sender.get_ack_num(ack_str)
            sender.rdt_rcv(ack_num)

        if sender.is_timeout():
```

```
            print(f"超时。重传{sender.show_gbn()}")
            sender.gbn()

    while sender.base_num < sender.next_seq_num:
        time.sleep(1)
        ack_str = networkLayer.udt_rcv()
        if ack_str:
            ack_num = sender.get_ack_num(ack_str)
            if ack_num is not None:
                sender.rdt_rcv(ack_num)

        if sender.is_timeout():
            print(f"超时。重传{sender.show_gbn()}")
            sender.gbn()

    print("序列传输完成。")
    networkLayer.close()
```

首先初始化应用层、网络层和GBN对象。设置最大序列长度为`max_seq_num`=20。

在每个时钟周期（定义为1秒）内，执行：

1. 从应用层的数组中获取一份数据，调用`sender`对象的`rdt_send()`方法发送。
2. 查看网络层是否收到`ACK`，如果有，调用`sender`对象的`rdt_rcv()`方法处理。
3. 调用`sender`对象的`is_timeout()`方法判断是否超时，如果超时，开始重传。

当应用层数据取完后，还会存在部分数据未传输完成，则继续处理上述执行循环的2和3步骤，直至所有数据传输完成。

完整代码见附录。

4.1.2.接收方

定义上层应用层类`ApplicationLayer`，用于交付数据。

```
class ApplicationLayer:
    def __init__(self) -> None:
        self.data = list()
```

定义下层网络层类`NetworkLayer`，模拟网络层的不可靠传输（但随机丢包在GBN类中模拟实现）。

```
class NetworkLayer:
    def __init__(self, host:str, port:int) -> None:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((host, port))
        print("下层的不可靠传输连接成功，等待发送方传输。")

    def udt_send(self, data:str):
        self.socket.send(data.encode())

    def udt_rcv(self) -> str:
        message = self.socket.recv(12).decode("utf-8")
        return message

```

```
    def close(self):
        self.socket.close()
```

使用`socket`实现发送方和接收方的连接。

我规定了发送方发送的数据包只有12字节，发送方每次接受12字节，否则容易发生粘包问题。

定义GBN接收方类`Receiver`实现GBN算法。

- 初始化

```
def __init__(self, networkLayer:NetworkLayer,
applicationLayer:ApplicationLayer):
    self.expected_seq_num = 1
    self.networkLayer = networkLayer
    self.applicationLayer = applicationLayer
```

- 交付下层收到的数据包

```
def deliver_data(self, data, seq_num):
    if seq_num == self.expected_seq_num:
        print(f'成功收到seq_num={seq_num}, data=\"{data}\"的包。')
        self.applicationLayer.data.append(data)
        self.udt_send(seq_num)
        self.expected_seq_num += 1
    else:
        print(f'收到seq_num={seq_num}, 与预期seq_num=
{self.expected_seq_num}不符。')
        self.udt_send(self.expected_seq_num - 1)
```

- 向下层发送`ACK`包

```
def udt_send(self, ack_num):
    print(f"发送ACK={ack_num}", end="")
    if random.random() > 0.25:
        self.networkLayer.socket.send("{:0>4d}".format(ack_num).encode())
    else:
        print(", 此包丢失。", end="")
    print()
```

- 获取收到的数据包中的数据

```
def extract(self, message:str):
    seq_num = int(message[:3])
    data = message[4:]
    return seq_num, data
```

编写`main`逻辑。

```
if __name__ == "__main__":
    max_seq_num = 20
    networkLayer = NetworkLayer(host="192.168.31.197", port=23666)
    applicationLayer = ApplicationLayer()
    receiver = Receiver(networkLayer, applicationLayer)
```

```
    while True:
        message = networkLayer.udt_rcv()
        if message:
            seq_num, data = receiver.extract(message)
            receiver.deliver_data(data, seq_num)
            if receiver.expected_seq_num > max_seq_num:
                break
    print("序列传输完成。")
    networkLayer.close()
```

首先初始化应用层、网络层和GBN对象。设置最大序列长度为`max_seq_num`=20。

在每个循环内，执行：

1. 查看网络层是否收到数据包，如果有，调用`receiver`对象的`extract()`方法解包；
2. 调用`receiver`对象的`deliver_data()`方法交付。
3. 如果序列已传输完成，跳出循环。

完整代码见附录。

4.2. 运行实验

首先需要在发送方机器上开放指定的`23666`端口。

