

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

目录

1. 实验目的

2. 实验环境

3. 实验原理

3.1. 发送方

3.2. 接收方

4. 实验过程

4.1. 编写代码

4.1.1. 发送方

4.1.2. 接收方

4.2. 运行实验

5. 遇到问题及解决方案

5.1. 重传窗口大小问题

5.2. 数据粘包问题

6. 总结和感想

7. 附录

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。

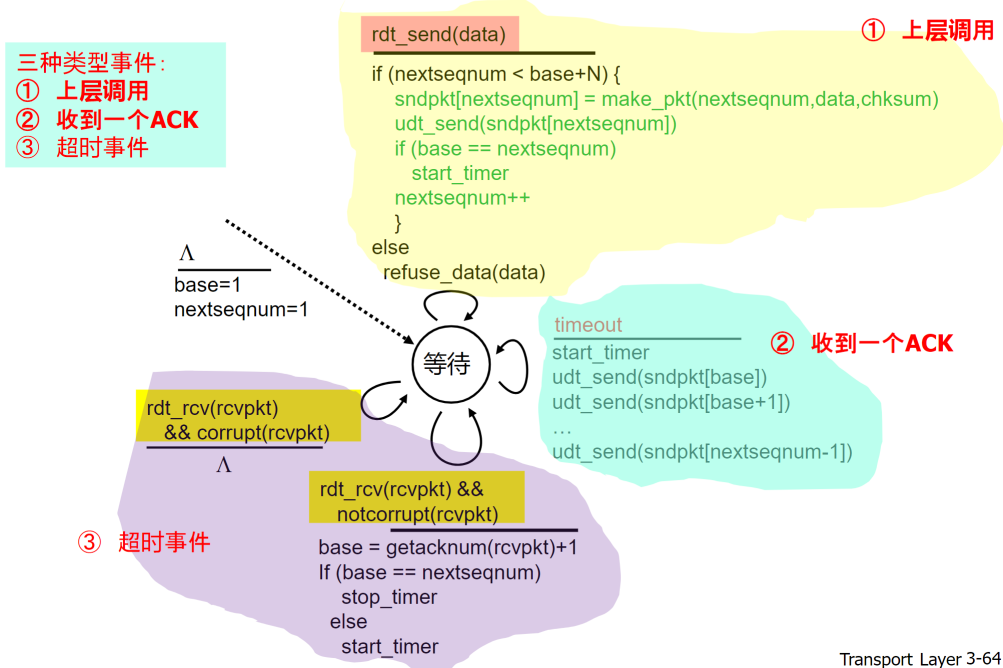
2. 实验环境

- 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

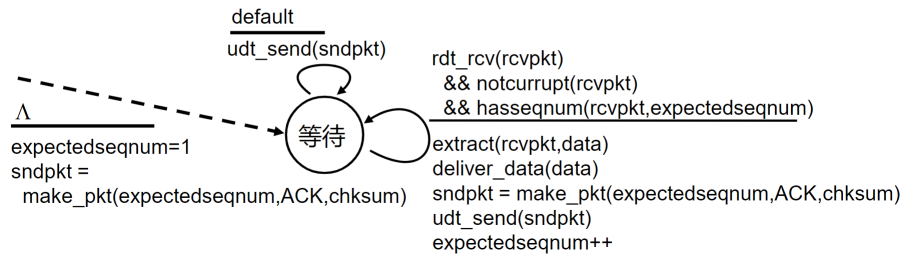
GBN发送方的扩展FSM



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

GBN接收方的扩展FSM



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`。

```
1 class Packet:
2     def __init__(self, data:str, seq_num:int) -> None:
3         self.data = data
4         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。

```
1 class ApplicationLayer:
2     def __init__(self, data_len:int=5000) -> None:
3         self.data_len = data_len
4         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）。

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

```

3     self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4     self.socket.bind((host, port))
5     self.socket.listen(1)
6     print("等待下层的不可靠传输连接。")
7     self.client_socket, address = self.socket.accept()
8     print("下层的不可靠传输连接成功。")
9     self.client_socket.setblocking(False)
10
11     def udt_send(self, data:str):
12         self.client_socket.send(data.encode())
13
14     def udt_rcv(self):
15         try:
16             return self.client_socket.recv(4).decode("utf-8")
17         except BlockingIOError:
18             return None
19
20     def close(self):
21         self.client_socket.close()
22         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

