北京邮电大学计算机学院

2021-2022 学年第 1 学期实验报告

课程名称： 计算机网络

实验名称： 传输层实验

实验完成人：

姓名： 胡杨 **学号：** 2019212008 **成绩：**

指导教师： 雷友珣

日　　期： 2021 年 12 月 24 日

一、 实验目的

通过本实验理解和掌握 TCP、UDP 协议功能和协议过程，掌握套接字（Socket）编程方法。

二、实验任务

1）TCP 协议过程分析：通过 Linux 网络命令或自行开发的程序，实现 TCP 连接上的数据传输，仿真数据传输过程中的丢包，使用抓包工具抓取 TCP 连接上产生的 TCP 报文，并分析 TCP 协议过程：分析 TCP 连接建立过程、TCP 差错控制过程，分析 TCP 连接三次握手过程中的 TCP 报文的端口号、顺序号、确认号字段的值；分析 TCP 数据报文、确认报文的顺序号、确认号字段的值；根据 IP 分组的长度字段（IP 分组头长度字段、IP 分组总长度字段），分析 IP 分组中所携带的 TCP 报文的长度；根据 TCP 报文中 TCP 报文头字段长度，分析 TCP 报文中携带的数据长度；分析 TCP 协议出错重传机制、重传的 TCP 数据报文的时间间隔。

2）基于 UDP 协议实现提供可靠通信服务的协议软件：完成基于 UDP 协议的可靠通信协议设计并编码实现，在实验报告中描述该协议软件的需求分析、设计、实现、测试。

三、实验内容

1） 下面以自行开发的 TCP 应用程序为例，说明 TCP 协议过程分析部分的实验内容：基于套接字编程接口（Socket API）开发基于 TCP 协议的客户端程序、服务器端程序，实现 TCP 客户端程序向 TCP 服务器端程序传输数据；在数据传输的过程中，使用 iptables 命令，使得 TCP 应用服务器所在主机丢弃接收到的 IP 分组，然后再使用 iptables 命令恢复 TCP 服务器端 IP 分组的接收，使 TCP 服务器不再丢弃客户端发送来的 IP 分组，以此来模拟 TCP 分组的丢失。观察 TCP 分组丢失情况下，TCP 差错控制功能的实现过程。客户端向服务器端发送完数据后，关闭客户端和服务器端的 TCP 连接。判断客户端和服务器端应用程序发送和接收到的数据是否相同。

2） 基于套接字编程接口，开发基于 UDP 协议的可靠通信协议软件，设计基于 UDP 协议实现可靠通信的协议机制和协议消息定义，完成基于 UDP 协议的可靠通信协议软件的设计与实现。

四、实验环境

macOS 12.0.1, Wireshark, Ubuntu 18.0.6, g++1z

五、实验过程描述

1. TCP 协议过程分析

1.1 正常运行原程序

1.1.1 服务器端

```
======waiting for client's request======
The received message is : 这是第一次测试. This is the first test.

The length of the received message is : 47
The received message is : 这是第一次测试. This is the first test.
这是第一次测试. This is the first test.

The length of the received message is : 94
The received message is : 这是第一次测试. This is the first test.
这是第一次测试. This is the first test.
这是第一次测试. This is the first test.
这是第一次测试. This is the first test.
这是第一次测试. This is the first test.

The length of the received message is : 282
The received message is : 这是第一次测试. This is the first test.

The length of the received message is : 47
The client closed the connection.
The total bytes received is : 470
```