```
sudo ufw allow 23666
sudo ufw reload
```

接下来先开启发送方。

```
$ python Sender.py
等待下层的不可靠传输连接。
```

然后开启接收方。

```
$ python Receiver.py
下层的不可靠传输连接成功，等待发送方传输。
```

此时发送方显示：

```
下层的不可靠传输连接成功。
按回车键开始传输：
```

按下回车后，开始传输。传输过程发送方打印内容如下：

```
发送data="001 data0000"
发送data="002 data0001"
收到ACK=1，将base_num设置为2。
发送data="003 data0002"，此包丢失。
发送data="004 data0003"
超时。重传[2, 3, 4]
发送data="002 data0001"
发送data="003 data0002"，此包丢失。
发送data="004 data0003"，此包丢失。
```

```
发送data="005 data0004"
收到ACK=2，将base_num设置为3。
发送data="006 data0005"
超时。重传[3, 4, 5, 6]
发送data="003 data0002"
发送data="004 data0003"
发送data="005 data0004"，此包丢失。
发送data="006 data0005"
收到ACK=3，将base_num设置为4。
发送data="007 data0006"
收到ACK=4，将base_num设置为5。
发送data="008 data0007"
超时。重传[5, 6, 7, 8]
发送data="005 data0004"
发送data="006 data0005"，此包丢失。
发送data="007 data0006"
发送data="008 data0007"
收到ACK=5，将base_num设置为6。
发送data="009 data0008"
收到ACK=5，(ACK=5) < (base=6)，ACK失效丢弃。
收到ACK=5，(ACK=5) < (base=6)，ACK失效丢弃。
超时。重传[6, 7, 8, 9]
发送data="006 data0005"
发送data="007 data0006"，此包丢失。
发送data="008 data0007"，此包丢失。
发送data="009 data0008"
收到ACK=6，将base_num设置为7。
发送data="010 data0009"
收到ACK=6，(ACK=6) < (base=7)，ACK失效丢弃。
超时。重传[7, 8, 9, 10]
发送data="007 data0006"
发送data="008 data0007"
发送data="009 data0008"，此包丢失。
发送data="010 data0009"
收到ACK=7，将base_num设置为8。
发送data="011 data0010"
收到ACK=8，将base_num设置为9。
发送data="012 data0011"，此包丢失。
超时。重传[9, 10, 11, 12]
发送data="009 data0008"
发送data="010 data0009"
发送data="011 data0010"
发送data="012 data0011"
收到ACK=9，将base_num设置为10。
发送data="013 data0012"，此包丢失。
收到ACK=10，将base_num设置为11。
发送data="014 data0013"
收到ACK=11，将base_num设置为12。
发送data="015 data0014"
收到ACK=12，将base_num设置为13。
发送data="016 data0015"，此包丢失。
收到ACK=12，(ACK=12) < (base=13)，ACK失效丢弃。
收到ACK=12，(ACK=12) < (base=13)，ACK失效丢弃。
超时。重传[13, 14, 15, 16]
发送data="013 data0012"
发送data="014 data0013"，此包丢失。
```

```
发送data="015 data0014"
发送data="016 data0015"，此包丢失。
收到ACK=13，将base_num设置为14。
发送data="017 data0016"
收到ACK=13，(ACK=13) < (base=14)，ACK失效丢弃。
收到ACK=13，(ACK=13) < (base=14)，ACK失效丢弃。
超时。重传[14, 15, 16, 17]
发送data="014 data0013"，此包丢失。
发送data="015 data0014"
发送data="016 data0015"，此包丢失。
发送data="017 data0016"，此包丢失。
超时。重传[14, 15, 16, 17]
发送data="014 data0013"
发送data="015 data0014"
发送data="016 data0015"，此包丢失。
发送data="017 data0016"，此包丢失。
收到ACK=14，将base_num设置为15。
发送data="018 data0017"，此包丢失。
收到ACK=15，将base_num设置为16。
发送data="019 data0018"
收到ACK=15，(ACK=15) < (base=16)，ACK失效丢弃。
超时。重传[16, 17, 18, 19]
发送data="016 data0015"
发送data="017 data0016"
发送data="018 data0017"，此包丢失。
发送data="019 data0018"
收到ACK=16，将base_num设置为17。
发送data="020 data0019"
收到ACK=17，将base_num设置为18。
收到ACK=17，(ACK=17) < (base=18)，ACK失效丢弃。
超时。重传[18, 19, 20]
发送data="018 data0017"
发送data="019 data0018"
发送data="020 data0019"
收到ACK=18，将base_num设置为19。
收到ACK=20，将base_num设置为下一个序列编号。
序列传输完成。
```

传输过程接收方打印内容如下：