```

1     def __init__(
2         self,
3         window_size:int,
4         max_seq_num:int,
5         timeout_ms:2000,
6         networkLayer:NetworkLayer,
7     ) -> None:
8         self.window_size = window_size
9         self.max_seq_num = max_seq_num
10        self.packet_list:list[Packet] = [None] * (self.max_seq_num + 1)
11        self.base_num = 1
12        self.next_seq_num = 1
13        self.timeout_ms = timeout_ms
14        self.networkLayer = networkLayer
15        self.timer = None

```

- 上层调用

```

1     def rdt_send(self, data:str) -> bool:
2         if self.next_seq_num > max_seq_num:
3             return False
4         if self.next_seq_num >= self.base_num + self.window_size:
5             return False
6
7         self.packet_list[self.next_seq_num] = Packet(data,
self.next_seq_num)

```

```

8     self.udt_send(
9         self.packet_list[self.next_seq_num].data,
10        self.packet_list[self.next_seq_num].seq_num
11    )
12    if self.base_num == self.next_seq_num:
13        self.timer = time.time()
14    self.next_seq_num += 1
15    return True

```

- 收到下层的包

```

1  def rdt_rcv(self, ack_index:int):
2      print(f"收到ACK={ack_index}, ", end="")
3      if (ack_index < self.base_num):
4          print(f"(ACK={ack_index}) < (base={self.base_num}), ACK失效丢
5          弃。")
6          return
7      self.base_num = ack_index + 1
8      self.timer = time.time()
9      if self.base_num == self.next_seq_num:
10         print(f"将base_num设置为下一个序列编号。")
11         self.timer = None
12     else:
13         print(f"将base_num设置为
14         {self.packet_list[self.base_num].seq_num}。")

```

- 向下层发送数据

```

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

```

- 定时器

```

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

```

- 回退N步

```

1 def gbn(self):
2     self.timer = time.time()
3     seq_index = self.base_num
4     while seq_index < self.next_seq_num:
5         self.udt_send(
6             self.packet_list[seq_index].data,
7             self.packet_list[seq_index].seq_num
8         )
9         seq_index += 1

```

- 显示回退N步的包

```

1 def show_gbn(self) -> list[int]:
2     show = []
3     seq_index = self.base_num
4     while seq_index < self.next_seq_num:
5         show.append(self.packet_list[seq_index].seq_num)
6         seq_index += 1
7     return show

```

- 获取ACK包的序列号

```

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

```

编写 main 逻辑。

```

1 if __name__ == "__main__":
2     max_seq_num = 20
3     networkLayer = NetworkLayer(host="0.0.0.0", port=23666)
4     applicationLayer = ApplicationLayer(max_seq_num)
5     sender = Sender(
6         window_size=4,
7         max_seq_num=max_seq_num,
8         timeout_ms=2000,
9         networkLayer=networkLayer,
10    )
11    input("按回车键开始传输: ")
12
13    pkg_list = applicationLayer.data_to_send
14    index = 1
15    while index <= max_seq_num:
16        time.sleep(1)
17        data = pkg_list[index - 1]
18        status = sender.rdt_send(data)
19        if status:
20            index += 1
21
22        ack_str = networkLayer.udt_rcv()
23        if ack_str is not None:
24            ack_num = sender.get_ack_num(ack_str)
25            sender.rdt_rcv(ack_num)
26
27        if sender.is_timeout():

```

```

28         print(f"超时。重传{sender.show_gbn()}")
29         sender.gbn()
30
31     while sender.base_num < sender.next_seq_num:
32         time.sleep(1)
33         ack_str = networkLayer.udt_rcv()
34         if ack_str:
35             ack_num = sender.get_ack_num(ack_str)
36             if ack_num is not None:
37                 sender.rdt_rcv(ack_num)
38
39         if sender.is_timeout():
40             print(f"超时。重传{sender.show_gbn()}")
41             sender.gbn()
42
43     print("序列传输完成。")
44     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。

```

1 class ApplicationLayer:
2     def __init__(self) -> None:
3         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）。

