

计算机网络实验报告

Lab3-2 基于UDP服务设计可靠传输协议并编程实现

网络空间安全学院 物联网工程

2110951 梁晓储

代码已发布到github: https://github.com/WangshuXC/Computer_network

一、实验要求

1. 实现单向数据传输（一端发数据，一端返回确认）。
2. 对于每个任务要求给出详细的协议设计。
3. 完成给定测试文件的传输，显示传输时间和平均吞吐率。
4. 性能测试指标：吞吐率、延时，给出图形结果并进行分析。
5. 完成详细的实验报告（每个任务完成一份，主要包含自己的协议设计、实现方法、遇到的问题、实验结果，不要抄写太多的背景知识）。
6. 编写的程序应该结构清晰，具有较好的可读性。
7. 提交程序源码、可执行文件和实验报告。
8. 在实验3-1的基础上，将停等机制改成基于滑动窗口的流量控制机制，发送窗口和接收窗口采用相同大小，支持累积确认，完成给定测试文件的传输。

二、协议设计和实验流程

Header协议设计

在send.cpp和receive.cpp中定义一个结构体 `HEADER`，其中包含如下信息。

```

1  struct HEADER
2  {
3      u_short sum = 0;
4      u_short datasize = 0;
5      unsigned char flag = 0;
6      unsigned char SEQ = 0;
7      HEADER() {
8          sum = 0;
9          datasize = 0;
10         flag = 0;
11         SEQ = 0;
12     }
13 };

```

每次发送packet需要修改header中的信息时修改该全局数组，再将其加入sendBuf。

- `sum`:16位的校验和
- `datasize`:所包含数据长度 16位
- `flag`:8位，使用后三位，排列是 FIN ACK SYN
- `SEQ`:8位，传输的序列号，0~255，超过后 mod

flag:

```

1  const unsigned char SYN = 0x1;
2  // 001— FIN = 0 ACK = 0 SYN = 1
3
4  const unsigned char ACK = 0x2;
5  // 010— FIN = 0 ACK = 1 SYN = 0
6
7  const unsigned char ACK_SYN = 0x3;
8  // 011— FIN = 0 ACK = 1 SYN = 1
9
10 const unsigned char FIN = 0x4;

```

```

11 // 100— FIN = 1 ACK = 0 SYN = 0
12
13 const unsigned char FIN_ACK = 0x5;
14 // 101— FIN = 1 ACK = 0 SYN = 1
15
16 const unsigned char OVER = 0x7;
17 // 结束标志 111— FIN = 1 ACK = 1 SYN = 1

```

GBN协议说明

首先建立窗口，以windows为10为例，定义head为窗口头，tail为窗口尾，当收到的seq大于现有head的值（不可能大于tail的值）且没有seq的溢出更新情况时，把head移动到收到的ack包的seq处，通过以下上方红色框中的代码实现窗口的滑动（如果收到的seq比head大很多则直接滑动到seq处）；如果收到的seq为因为溢出更新过的seq，则需要+256再进行处理

```

1  if (int(header.SEQ) ≥ head % 256)
2  {
3      //收到的ACK序列号等于当前已确认的最大序列号，表示这是一条重复的确认信息，需要进行计数处理
4      if (int(header.SEQ) == head % 256)
5      {
6          count++;
7      }
8      else
9      {
10         count = 0;
11     }
12
13     head = head + int(header.SEQ) - head % 256;
14
15     cout << "[\033[1;32mReceive\033[0m] 收到了ACK:  Flag:" <<
16     int(header.flag)
17         << " SEQ:" << int(header.SEQ) << endl;

```

```

18     cout << "[\033[1;33mInfo\033[0m] head:" << head % 256 <<
    ' ' << "tail:" << tail % 256;
19     cout << " count:" << count << endl;
20     if (count == 2)
21     {
22         tail = head + 1;
23         cout << "[\033[1;33mInfo\033[0m] 出现丢包, 将窗口内未确
    认的包重传" << endl;
24     }
25     else if (count > 2)
26     {
27         tail = head + 1;
28     }
29 }
30 //如果接收到的ACK序列号小于窗口大小, 但跨越了序列号环, 需要更新缓冲区头
    部
31 else if (head % 256 > 256 - WINDOWS - 1 && int(header.SEQ)
    < WINDOWS)
32 {
33     head = head + 256 - head % 256 + int(header.SEQ);
34     cout << "[\033[1;32mReceive\033[0m] 收到了ACK (跨seq) :
    Flag:" << int(header.flag)
35         << " SEQ:" << int(header.SEQ) << endl;
36 }

```

代码中通过两条代码进行窗口滑动

```

1     head = head + int(header.SEQ) - head % 256;
2
3     head = head + 256 - head % 256 + int(header.SEQ);

```

如果出现错误, 如校验和出错或者丢包或者超时, 则 `tail = head + 1`, 通过此条命令清空除了head之后的窗口中未确认的数据包, 将从head往后的所有数据包重新打包通过while循环依次发送, 实现“回退N步”