```
成功收到seq_num=1, data="data0000"的包。
发送ACK=1
成功收到seq_num=2, data="data0001"的包。
发送ACK=2，此包丢失。
收到seq_num=4, 与预期seq_num=3不符。
发送ACK=2
收到seq_num=2, 与预期seq_num=3不符。
发送ACK=2，此包丢失。
收到seq_num=5, 与预期seq_num=3不符。
发送ACK=2，此包丢失。
收到seq_num=6, 与预期seq_num=3不符。
发送ACK=2，此包丢失。
成功收到seq_num=3, data="data0002"的包。
发送ACK=3
成功收到seq_num=4, data="data0003"的包。
```

```
发送ACK=4，此包丢失。
收到seq_num=6，与预期seq_num=5不符。
发送ACK=4
收到seq_num=7，与预期seq_num=5不符。
发送ACK=4，此包丢失。
收到seq_num=8，与预期seq_num=5不符。
发送ACK=4，此包丢失。
成功收到seq_num=5，data="data0004"的包。
发送ACK=5
收到seq_num=7，与预期seq_num=6不符。
发送ACK=5
收到seq_num=8，与预期seq_num=6不符。
发送ACK=5
收到seq_num=9，与预期seq_num=6不符。
发送ACK=5，此包丢失。
成功收到seq_num=6，data="data0005"的包。
发送ACK=6
收到seq_num=9，与预期seq_num=7不符。
发送ACK=6，此包丢失。
收到seq_num=10，与预期seq_num=7不符。
发送ACK=6
成功收到seq_num=7，data="data0006"的包。
发送ACK=7
成功收到seq_num=8，data="data0007"的包。
发送ACK=8，此包丢失。
收到seq_num=10，与预期seq_num=9不符。
发送ACK=8
收到seq_num=11，与预期seq_num=9不符。
发送ACK=8，此包丢失。
成功收到seq_num=9，data="data0008"的包。
发送ACK=9
成功收到seq_num=10，data="data0009"的包。
发送ACK=10
成功收到seq_num=11，data="data0010"的包。
发送ACK=11
成功收到seq_num=12，data="data0011"的包。
发送ACK=12
收到seq_num=14，与预期seq_num=13不符。
发送ACK=12
收到seq_num=15，与预期seq_num=13不符。
发送ACK=12
成功收到seq_num=13，data="data0012"的包。
发送ACK=13
收到seq_num=15，与预期seq_num=14不符。
发送ACK=13
收到seq_num=17，与预期seq_num=14不符。
发送ACK=13
收到seq_num=15，与预期seq_num=14不符。
发送ACK=13，此包丢失。
成功收到seq_num=14，data="data0013"的包。
发送ACK=14
成功收到seq_num=15，data="data0014"的包。
发送ACK=15
收到seq_num=19，与预期seq_num=16不符。
发送ACK=15
成功收到seq_num=16，data="data0015"的包。
```