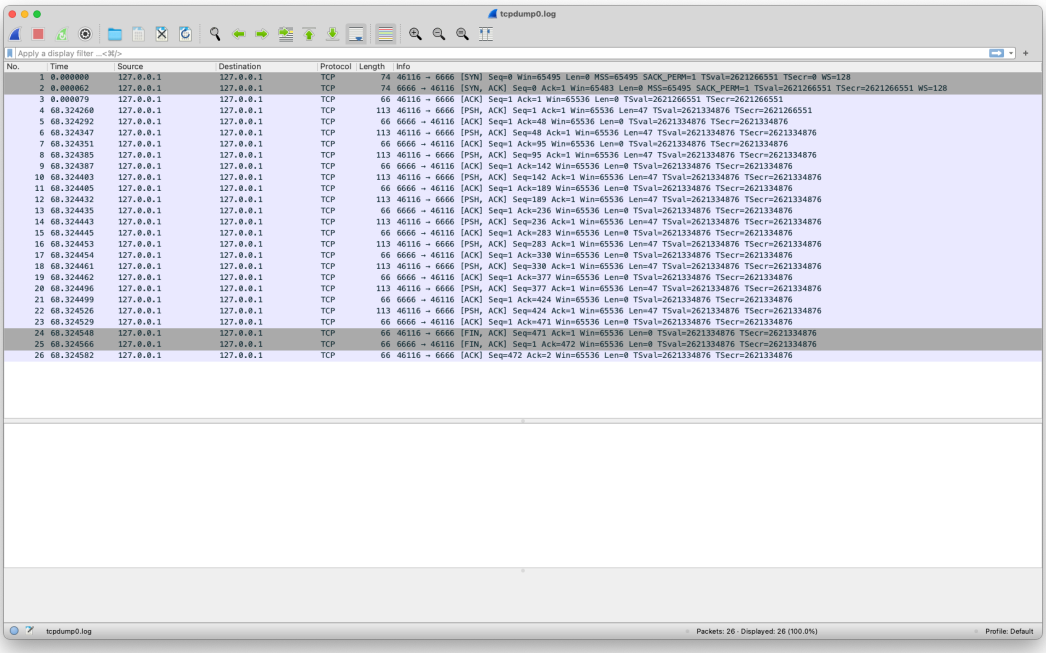
1.1.2 客户端

```
starting to send data to server, please input the data to be sent to server repeatedly:
这是第一次测试. This is the first test.
The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The message sent is:  这是第一次测试. This is the first test.

The total bytes sent to server is : 470
```

1.1.3 抓包

```
root@node1:~# sudo tcpdump -i lo tcp port 6666 -w "tcpdump0.log"
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
^C26 packets captured
52 packets received by filter
0 packets dropped by kernel
```

1.1.4 在 Wireshark 软件中查看抓包结果



1.1.5 抓包结果分析

数据包	内容
1	第一次握手（SYN 消息，序列号 0）
2	第二次握手（SYN 消息，序列号 0，ACK = 0 + 1 = 1）
3	第三次握手（序列号 1，ACK = 0 + 1 = 1）
4	客户端发送第一条消息（序列号 1，长度为 47，ACK = 1 + Len = 1）
5	服务器返回 ACK（1 + 47 = 48）
6	客户端发送第二条消息（序列号 48，长度为 47，ACK = 1 + Len = 1）
7	服务器返回 ACK（48 + 47 = 95）
8	客户端发送第三条消息（序列号 95，长度为 47，ACK = 1 + Len = 1）
9	服务器返回 ACK（95 + 47 = 142）
…	客户端发送第 n 条消息（长度为 47），服务器返回对应的 ACK
22	客户端发送第 10 条消息（序列号 424，长度为 47，ACK = 1 + Len = 1）
23	服务器返回 ACK（424 + 47 = 471）
24	客户端发送 FIN 消息，第一次挥手（序列号 471，ACK = 1 + Len = 1）
25	服务器发送 FIN 消息，并捎带 ACK（471 + 1 = 472），第二、三次挥手
26	客户端返回服务器 FIN 消息的 ACK 消息（ACK = 1 + 1 = 2）

值得注意的是，在三次握手和四次挥手期间，ACK 的计算是 SEQ + 1，与消息长度无关。为了保持简练，在接下来的抓包结果分析中我们不详述 ACK 的计算过程。

1.2 拒收来自 6666 端口的包

1.2.1 服务器端

```
======waiting for client's request======
The received message is : This is the second test.
This is the second test.
This is the second test.
This is the second test.
This is the second test.
This is the second test.
This is the second test.
This is the second test.
This is the second test.
This is the second test.

The length of the received message is : 250
The client closed the connection.
The total bytes received is : 250
```