```

1 class NetworkLayer:
2     def __init__(self, host:str, port:int) -> None:
3         self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4         self.socket.connect((host, port))
5         print("下层的不可靠传输连接成功，等待发送方传输。")
6
7     def udt_send(self, data:str):
8         self.socket.send(data.encode())
9
10    def udt_rcv(self) -> str:
11        message = self.socket.recv(12).decode("utf-8")
12        return message
13

```

```

14     def close(self):
15         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

```

1  def __init__(self, networkLayer:NetworkLayer,
2      applicationLayer:ApplicationLayer):
3      self.expected_seq_num = 1
4      self.networkLayer = networkLayer
5      self.applicationLayer = applicationLayer

```

- 交付下层收到的数据包

```

1  def deliver_data(self, data, seq_num):
2      if seq_num == self.expected_seq_num:
3          print(f'成功收到seq_num={seq_num}, data="{data}"的包。')
4          self.applicationLayer.data.append(data)
5          self.udt_send(seq_num)
6          self.expected_seq_num += 1
7      else:
8          print(f'收到seq_num={seq_num}, 与预期seq_num=
9      {self.expected_seq_num}不符。')
10         self.udt_send(self.expected_seq_num - 1)

```

- 向下层发送 ACK 包

```

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

```

- 获取收到的数据包中的数据

```

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

```

编写 `main` 逻辑。

```

1  if __name__ == "__main__":
2      max_seq_num = 20
3      networkLayer = NetworkLayer(host="192.168.31.197", port=23666)
4      applicationLayer = ApplicationLayer()
5      receiver = Receiver(networkLayer, applicationLayer)

```



```

6     while True:
7         message = networkLayer.udt_rcv()
8         if message:
9             seq_num, data = receiver.extract(message)
10            receiver.deliver_data(data, seq_num)
11            if receiver.expected_seq_num > max_seq_num:
12                break
13            print("序列传输完成。")
14            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。

```

1 | sudo ufw allow 23666
2 | sudo ufw reload

```

接下来先开启发送方。

```

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

```

然后开启接收方。

```

1 | $ python Receiver.py
2 | 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：

```

1 | 下层的不可靠传输连接成功。
2 | 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：

```

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

```

10 发送data="005 data0004"
11 收到ACK=2, 将base_num设置为3。
12 发送data="006 data0005"
13 超时。重传[3, 4, 5, 6]
14 发送data="003 data0002"
15 发送data="004 data0003"
16 发送data="005 data0004", 此包丢失。
17 发送data="006 data0005"
18 收到ACK=3, 将base_num设置为4。
19 发送data="007 data0006"
20 收到ACK=4, 将base_num设置为5。
21 发送data="008 data0007"
22 超时。重传[5, 6, 7, 8]
23 发送data="005 data0004"
24 发送data="006 data0005", 此包丢失。
25 发送data="007 data0006"
26 发送data="008 data0007"
27 收到ACK=5, 将base_num设置为6。
28 发送data="009 data0008"
29 收到ACK=5, (ACK=5) < (base=6), ACK失效丢弃。
30 收到ACK=5, (ACK=5) < (base=6), ACK失效丢弃。
31 超时。重传[6, 7, 8, 9]
32 发送data="006 data0005"
33 发送data="007 data0006", 此包丢失。
34 发送data="008 data0007", 此包丢失。
35 发送data="009 data0008"
36 收到ACK=6, 将base_num设置为7。
37 发送data="010 data0009"
38 收到ACK=6, (ACK=6) < (base=7), ACK失效丢弃。
39 超时。重传[7, 8, 9, 10]
40 发送data="007 data0006"
41 发送data="008 data0007"
42 发送data="009 data0008", 此包丢失。
43 发送data="010 data0009"
44 收到ACK=7, 将base_num设置为8。
45 发送data="011 data0010"
46 收到ACK=8, 将base_num设置为9。
47 发送data="012 data0011", 此包丢失。
48 超时。重传[9, 10, 11, 12]
49 发送data="009 data0008"
50 发送data="010 data0009"
51 发送data="011 data0010"
52 发送data="012 data0011"
53 收到ACK=9, 将base_num设置为10。
54 发送data="013 data0012", 此包丢失。
55 收到ACK=10, 将base_num设置为11。
56 发送data="014 data0013"
57 收到ACK=11, 将base_num设置为12。
58 发送data="015 data0014"
59 收到ACK=12, 将base_num设置为13。
60 发送data="016 data0015", 此包丢失。
61 收到ACK=12, (ACK=12) < (base=13), ACK失效丢弃。
62 收到ACK=12, (ACK=12) < (base=13), ACK失效丢弃。
63 超时。重传[13, 14, 15, 16]
64 发送data="013 data0012"
65 发送data="014 data0013", 此包丢失。