```
发送ACK=16
成功收到seq_num=17, data="data0016"的包。
发送ACK=17
收到seq_num=19, 与预期seq_num=18不符。
发送ACK=17，此包丢失。
收到seq_num=20, 与预期seq_num=18不符。
发送ACK=17
成功收到seq_num=18, data="data0017"的包。
发送ACK=18
成功收到seq_num=19, data="data0018"的包。
发送ACK=19，此包丢失。
成功收到seq_num=20, data="data0019"的包。
发送ACK=20
序列传输完成。
```

5. 遇到问题及解决方案

5.1. 重传窗口大小问题

我将重传窗口设置为`[base, nextseqnum)`，其大小不一定等于N。当上层调用没有提供大于N和数据包给GBN时，GBN重传只需要重传在窗口中的数据包，显然这些数据包个数不一定是N。

但是与助教的讨论中，受到助教的质疑，认为每次重传的数据包个数一定为N。

解决方案：

与老师讨论后，老师认为重传窗口设置为`[base, nextseqnum)`正确，这也符合GBN的FSM中的描述。

5.2. 数据粘包问题

一开始我将`client_socket.recv()`的参数设置为1024，结果出现了数据粘包问题，程序无法解析收到的数据包和`ACK`，出错率较大。

解决方案：

将数据包和`ACK`包的长度固定，每次从buffer中读入固定长度的数据。

6. 总结和感想

在本次计算机网络编程实验中，我深入学习了GBN协议的实现原理和应用。通过编写具体的发送方和接收方代码，我不仅加深了对于窗口滑动协议的理解，也实际体验了网络编程的挑战和魅力。

本实验的主要目的是实现基于TCP/IP协议栈中传输层的GBN协议，以确保在不可靠的传输环境中数据能够可靠地传输。通过模拟网络层的发送和接收功能，重点学习了如何处理数据包的序列化和确认机制，以及如何管理窗口大小以防止数据丢失和错误。

实验过程中，我首先定义了发送方和接收方的数据结构和基本逻辑。此外，通过设置超时重传机制来应对丢包问题，确保了数据传输的完整性。通过本次实验，我成功实现了一个简单的基于GBN协议的数据传输模型，包括数据的发送、接收和错误处理。实验不仅验证了理论知识的实际应用，也增强了我解决实际问题的能力。

这次实验极大地提升了我对网络编程的兴趣和理解。通过亲自设计和实现复杂的网络传输协议，我更加深刻地理解了计算机网络中的数据流和控制机制。实验不仅让我掌握了网络编程的技巧，也激发了我进一步探索更高级网络技术的热情。

7. 附录

`Sender.py`：

```python
import time
import socket
import random

class Package:
    def __init__(self, data:str, seq_num:int) -> None:
        self.data = data
        self.seq_num = seq_num

class ApplicationLayer:
    def __init__(self, data_len:int=5000) -> None:
        self.data_len = data_len
        self.data_to_send = ["data{:0>4d}".format(i) for i in
range(data_len)]

class NetworkLayer:
    def __init__(self, host:str, port:int) -> None:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.bind((host, port))
        self.socket.listen(1)
        print("等待下层的不可靠传输连接。")
        self.client_socket, address = self.socket.accept()
        print("下层的不可靠传输连接成功。")
        self.client_socket.setblocking(False)

    def udt_send(self, data:str):
        self.client_socket.send(data.encode())

    def udt_rcv(self):
        try:
            return self.client_socket.recv(4).decode("utf-8")
        except BlockingIOError:
            return None

    def close(self):
        self.client_socket.close()
        self.socket.close()

class Sender:
    def __init__(
            self,
            window_size:int,
            max_seq_num:int,
            timeout_ms:2000,
            networkLayer:NetworkLayer,
```

```
        ) -> None:
        self.window_size = window_size
        self.max_seq_num = max_seq_num
        self.package_list:list[Package] = [None] * (self.max_seq_num + 1)
        self.base_num = 1
        self.next_seq_num = 1
        self.timeout_ms = timeout_ms
        self.networkLayer = networkLayer
        self.timer = None

    def rdt_send(self, data:str) -> bool:
        if self.next_seq_num > max_seq_num:
            return False
        if self.next_seq_num >= self.base_num + self.window_size:
            return False

        self.package_list[self.next_seq_num] = Package(data,
self.next_seq_num)
        self.udt_send(
            self.package_list[self.next_seq_num].data,
            self.package_list[self.next_seq_num].seq_num
        )
        if self.base_num == self.next_seq_num:
            self.timer = time.time()
        self.next_seq_num += 1
        return True

    def rdt_rcv(self, ack_index:int):
        print(f"收到ACK={ack_index}, ", end="")
        if (ack_index < self.base_num):
            print(f"(ACK={ack_index}) < (base={self.base_num}), ACK失效丢
弃。")
            return
        self.base_num = ack_index + 1
        self.timer = time.time()
        if self.base_num == self.next_seq_num:
            print(f"将base_num设置为下一个序列编号。")
            self.timer = None
        else:
            print(f"将base_num设置为
{self.package_list[self.base_num].seq_num}。")

    def udt_send(self, data:str, index:int):
        index_data = '{:0>3d} '.format(index) + data
        print(f"发送data=\"{index_data}\"", end="")
        if random.random() > 0.25:
            self.networkLayer.udt_send(index_data)
        else:
            print(", 此包丢失。", end="")
        print()

    def is_timeout(self) -> bool:
        if self.timer is None:
            return False
        return time.time() - self.timer >= 0.001 * self.timeout_ms

```