实验流程

1. 三次握手建立连接：

- 发送端发送第一次握手（SYN）。
- 接收端接收第一次握手，并发送第二次握手（ACK）。
- 发送端接收第二次握手，并发送第三次握手（ACK_SYN）。

2. 文件数据传输：

- 发送端发送文件数据，每个数据包都有一个 **HEADER** 头部，包含校验和、数据长度、标志位等信息。
- 接收端接收文件数据，对每个数据包进行校验和验证，如果校验和正确，发送确认 ACK 给发送端，表示成功接收数据。

3. 四次挥手断开连接：

- 接收端发送第一次挥手（FIN）。
- 发送端接收第一次挥手，并发送第二次挥手（ACK）。
- 接收端接收第二次挥手，并发送第三次挥手（ACK）。
- 发送端接收第三次挥手，并发送第四次挥手（FIN_ACK）。

三、功能实现和代码分析

差错检测实现

差错校验是通过计算校验和来实现的。具体来说，校验和是在每个数据包的 **HEADER** 结构中计算得出的一个值，用于检测数据在传输过程中是否发生了错误或丢失。

在发送数据包之前，通过 **send_package** 函数将数据按照指定长度和顺序号组织成 **HEADER** 结构。然后，在计算校验和之前，先将校验和字段置为0。接下来，对 **HEADER** 结构中的所有成员变量（包括数据、数据长度、标志位和序列号）进行逐位异或（XOR）运算，最终得到校验和的值。将这个计算得到的校验和写入 **HEADER** 结构的校验和字段中。

当接收端收到数据包时，它会重新计算接收到的数据包的校验和。如果计算得到的校验和与接收到的数据包中的校验和相等，说明数据在传输过程中没有发生错误或丢失。如果两者不相等，则表示数据包可能存在差错，需要进行处理。

```
1  u_short checksum(u_short* mes, int size) {
2      int count = (size + 1) / 2;
3      u_short* buf = (u_short*)malloc(size + 1);
4      memset(buf, 0, size + 1);
5      memcpy(buf, mes, size);
6
7      u_long sum = 0;
8      while (count--) {
9          sum += *buf++;
10         if (sum & 0xffff0000) {
11             sum &= 0xffff;
12             sum++;
13         }
14     }
15     return ~(sum & 0xffff);
16 }
```

丢包加速处理

1. 丢包模拟

```
1  // 丢包测试
2  int drop_probability = rand() % 100;
3  if (drop_probability < LOSS) {
4      cout << "[\033[1;34mLoss\033[0m] 模拟丢包" << endl;
5      continue;
6  }
```

2. 接收端丢失来自发送端的数据包

以如下情况为例：

发送端以收到2号ack，接收端此时收到3号数据包，但是由于丢包程序，自动将这个包丢弃，则接收端无法处理3号数据包，也就无法返回3号的ack；与此同时，发送端一直在发送数据包，将4号数据包发送给接收端，接收端收到4号数据包，但此时接收端希望收到3号数据包，所以无法返回正确的ack，由于不是希望的seq，则返回2号的ack（已经确认的最大序列号为2号），以后一直返回2号的ack

发送端一直收到2号ack，无法滑动窗口，等待窗口满后，无法收到任何ack，导致超时，在超时时执行 `tail = head + 1`，进行回退，之后将head及之后的包重新发送。

可以实现丢包后的重新发送，但是需要等待窗口满后才能开始计算超时的起始时间，所以相当于窗口越大，一旦丢包，被发现所需要的时间就越长。对此想到了两种解决方法，可以将超时的开始时间定为发送完head之后的时间，以此计算超时，第二种方法是对收到的重复ack进行计数，一旦收到超过3次一样的ack，则判定发生丢包，直接回退重传，代码中实现了第二种：

设置count变量，通过返回ack的seq判定返回的包是否与上一次的相同，count表示再次收到相同重复的包的次数，如果count>=2，则直接回退：

```
1  if (int(header.SEQ) ≥ head % 256)
2  {
3      if (int(header.SEQ) == head % 256)
4      {
5          count++;
6      }
7      else
8      {
9          count = 0;
10     }
11
12     head = head + int(header.SEQ) - head % 256;
```

```

13
14     cout << "[\033[1;32mReceive\033[0m] 收到了ACK:  Flag:" <<
int(header.flag)
15         << " SEQ:" << int(header.SEQ) << endl;
16
17     cout << "[\033[1;33mInfo\033[0m] head:" << head % 256 <<
' ' << "tail:" << tail % 256;
18     cout << " count:" << count << endl;
19     if (count == 2)
20     {
21         tail = head + 1;
22         cout << "[\033[1;33mInfo\033[0m] 出现丢包, 将窗口内未确
认的包重传" << endl;
23     }
24     else if (count > 2)
25     {
26         tail = head + 1;
27     }
28 }

```

3. 发送端丢失来自接收端的ack

以如下情况为例：

发送端已经收到2号ack，此时发送3号数据包接收端返回的3号ack丢失，由于窗口未满，所以继续发送端继续发送4号及以后的数据包，倘若接收端已经确认了3号ack，即发出过3号ack，那么此时接收端在等待4号ack，如果接收端接收到了4号ack，确认后则可以返回给发送端4号ack，发送端虽然没有收到3号ack，但是收到了4号ack，由于此次实验采用累计确认，收到4号ack即代表4号及4号之前的数据包都已经成功确认、成功接收，所以即使3号ack丢失也没关系，不会对数据传输造成影响。