```

66 发送data="015 data0014"
67 发送data="016 data0015", 此包丢失。
68 收到ACK=13, 将base_num设置为14。
69 发送data="017 data0016"
70 收到ACK=13, (ACK=13) < (base=14), ACK失效丢弃。
71 收到ACK=13, (ACK=13) < (base=14), ACK失效丢弃。
72 超时。重传[14, 15, 16, 17]
73 发送data="014 data0013", 此包丢失。
74 发送data="015 data0014"
75 发送data="016 data0015", 此包丢失。
76 发送data="017 data0016", 此包丢失。
77 超时。重传[14, 15, 16, 17]
78 发送data="014 data0013"
79 发送data="015 data0014"
80 发送data="016 data0015", 此包丢失。
81 发送data="017 data0016", 此包丢失。
82 收到ACK=14, 将base_num设置为15。
83 发送data="018 data0017", 此包丢失。
84 收到ACK=15, 将base_num设置为16。
85 发送data="019 data0018"
86 收到ACK=15, (ACK=15) < (base=16), ACK失效丢弃。
87 超时。重传[16, 17, 18, 19]
88 发送data="016 data0015"
89 发送data="017 data0016"
90 发送data="018 data0017", 此包丢失。
91 发送data="019 data0018"
92 收到ACK=16, 将base_num设置为17。
93 发送data="020 data0019"
94 收到ACK=17, 将base_num设置为18。
95 收到ACK=17, (ACK=17) < (base=18), ACK失效丢弃。
96 超时。重传[18, 19, 20]
97 发送data="018 data0017"
98 发送data="019 data0018"
99 发送data="020 data0019"
100 收到ACK=18, 将base_num设置为19。
101 收到ACK=20, 将base_num设置为下一个序列编号。
102 序列传输完成。

```

传输过程接收方打印内容如下：

```

1 成功收到seq_num=1, data="data0000"的包。
2 发送ACK=1
3 成功收到seq_num=2, data="data0001"的包。
4 发送ACK=2, 此包丢失。
5 收到seq_num=4, 与预期seq_num=3不符。
6 发送ACK=2
7 收到seq_num=2, 与预期seq_num=3不符。
8 发送ACK=2, 此包丢失。
9 收到seq_num=5, 与预期seq_num=3不符。
10 发送ACK=2, 此包丢失。
11 收到seq_num=6, 与预期seq_num=3不符。
12 发送ACK=2, 此包丢失。
13 成功收到seq_num=3, data="data0002"的包。
14 发送ACK=3
15 成功收到seq_num=4, data="data0003"的包。

```

16 发送ACK=4, 此包丢失。
17 收到seq_num=6, 与预期seq_num=5不符。
18 发送ACK=4
19 收到seq_num=7, 与预期seq_num=5不符。
20 发送ACK=4, 此包丢失。
21 收到seq_num=8, 与预期seq_num=5不符。
22 发送ACK=4, 此包丢失。
23 成功收到seq_num=5, data="data0004"的包。
24 发送ACK=5
25 收到seq_num=7, 与预期seq_num=6不符。
26 发送ACK=5
27 收到seq_num=8, 与预期seq_num=6不符。
28 发送ACK=5
29 收到seq_num=9, 与预期seq_num=6不符。
30 发送ACK=5, 此包丢失。
31 成功收到seq_num=6, data="data0005"的包。
32 发送ACK=6
33 收到seq_num=9, 与预期seq_num=7不符。
34 发送ACK=6, 此包丢失。
35 收到seq_num=10, 与预期seq_num=7不符。
36 发送ACK=6
37 成功收到seq_num=7, data="data0006"的包。
38 发送ACK=7
39 成功收到seq_num=8, data="data0007"的包。
40 发送ACK=8, 此包丢失。
41 收到seq_num=10, 与预期seq_num=9不符。
42 发送ACK=8
43 收到seq_num=11, 与预期seq_num=9不符。
44 发送ACK=8, 此包丢失。
45 成功收到seq_num=9, data="data0008"的包。
46 发送ACK=9
47 成功收到seq_num=10, data="data0009"的包。
48 发送ACK=10
49 成功收到seq_num=11, data="data0010"的包。
50 发送ACK=11
51 成功收到seq_num=12, data="data0011"的包。
52 发送ACK=12
53 收到seq_num=14, 与预期seq_num=13不符。
54 发送ACK=12
55 收到seq_num=15, 与预期seq_num=13不符。
56 发送ACK=12
57 成功收到seq_num=13, data="data0012"的包。
58 发送ACK=13
59 收到seq_num=15, 与预期seq_num=14不符。
60 发送ACK=13
61 收到seq_num=17, 与预期seq_num=14不符。
62 发送ACK=13
63 收到seq_num=15, 与预期seq_num=14不符。
64 发送ACK=13, 此包丢失。
65 成功收到seq_num=14, data="data0013"的包。
66 发送ACK=14
67 成功收到seq_num=15, data="data0014"的包。
68 发送ACK=15
69 收到seq_num=19, 与预期seq_num=16不符。
70 发送ACK=15
71 成功收到seq_num=16, data="data0015"的包。