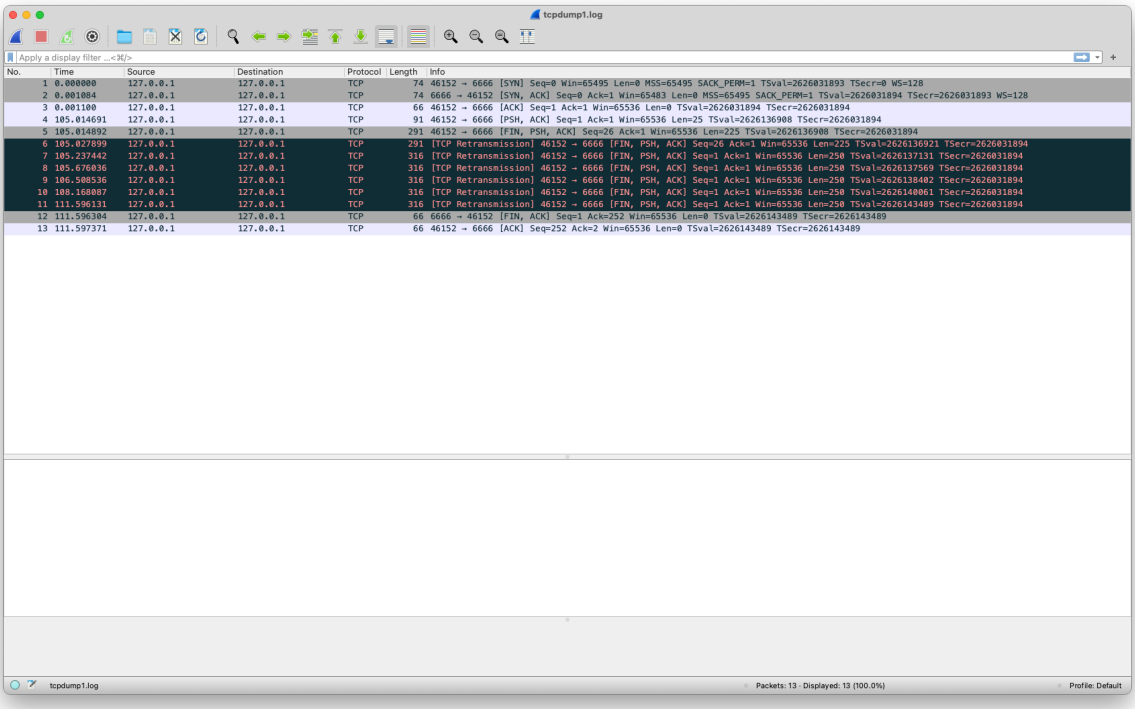
1.2.2 客户端

```
starting to send data to server, please input the data to be sent to server repeatedly:
This is the second test.
The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The message sent is:  This is the second test.

The total bytes sent to server is : 250
```

1.2.3 抓包

```
root@node1:~# sudo tcpdump -i lo tcp port 6666 -w "tcpdump1.log"
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
^C13 packets captured
26 packets received by filter
0 packets dropped by kernel
```

1.2.4 在 Wireshark 软件中查看抓包结果



1.2.5 抓包结果分析

数据包	内容
1	第一次握手
2	第二次握手
3	第三次握手
4	客户端发送第一条消息（序列号 1）
5	客户端发送后九条消息（序列号 26），第一次挥手
6	客户端未收到 ACK，重传后九条消息（序列号 26）
7	客户端未收到 ACK，重传十条消息（序列号 1），与之前间隔 200ms
8	客户端未收到 ACK，重传十条消息（序列号 1），与之前间隔 400ms
9	客户端未收到 ACK，重传十条消息（序列号 1），与之前间隔 800ms
10	客户端未收到 ACK，重传十条消息（序列号 1），与之前间隔 1600ms
11	客户端未收到 ACK，重传十条消息（序列号 1），与之前间隔 3200ms
12	服务器收到十条消息（序列号 1），返回 ACK 和 FIN，第二、三次挥手
13	客户端返回 ACK，第四次挥手，连接终止

1.3 执行修改后的代码

1.3.1 服务器端

```
======waiting for client's request======
The received message is : This is a test message!

The length of the received message is : 24
The received message is : This is a test message!
This is a test message!
This is a test message!
This is a test message!
This is a test message!
This is a test message!
This is a test message!
This is a test message!
This is a test message!

The length of the received message is : 216
The client closed the connection.
The total bytes received is : 240
```