三次握手和四次挥手

基于3-1中的实验代码，未作过多的改进

文件传输

在连接（握手）完毕后，会进入到文件传输的过程。

发送端

首先发送端在首部宏定义中添加窗口大小变量,同时也可在后续通过输入进行修改

```
1  int WINDOWS = 10;
2
3  cout << endl << "输入希望的滑动窗口大小" << endl;
4  cin >> WINDOWS;
5  cout << "当前滑动窗口大小为 " << WINDOWS << endl;
```

发送端有关文件传输的函数是send()和send_package(), 大致步骤如下:

1. 发送方将要传输的数据分割成若干个大小为MAXSIZE的数据包，并为每个数据包添加了一个头部HEADER。其中，MAXSIZE为数据包的最大长度，HEADER结构体存储了数据包的一些信息，例如包的大小、序列号、校验和等。

```

1  HEADER header;
2  char *buffer = new char[MAXSIZE + sizeof(header)];
3  header.datasize = len;
4  header.SEQ = unsigned char(order); // 序列号
5  memcpy(buffer, &header, sizeof(header));
6  memcpy(buffer + sizeof(header), message, sizeof(header) +
   len);
7  u_short check = cksum((u_short *)buffer, sizeof(header) +
   len); // 计算校验和：头部+数据

```

2. 发送方将数据包按顺序发送给接收方，并维护一个滑动窗口。滑动窗口的大小为WINDOWS，每次向接收方发送数据包时，都会将该数据包放入滑动窗口中，直到滑动窗口满或者所有数据包都已经发送完毕。

```

1  int packagenum = len / MAXSIZE + (len % MAXSIZE != 0);
2  int head = -1; // 缓冲区头部，前方为已经被确认的报文
3  int tail = 0; // 缓冲区尾部
4
5  if (tail - head ≤ WINDOWS && tail ≠ packagenum)
6      {
7          send_package(socketClient, servAddr,
8          servAddrlen, message + tail * MAXSIZE,
9          tail = packagenum - 1 ? len -
10         (packagenum - 1) * MAXSIZE : MAXSIZE, tail % 256);
11
12         start = clock(); // 记录发送时间
13         tail++;
14     }

```

3. 发送方在发送每个数据包时，记录下当前的时间。如果在一个固定时间内（MAX_TIME）没有收到接收方的确认消息，那么就认为这个数据包丢失了，需要重新发送。

```
1  start = clock(); // 记录发送时间
2  tail++;
3
4  if (clock() - start > MAX_TIME)
5  {
6      tail = head + 1;
7      cout << "[\033[1;33mInfo\033[0m] 超时了, tail=head+1";
8  }
```

4. 接收方接收到数据包后，会计算校验和并检查是否正确。如果校验和正确，则将接收到的数据包的序列号与滑动窗口的序列号进行比较。如果接收到的数据包的序列号等于滑动窗口的序列号，说明该数据包已经被接收方成功接收，此时发送方可以将滑动窗口向前滑动一位，继续发送下一个数据包。如果接收到的数据包的序列号小于滑动窗口的序列号，则说明该数据包是之前已经接收过的数据包，此时仅需要发送ACK确认消息即可。如果接收到的数据包的序列号大于滑动窗口的序列号，则说明接收到的数据包是之前还未接收到的数据包，此时需要向发送方发送重复ACK确认消息。

```
1  if (int(header.SEQ) ≥ head % 256)
2  {
3      // 如果是重传就+1
4      if (int(header.SEQ) == head % 256)
5      {
6          count++;
7      }
8      else
9      {
10         count = 0;
11     }
12     head = head + int(header.SEQ) - head % 256;
```

```

13     cout << "[\033[1;32mReceive\033[0m] 收到了ACK: Flag:" <<
    int(header.flag)
14         << " SEQ:" << int(header.SEQ) << endl;
15     cout << "head:" << head << ' ' << "tail:" << tail <<
    endl;
16     cout << "count:" << count << endl;
17     if (count ≥ 2)
18     {
19         tail = head + 1;
20         cout << "[\033[1;33mInfo\033[0m] 出现丢包, 将窗口内未确
    认的包重传" << endl;
21     }
22 }
23 else if (head % 256 > 256 - WINDOWS - 1 && int(header.SEQ)
    < WINDOWS)
24 {
25     head = head + 256 - head % 256 + int(header.SEQ);
26     cout << "[\033[1;32mReceive\033[0m] 收到了ACK (跨seq) :
    Flag:" << int(header.flag)
27         << " SEQ:" << int(header.SEQ) << endl;
28 }

```

5. 如果发送方连续接收到两个相同的ACK确认消息，则说明最近发送的某个数据包丢失了，需要将滑动窗口内所有未被确认的数据包重新发送。

```

1     if (count ≥ 2)
2     {
3         tail = head + 1;
4         cout << "[\033[1;33mInfo\033[0m] 出现丢包, 将窗口内未确认的包
    重传" << endl;
5     }