```
72  发送ACK=16
73  成功收到seq_num=17, data="data0016"的包。
74  发送ACK=17
75  收到seq_num=19, 与预期seq_num=18不符。
76  发送ACK=17, 此包丢失。
77  收到seq_num=20, 与预期seq_num=18不符。
78  发送ACK=17
79  成功收到seq_num=18, data="data0017"的包。
80  发送ACK=18
81  成功收到seq_num=19, data="data0018"的包。
82  发送ACK=19, 此包丢失。
83  成功收到seq_num=20, data="data0019"的包。
84  发送ACK=20
85  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，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:

```
1  import time
2  import socket
3  import random
4
5  class Package:
6      def __init__(self, data:str, seq_num:int) -> None:
7          self.data = data
8          self.seq_num = seq_num
9
10 class ApplicationLayer:
11     def __init__(self, data_len:int=5000) -> None:
12         self.data_len = data_len
13         self.data_to_send = ["data{:0>4d}".format(i) for i in
14                               range(data_len)]
15
16 class NetworkLayer:
17     def __init__(self, host:str, port:int) -> None:
18         self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19         self.socket.bind((host, port))
20         self.socket.listen(1)
21         print("等待下层的不可靠传输连接。")
22         self.client_socket, address = self.socket.accept()
23         print("下层的不可靠传输连接成功。")
24         self.client_socket.setblocking(False)
25
26     def udt_send(self, data:str):
27         self.client_socket.send(data.encode())
28
29     def udt_rcv(self):
30         try:
31             return self.client_socket.recv(4).decode("utf-8")
32         except BlockingIOError:
33             return None
34
35     def close(self):
36         self.client_socket.close()
37         self.socket.close()
38
39 class Sender:
40     def __init__(
41         self,
42         window_size:int,
43         max_seq_num:int,
44         timeout_ms:2000,
45         networkLayer:NetworkLayer,
```

```

45     ) -> None:
46         self.window_size = window_size
47         self.max_seq_num = max_seq_num
48         self.package_list:list[Package] = [None] * (self.max_seq_num + 1)
49         self.base_num = 1
50         self.next_seq_num = 1
51         self.timeout_ms = timeout_ms
52         self.networkLayer = networkLayer
53         self.timer = None
54
55     def rdt_send(self, data:str) -> bool:
56         if self.next_seq_num > max_seq_num:
57             return False
58         if self.next_seq_num >= self.base_num + self.window_size:
59             return False
60
61         self.package_list[self.next_seq_num] = Package(data,
self.next_seq_num)
62         self.udt_send(
63             self.package_list[self.next_seq_num].data,
64             self.package_list[self.next_seq_num].seq_num
65         )
66         if self.base_num == self.next_seq_num:
67             self.timer = time.time()
68         self.next_seq_num += 1
69         return True
70
71     def rdt_rcv(self, ack_index:int):
72         print(f"收到ACK={ack_index}, ", end="")
73         if (ack_index < self.base_num):
74             print(f"(ACK={ack_index}) < (base={self.base_num}), ACK失效丢
弃。")
75             return
76         self.base_num = ack_index + 1
77         self.timer = time.time()
78         if self.base_num == self.next_seq_num:
79             print(f"将base_num设置为下一个序列编号。")
80             self.timer = None
81         else:
82             print(f"将base_num设置为
{self.package_list[self.base_num].seq_num}。")
83
84     def udt_send(self, data:str, index:int):
85         index_data = '{:0>3d} '.format(index) + data
86         print(f"发送data=\"{index_data}\"", end="")
87         if random.random() > 0.25:
88             self.networkLayer.udt_send(index_data)
89         else:
90             print(", 此包丢失。", end="")
91         print()
92
93     def is_timeout(self) -> bool:
94         if self.timer is None:
95             return False
96         return time.time() - self.timer >= 0.001 * self.timeout_ms
97