```python
    def gbn(self):
        self.timer = time.time()
        seq_index = self.base_num
        while seq_index < self.next_seq_num:
            self.udt_send(
                self.package_list[seq_index].data,
                self.package_list[seq_index].seq_num
            )
            seq_index += 1

    def show_gbn(self) -> list[int]:
        show = []
        seq_index = self.base_num
        while seq_index < self.next_seq_num:
            show.append(self.package_list[seq_index].seq_num)
            seq_index += 1
        return show

    def get_ack_num(self, ack_str:str) -> int:
        return int(ack_str)


if __name__ == "__main__":
    max_seq_num = 20
    networkLayer = NetworkLayer(host="0.0.0.0", port=23666)
    applicationLayer = ApplicationLayer(max_seq_num)
    sender = Sender(
        window_size=4,
        max_seq_num=max_seq_num,
        timeout_ms=2000,
        networkLayer=networkLayer,
    )
    input("按回车键开始传输：")

    pkg_list = applicationLayer.data_to_send
    index = 1
    while index <= max_seq_num:
        time.sleep(1)
        data = pkg_list[index - 1]
        status = sender.rdt_send(data)
        if status:
            index += 1

        ack_str = networkLayer.udt_rcv()
        if ack_str is not None:
            ack_num = sender.get_ack_num(ack_str)
            sender.rdt_rcv(ack_num)

        if sender.is_timeout():
            print(f"超时。重传{sender.show_gbn()}")
            sender.gbn()

    while sender.base_num < sender.next_seq_num:
        time.sleep(1)
        ack_str = networkLayer.udt_rcv()
        if ack_str:
```

```
            ack_num = sender.get_ack_num(ack_str)
            if ack_num is not None:
                sender.rdt_rcv(ack_num)

        if sender.is_timeout():
            print(f"超时。重传{sender.show_gbn()}")
            sender.gbn()

    print("序列传输完成。")
    networkLayer.close()
```

`Receiver.py`:

```
import socket
import random

class NetworkLayer:
    def __init__(self, host:str, port:int) -> None:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((host, port))
        print("下层的不可靠传输连接成功，等待发送方传输。")

    def udt_send(self, data:str):
        self.socket.send(data.encode())

    def udt_rcv(self) -> str:
        message = self.socket.recv(12).decode("utf-8")
        return message

    def close(self):
        self.socket.close()


class ApplicationLayer:
    def __init__(self) -> None:
        self.data = list()


class Receiver:
    def __init__(self, networkLayer:NetworkLayer,
applicationLayer:ApplicationLayer):
        self.expected_seq_num = 1
        self.networkLayer = networkLayer
        self.applicationLayer = applicationLayer

    def deliver_data(self, data, seq_num):
        if seq_num == self.expected_seq_num:
            print(f'成功收到seq_num={seq_num}, data=\"{data}\"的包。')
            self.applicationLayer.data.append(data)
            self.udt_send(seq_num)
            self.expected_seq_num += 1
        else:
            print(f'收到seq_num={seq_num}, 与预期seq_num=
{self.expected_seq_num}不符。')
            self.udt_send(self.expected_seq_num - 1)
```

```
41
42      def udt_send(self, ack_num):
43          print(f"发送ACK={ack_num}", end="")
44          if random.random() > 0.25:
45              self.networkLayer.socket.send("
    {:0>4d}".format(ack_num).encode())
46          else:
47              print("，此包丢失。", end="")
48          print()
49
50      def extract(self, message:str):
51          seq_num = int(message[:3])
52          data = message[4:]
53          return seq_num, data
54
55  if __name__ == "__main__":
56      max_seq_num = 20
57      networkLayer = NetworkLayer(host="192.168.31.197", port=23666)
58      applicationLayer = ApplicationLayer()
59      receiver = Receiver(networkLayer, applicationLayer)
60      while True:
61          message = networkLayer.udt_rcv()
62          if message:
63              seq_num, data = receiver.extract(message)
64              receiver.deliver_data(data, seq_num)
65              if receiver.expected_seq_num > max_seq_num:
66                  break
67      print("序列传输完成。")
68      networkLayer.close()
```