```

6. 发送方在发送完所有的数据包后，向接收方发送一个OVER消息，表示数据传输已经结束。接收方收到OVER消息后，向发送方发送一个ACK确认消息，表示已经成功接收到了所有数据包。如果发送方没有收到ACK确认消息，则需要重新发送OVER消息。

```
1  header.flag = OVER;
2  header.sum = 0;
3  u_short temp = cksum((u_short *)&header, sizeof(header));
4  header.sum = temp;
5  memcpy(Buffer, &header, sizeof(header));
6  sendto(socketClient, Buffer, sizeof(header), 0, (sockaddr
   *)&servAddr, servAddrLen);
7  cout << "[\033[1;31mSend\033[0m] 发送OVER信号" << endl;
8  start = clock();
9  while (1)
10 {
11     u_long mode = 1;
12     ioctlsocket(socketClient, FIONBIO, &mode);
13     while (recvfrom(socketClient, Buffer, MAXSIZE, 0,
   (sockaddr *)&servAddr, &servAddrLen) ≤ 0)
14     {
15         if (clock() - start > MAX_TIME)
16         {
17             char *Buffer = new char[sizeof(header)];
18             header.flag = OVER;
19             header.sum = 0;
20             u_short temp = cksum((u_short *)&header,
   sizeof(header));
21             header.sum = temp;
22             memcpy(Buffer, &header, sizeof(header));
23             sendto(socketClient, Buffer, sizeof(header), 0,
   (sockaddr *)&servAddr, servAddrLen);
24             cout << "[\033[1;33mInfo\033[0m] OVER消息发送超
   时, 已经重传" << endl;
25             start = clock();
26         }
27     }
```

```

28     memcpy(&header, Buffer, sizeof(header)); // 缓冲区接收到信
    息, 读取
29     u_short check = cksum((u_short *)&header,
    sizeof(header));
30     if (header.flag == OVER)
31     {
32         cout << "[\033[1;33mInfo\033[0m] 对方已成功接收文件" <<
        endl;
33         break;
34     }
35     else
36     {
37         continue;
38     }
39 }

```

接收端

接收端接受文件的函数是RecvMessage(), 大致步骤如下:

1. 首先定义了一个 `fileLength` 变量用于记录接收到的文件长度, 以及一些其他变量和数据结构的定义。

```

1  long int fileLength = 0;
2  HEADER header;
3  char *Buffer = new char[MAXSIZE + sizeof(header)];
4  int seq = 0;
5  int index = 0;

```

2. 进入无限循环, 不断接收来自发送端的数据包, 直到接收到结束标识 `OVER` 的数据包为止。

3. 在每次循环中，首先调用 `recvfrom` 函数接收数据包，并进行一些处理：

- 判断接收到的长度，如果长度小于等于0，则跳出循环。
- 使用 `memcpy` 从接收到的数据包中提取出头部信息 `header`。
- 对数据包进行校验和验证，如果校验和不为0，则丢弃该数据包并等待重传；否则进行后续处理。

```
1  int length = recvfrom(sockServ, Buffer, sizeof(header) +  
    MAXSIZE, 0, (sockaddr *)&ClientAddr, &ClientAddrLen); // 接  
    收报文长度  
2  
3  if (length ≤ 0)  
4      break;  
5  
6  int temp2 = checkSum((u_short *)Buffer, length);  
7  if (temp2 ≠ 0)  
8  {  
9      cout << "[\033[1;33mInfo\033[0m] 数据包出现错误, 收到的  
    length为" << length << "已经丢弃, 等待重传 " << endl;  
10     continue; // 丢弃数据包  
11 }
```

4. 判断接收到的数据包的标识 `flag`：

- 如果是结束标识 `OVER`，则打印传输结束信息并跳出循环。
- 如果是其他标识，继续处理数据包。

```
1  if (header.flag = OVER)  
2  {  
3      cout << "[\033[1;33mInfo\033[0m] 传输结束" << endl;  
4      break;  
5  }
```

5. 检查接收到的数据包的序列号 `SEQ` 和期望的序列号 `seq` 是否一致:

- 如果一致, 表示收到了期望的序列号的数据包, 进行后续处理。
- 如果不一致, 表示出现了问题, 需要返回ACK并重发之前的ACK, 然后丢弃当前数据包并继续等待。

```
1  if (seq != int(header.SEQ))
2  {
3      // 说明出了问题, 返回ACK
4      header.flag = ACK;
5      header.datasize = 0;
6      header.SEQ = (unsigned char)seq - 1; // 假设已经确认2, 希望
      收到3, 但收到4, 所以应该返回2的ACK, 所以seq要减1
7      header.sum = 0;
8      u_short temp = checkSum((u_short *)&header,
      sizeof(header));
9      header.sum = temp;
10     memcpy(Buffer, &header, sizeof(header));
11     // 重发该包的ACK
12     sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr
      *)&ClientAddr, ClientAddrLen);
13     cout << "[\033[1;31mSend\033[0m] 非希望的seq, 重发给发送端
      flag:" << (int)header.flag << " SEQ:" << (int)header.SEQ <<
      " SUM:" << int(header.sum) << endl;
14     continue;
15 }
```