```

```

98     def gbn(self):
99         self.timer = time.time()
100        seq_index = self.base_num
101        while seq_index < self.next_seq_num:
102            self.udt_send(
103                self.package_list[seq_index].data,
104                self.package_list[seq_index].seq_num
105            )
106            seq_index += 1
107
108        def show_gbn(self) -> list[int]:
109            show = []
110            seq_index = self.base_num
111            while seq_index < self.next_seq_num:
112                show.append(self.package_list[seq_index].seq_num)
113                seq_index += 1
114            return show
115
116        def get_ack_num(self, ack_str:str) -> int:
117            return int(ack_str)
118
119
120    if __name__ == "__main__":
121        max_seq_num = 20
122        networkLayer = NetworkLayer(host="0.0.0.0", port=23666)
123        applicationLayer = ApplicationLayer(max_seq_num)
124        sender = Sender(
125            window_size=4,
126            max_seq_num=max_seq_num,
127            timeout_ms=2000,
128            networkLayer=networkLayer,
129        )
130        input("按回车键开始传输: ")
131
132        pkg_list = applicationLayer.data_to_send
133        index = 1
134        while index <= max_seq_num:
135            time.sleep(1)
136            data = pkg_list[index - 1]
137            status = sender.rdt_send(data)
138            if status:
139                index += 1
140
141            ack_str = networkLayer.udt_rcv()
142            if ack_str is not None:
143                ack_num = sender.get_ack_num(ack_str)
144                sender.rdt_rcv(ack_num)
145
146            if sender.is_timeout():
147                print(f"超时。重传{sender.show_gbn()}")
148                sender.gbn()
149
150        while sender.base_num < sender.next_seq_num:
151            time.sleep(1)
152            ack_str = networkLayer.udt_rcv()
153            if ack_str:

```



```

154         ack_num = sender.get_ack_num(ack_str)
155         if ack_num is not None:
156             sender.rdt_rcv(ack_num)
157
158         if sender.is_timeout():
159             print(f"超时。重传{sender.show_gbn()}")
160             sender.gbn()
161
162     print("序列传输完成。")
163     networkLayer.close()

```

Receiver.py:

```

1  import socket
2  import random
3
4  class NetworkLayer:
5      def __init__(self, host:str, port:int) -> None:
6          self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7          self.socket.connect((host, port))
8          print("下层的不可靠传输连接成功，等待发送方传输。")
9
10     def udt_send(self, data:str):
11         self.socket.send(data.encode())
12
13     def udt_rcv(self) -> str:
14         message = self.socket.recv(12).decode("utf-8")
15         return message
16
17     def close(self):
18         self.socket.close()
19
20
21 class ApplicationLayer:
22     def __init__(self) -> None:
23         self.data = list()
24
25
26 class Receiver:
27     def __init__(self, networkLayer:NetworkLayer,
28 applicationLayer:ApplicationLayer):
29         self.expected_seq_num = 1
30         self.networkLayer = networkLayer
31         self.applicationLayer = applicationLayer
32
33     def deliver_data(self, data, seq_num):
34         if seq_num == self.expected_seq_num:
35             print(f'成功收到seq_num={seq_num}, data="{data}"的包。')
36             self.applicationLayer.data.append(data)
37             self.udt_send(seq_num)
38             self.expected_seq_num += 1
39         else:
40             print(f'收到seq_num={seq_num}, 与预期seq_num=
{self.expected_seq_num}不符。')
41             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()

```