6. 处理接收到的数据包:

- 提取数据包中的内容, 并将内容存入 `message` 中。
- 更新文件长度 `fileLength`。
- 返回ACK给发送端。

```
1  seq = int(header.SEQ);
```



```
2   if (seq > 255)
3   {
4       seq = seq - 256;
5   }
6   // 取出buffer中的内容
7   cout << "[\033[1;32mReceive\033[0m] 收到了 " << length -
      sizeof(header) << " 字节 - Flag:" << int(header.flag)
8       << " SEQ : " << int(header.SEQ) << " SUM:" <<
      int(header.sum) << endl;
9   char *temp = new char[length - sizeof(header)];
10  memcpy(temp, Buffer + sizeof(header), length -
      sizeof(header)); // temp中存入当前数据包内容
11  // cout << "size" << sizeof(message) << endl;
12  memcpy(message + fileLength, temp, length - sizeof(header));
      // 把每一个文件数据包中的内容, 通过all的偏移, 存入message
13  fileLength = fileLength + int(header.datasize);
14
15  // 返回ACK
16  header.flag = ACK;
17  header.datasize = 0;
18  header.SEQ = (unsigned char)seq;
19  header.sum = 0;
20  u_short temp1 = checkSum((u_short *)&header,
      sizeof(header));
21  header.sum = temp1;
22  memcpy(Buffer, &header, sizeof(header));
23  // Sleep(3); //sleep一下再返回ack, 延迟返回ack
24
25  sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr
      *)&ClientAddr, ClientAddrLen);
26  cout << "[\033[1;31mSend\033[0m] 回复发送端 - flag:" <<
      (int)header.flag << " SEQ:" << (int)header.SEQ << " SUM:" <<
      int(header.sum) << endl;
27  seq++;
28  if (seq > 255)
29  {
30      seq = seq - 256;
31  }
```

7. 在循环结束后，发送结束标识 `OVER` 的数据包给发送端。

```
1  header.flag = OVER;
2  header.sum = 0;
3  u_short temp = checksum((u_short *)&header,
    sizeof(header));
4  header.sum = temp;
5  memcpy(Buffer, &header, sizeof(header));
6  if (sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr
    *)&ClientAddr, ClientAddrLen) == -1)
7  {
8      return -1;
9  }
10 return fileLength;
```

计算传输时间和吞吐率

该过程是在发送端中完成的

```
1  send(server, severAddr, len, (char *)(inputFile.c_str()),
    inputFile.length());
2  clock_t start1 = clock();
3  send(server, severAddr, len, buffer, i);
4  clock_t end1 = clock();
5
6  cout << "[\\033[1;36mOut\\033[0m] 传输总时间为:" << (end1 -
    start1) / CLOCKS_PER_SEC << "s" << endl;
7  cout << "[\\033[1;36mOut\\033[0m] 吞吐率为:" << fixed <<
    setprecision(2) << (((double)i) / ((end1 - start1) /
    CLOCKS_PER_SEC)) << "byte/s" << endl;
```

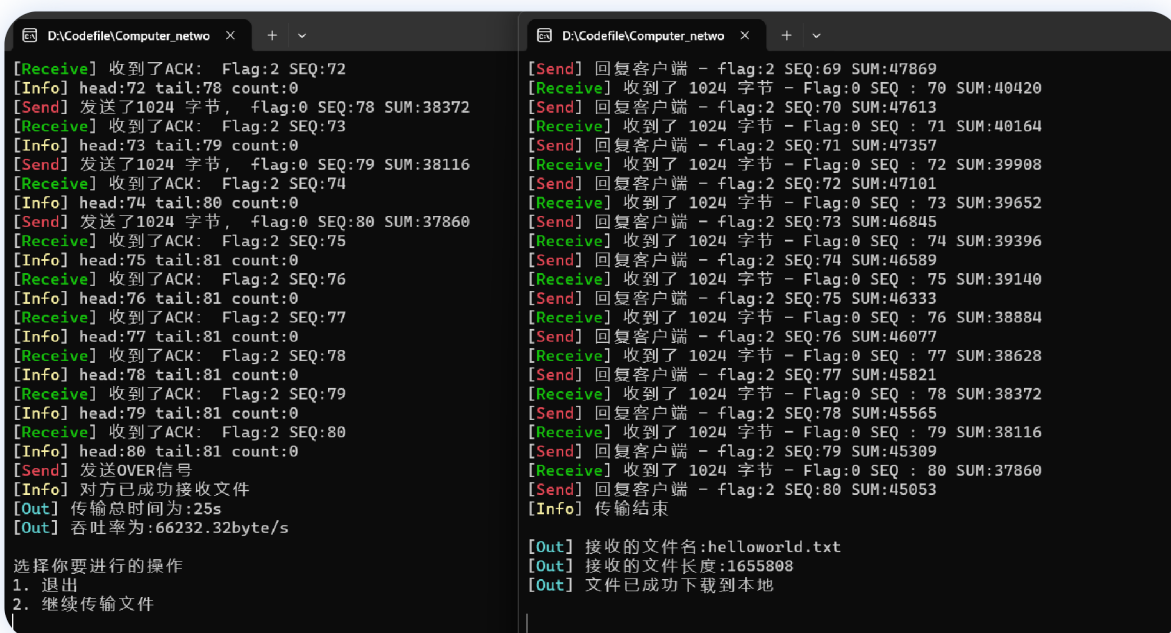
首先分别在发送文件内容前后运用clock函数记录两次时间。

然后用第二个clock函数调用返回的时间减去第一个clock函数调用返回的时间，并除以CLOCKS_PER_SEC（每秒钟的时钟周期数），得到传输时间。

最后，通过将文件大小除以传输时间来得到吞吐量，同时我为了数据的精准，使用了fixed和setprecision函数来设置输出的小数位数。

实验结果展示

发送成功截图



```
D:\Codefile\Computer_netwo x + v
[Receive] 收到了ACK: Flag:2 SEQ:72
[Info] head:72 tail:78 count:0
[Send] 发送了1024 字节, flag:0 SEQ:78 SUM:38372
[Receive] 收到了ACK: Flag:2 SEQ:73
[Info] head:73 tail:79 count:0
[Send] 发送了1024 字节, flag:0 SEQ:79 SUM:38116
[Receive] 收到了ACK: Flag:2 SEQ:74
[Info] head:74 tail:80 count:0
[Send] 发送了1024 字节, flag:0 SEQ:80 SUM:37860
[Receive] 收到了ACK: Flag:2 SEQ:75
[Info] head:75 tail:81 count:0
[Receive] 收到了ACK: Flag:2 SEQ:76
[Info] head:76 tail:81 count:0
[Receive] 收到了ACK: Flag:2 SEQ:77
[Info] head:77 tail:81 count:0
[Receive] 收到了ACK: Flag:2 SEQ:78
[Info] head:78 tail:81 count:0
[Receive] 收到了ACK: Flag:2 SEQ:79
[Info] head:79 tail:81 count:0
[Receive] 收到了ACK: Flag:2 SEQ:80
[Info] head:80 tail:81 count:0
[Send] 发送OVER信号
[Info] 对方已成功接收文件
[Out] 传输总时间为:25s
[Out] 吞吐率为:66232.32byte/s

选择你要进行的操作
1. 退出
2. 继续传输文件

D:\Codefile\Computer_netwo x + v
[Send] 回复客户端 - flag:2 SEQ:69 SUM:47869
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 70 SUM:40420
[Send] 回复客户端 - flag:2 SEQ:70 SUM:47613
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 71 SUM:40164
[Send] 回复客户端 - flag:2 SEQ:71 SUM:47357
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 72 SUM:39908
[Send] 回复客户端 - flag:2 SEQ:72 SUM:47101
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 73 SUM:39652
[Send] 回复客户端 - flag:2 SEQ:73 SUM:46845
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 74 SUM:39396
[Send] 回复客户端 - flag:2 SEQ:74 SUM:46589
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 75 SUM:39140
[Send] 回复客户端 - flag:2 SEQ:75 SUM:46333
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 76 SUM:38884
[Send] 回复客户端 - flag:2 SEQ:76 SUM:46077
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 77 SUM:38628
[Send] 回复客户端 - flag:2 SEQ:77 SUM:45821
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 78 SUM:38372
[Send] 回复客户端 - flag:2 SEQ:78 SUM:45565
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 79 SUM:38116
[Send] 回复客户端 - flag:2 SEQ:79 SUM:45309
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 80 SUM:37860
[Send] 回复客户端 - flag:2 SEQ:80 SUM:45053
[Info] 传输结束

[Out] 接收的文件名:helloworld.txt
[Out] 接收的文件长度:1655808
[Out] 文件已成功下载到本地
```

```
D:\Codefile\Computer_netwo x + v
[Send] 回复客户端 - flag:2 SEQ:10 SUM:62973
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 11 SUM:15251
[Send] 回复客户端 - flag:2 SEQ:11 SUM:62717
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 12 SUM:44031
[Send] 回复客户端 - flag:2 SEQ:12 SUM:62461
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 13 SUM:37579
[Send] 回复客户端 - flag:2 SEQ:13 SUM:62205
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 14 SUM:29023
[Send] 回复客户端 - flag:2 SEQ:14 SUM:61949
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 15 SUM:23827
[Send] 回复客户端 - flag:2 SEQ:15 SUM:61693
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 16 SUM:63904
[Send] 回复客户端 - flag:2 SEQ:16 SUM:61437
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 17 SUM:55420
[Send] 回复客户端 - flag:2 SEQ:17 SUM:61181
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 18 SUM:52551
[Send] 回复客户端 - flag:2 SEQ:18 SUM:60925
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 19 SUM:33471
[Send] 回复客户端 - flag:2 SEQ:19 SUM:60669
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 20 SUM:10120
[Send] 回复客户端 - flag:2 SEQ:20 SUM:60413
[Receive] 收到了 841 字节 - Flag:0 SEQ : 21 SUM:40532
[Send] 回复客户端 - flag:2 SEQ:21 SUM:60157
[Info] 传输结束

[Out] 接收的文件名:1.jpg
[Out] 接收的文件长度:1857353
[Out] 文件已成功下载到本地

D:\Codefile\Computer_netwo x + v
[Receive] 收到了ACK: Flag:2 SEQ:13
[Info] head:13 tail:19 count:0
[Send] 发送了1024 字节, flag:0 SEQ:19 SUM:33471
[Receive] 收到了ACK: Flag:2 SEQ:14
[Info] head:14 tail:20 count:0
[Send] 发送了1024 字节, flag:0 SEQ:20 SUM:10120
[Receive] 收到了ACK: Flag:2 SEQ:15
[Info] head:15 tail:21 count:0
[Send] 发送了841 字节, flag:0 SEQ:21 SUM:40532
[Receive] 收到了ACK: Flag:2 SEQ:16
[Info] head:16 tail:22 count:0
[Receive] 收到了ACK: Flag:2 SEQ:17
[Info] head:17 tail:22 count:0
[Receive] 收到了ACK: Flag:2 SEQ:18
[Info] head:18 tail:22 count:0
[Receive] 收到了ACK: Flag:2 SEQ:19
[Info] head:19 tail:22 count:0
[Receive] 收到了ACK: Flag:2 SEQ:20
[Info] head:20 tail:22 count:0
[Receive] 收到了ACK: Flag:2 SEQ:21
[Info] head:21 tail:22 count:0
[Send] 发送OVER信号
[Info] 对方已成功接收文件
[Out] 传输总时间为:28s
[Out] 吞吐率为:66334.04byte/s

选择你要进行的操作
1. 退出
2. 继续传输文件
```

```
D:\Codefile\Computer_netwo x + v
[Receive] 收到了ACK: Flag:2 SEQ:120
[Info] head:120 tail:126 count:0
[Send] 发送了1024 字节, flag:0 SEQ:126 SUM:57777
[Receive] 收到了ACK: Flag:2 SEQ:121
[Info] head:121 tail:127 count:0
[Send] 发送了1024 字节, flag:0 SEQ:127 SUM:1644
[Receive] 收到了ACK: Flag:2 SEQ:122
[Info] head:122 tail:128 count:0
[Send] 发送了265 字节, flag:0 SEQ:128 SUM:2783
[Receive] 收到了ACK: Flag:2 SEQ:123
[Info] head:123 tail:129 count:0
[Receive] 收到了ACK: Flag:2 SEQ:124
[Info] head:124 tail:129 count:0
[Receive] 收到了ACK: Flag:2 SEQ:125
[Info] head:125 tail:129 count:0
[Receive] 收到了ACK: Flag:2 SEQ:126
[Info] head:126 tail:129 count:0
[Receive] 收到了ACK: Flag:2 SEQ:127
[Info] head:127 tail:129 count:0
[Receive] 收到了ACK: Flag:2 SEQ:128
[Info] head:128 tail:129 count:0
[Send] 发送OVER信号
[Info] 对方已成功接收文件
[Out] 传输总时间为:90s
[Out] 吞吐率为:65538.94byte/s

选择你要进行的操作
1. 退出
2. 继续传输文件

D:\Codefile\Computer_netwo x + v
[Send] 回复客户端 - flag:2 SEQ:117 SUM:35581
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 118 SUM:35468
[Send] 回复客户端 - flag:2 SEQ:118 SUM:35325
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 119 SUM:7293
[Send] 回复客户端 - flag:2 SEQ:119 SUM:35069
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 120 SUM:14612
[Send] 回复客户端 - flag:2 SEQ:120 SUM:34813
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 121 SUM:36019
[Send] 回复客户端 - flag:2 SEQ:121 SUM:34557
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 122 SUM:11354
[Send] 回复客户端 - flag:2 SEQ:122 SUM:34301
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 123 SUM:5474
[Send] 回复客户端 - flag:2 SEQ:123 SUM:34045
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 124 SUM:38097
[Send] 回复客户端 - flag:2 SEQ:124 SUM:33789
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 125 SUM:32134
[Send] 回复客户端 - flag:2 SEQ:125 SUM:33533
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 126 SUM:57777
[Send] 回复客户端 - flag:2 SEQ:126 SUM:33277
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 127 SUM:1644
[Send] 回复客户端 - flag:2 SEQ:127 SUM:33021
[Receive] 收到了 265 字节 - Flag:0 SEQ : 128 SUM:2783
[Send] 回复客户端 - flag:2 SEQ:128 SUM:32765
[Info] 传输结束

[Out] 接收的文件名:2.jpg
[Out] 接收的文件长度:5898505
[Out] 文件已成功下载到本地
```

```
D:\Codefile\Computer_netwo x + v
[Receive] 收到了ACK: Flag:2 SEQ:160
[Info] head:160 tail:166 count:0
[Send] 发送了1024 字节, flag:0 SEQ:166 SUM:29999
[Receive] 收到了ACK: Flag:2 SEQ:161
[Info] head:161 tail:167 count:0
[Send] 发送了1024 字节, flag:0 SEQ:167 SUM:32801
[Receive] 收到了ACK: Flag:2 SEQ:162
[Info] head:162 tail:168 count:0
[Send] 发送了482 字节, flag:0 SEQ:168 SUM:62972
[Receive] 收到了ACK: Flag:2 SEQ:163
[Info] head:163 tail:169 count:0
[Receive] 收到了ACK: Flag:2 SEQ:164
[Info] head:164 tail:169 count:0
[Receive] 收到了ACK: Flag:2 SEQ:165
[Info] head:165 tail:169 count:0
[Receive] 收到了ACK: Flag:2 SEQ:166
[Info] head:166 tail:169 count:0
[Receive] 收到了ACK: Flag:2 SEQ:167
[Info] head:167 tail:169 count:0
[Receive] 收到了ACK: Flag:2 SEQ:168
[Info] head:168 tail:169 count:0
[Send] 发送OVER信号
[Info] 对方已成功接收文件
[Out] 传输总时间为:182s
[Out] 吞吐率为:65763.70byte/s

选择你要进行的操作
1. 退出
2. 继续传输文件

D:\Codefile\Computer_netwo x + v
[Send] 回复客户端 - flag:2 SEQ:157 SUM:25341
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 158 SUM:43181
[Send] 回复客户端 - flag:2 SEQ:158 SUM:25085
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 159 SUM:40583
[Send] 回复客户端 - flag:2 SEQ:159 SUM:24829
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 160 SUM:22970
[Send] 回复客户端 - flag:2 SEQ:160 SUM:24573
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 161 SUM:43121
[Send] 回复客户端 - flag:2 SEQ:161 SUM:24317
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 162 SUM:20139
[Send] 回复客户端 - flag:2 SEQ:162 SUM:24061
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 163 SUM:15395
[Send] 回复客户端 - flag:2 SEQ:163 SUM:23805
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 164 SUM:8994
[Send] 回复客户端 - flag:2 SEQ:164 SUM:23549
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 165 SUM:57771
[Send] 回复客户端 - flag:2 SEQ:165 SUM:23293
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 166 SUM:29999
[Send] 回复客户端 - flag:2 SEQ:166 SUM:23037
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 167 SUM:32801
[Send] 回复客户端 - flag:2 SEQ:167 SUM:22781
[Receive] 收到了 482 字节 - Flag:0 SEQ : 168 SUM:62972
[Send] 回复客户端 - flag:2 SEQ:168 SUM:22525
[Info] 传输结束

[Out] 接收的文件名:3.jpg
[Out] 接收的文件长度:11968994
[Out] 文件已成功下载到本地
```

由于未使用路由，固无路由器设置

三次握手建立连接

```
D:\Codefile\Computer_netwo x + v
输入需要模拟的丢包率(0-100)
1
[Info] 成功建立socket
[Info] 进入监听状态, 等待客户端上线
[Receive] 接收到第一次握手数据
[Send] 成功发送第二次握手数据
[Receive] 接收到第三次握手数据
[Info] 成功连接
[Info] 成功建立连接, 正在等待接收文件

D:\Codefile\Computer_netwo x + v
[Send] 成功发送第一次握手数据
[Receive] 接收到第二次握手数据
[Send] 成功发送第三次握手数据
[Info] 服务器成功连接! 可以发送数据

输入希望的滑动窗口大小
|
```

丢包后实现超时重传

```
[Loss] 模拟丢包  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 发送了1024 字节, flag:0 SEQ:43 SUM:19109  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:44 count:0  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:1  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:45 SUM:12823  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:46 count:2  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] 出现丢包, 将窗口内未确认的包重传  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:3  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:4  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:5  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:6  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:7  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Receive] 收到了ACK: Flag:2 SEQ:43  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Info] head:43 tail:45 count:8  
[Send] 非希望的seq, 重发给客户端 flag:2 SEQ:43 SUM:54525 [Send] 发送了1024 字节, flag:0 SEQ:44 SUM:40357
```

输出传输时间和吞吐率，四次挥手断开连接

The figure consists of two side-by-side screenshots of a network traffic analysis tool. The left screenshot shows a series of 'Send' and 'Receive' messages between a client and a server. The messages are labeled with 'Send' and 'Receive' in red and green respectively. The details for each message include the flag, sequence number, and sum. The right screenshot shows a similar series of messages, but with a 'Loss' message indicating a packet was lost, and a 'Timeout' message indicating the connection was closed.

性能测试指标

对三个文件进行传输测试（未经过路由,未设置丢包,窗口大小设置为6）

文件名	文件大小	传输时间	吞吐率
helloworld.txt	1655808byte	25s	66232.32byte/s
1.jpg	1857353byte	28s	66334.04byte/s
2.jpg	5898505byte	90s	65538.94byte/s
3.jpg	11968994byte	182s	65763.70byte/s

图表结果如下