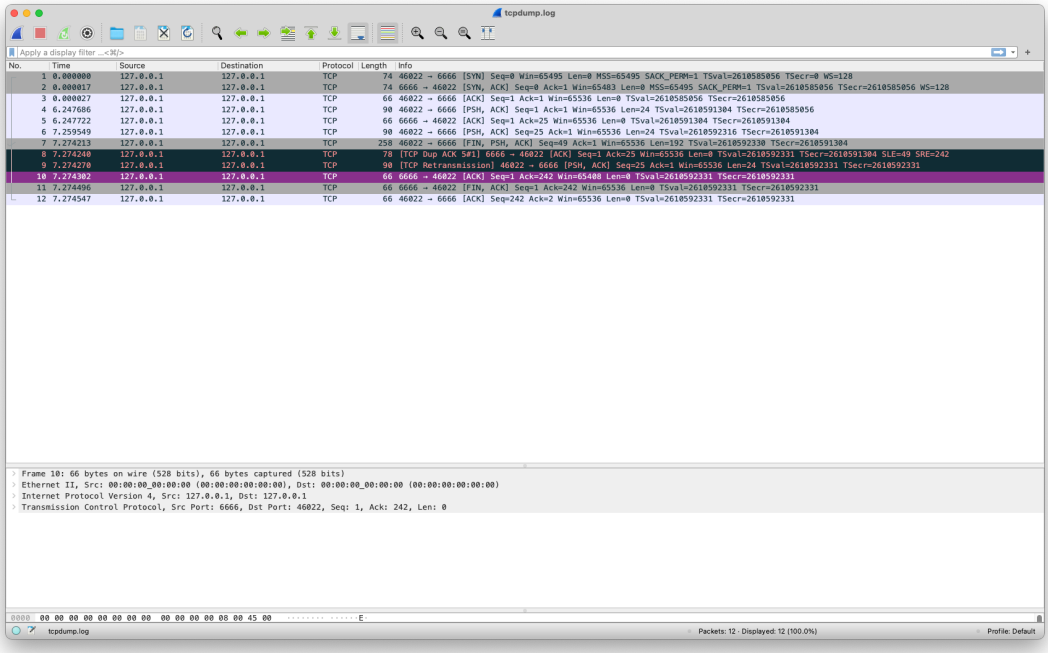
1.3.2 客户端

```
starting to send data to server, please input the data to be sent to server repeatedly:
This is a test message!
The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The message sent is:  This is a test message!

The total bytes sent to server is : 240
```

1.3.3 抓包

```
root@node1:~# sudo tcpdump -i lo tcp port 6666 -w "tcpdump.log"
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
^C12 packets captured
24 packets received by filter
0 packets dropped by kernel
```

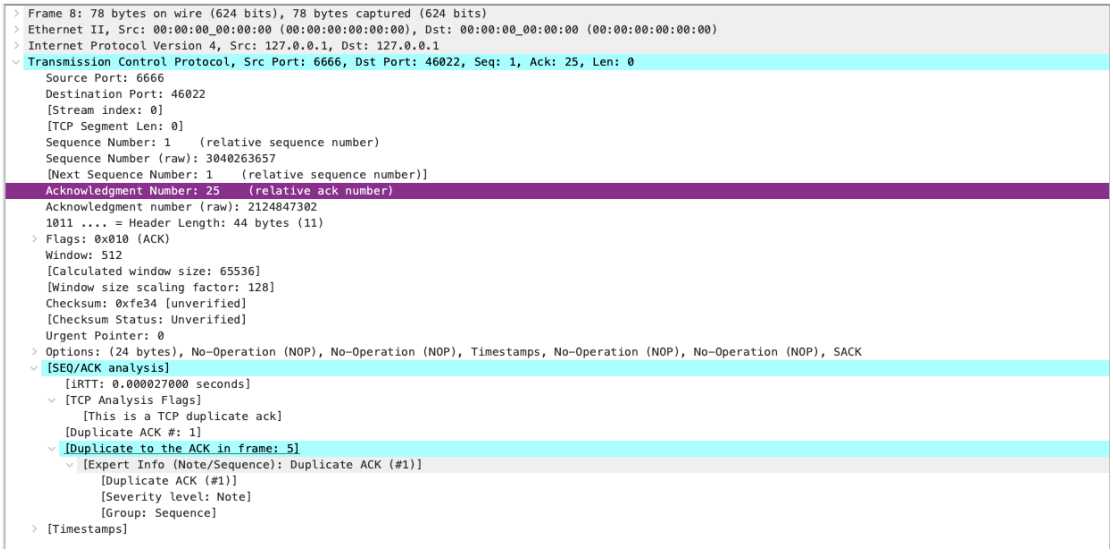
1.3.4 在 Wireshark 中查看抓包结果



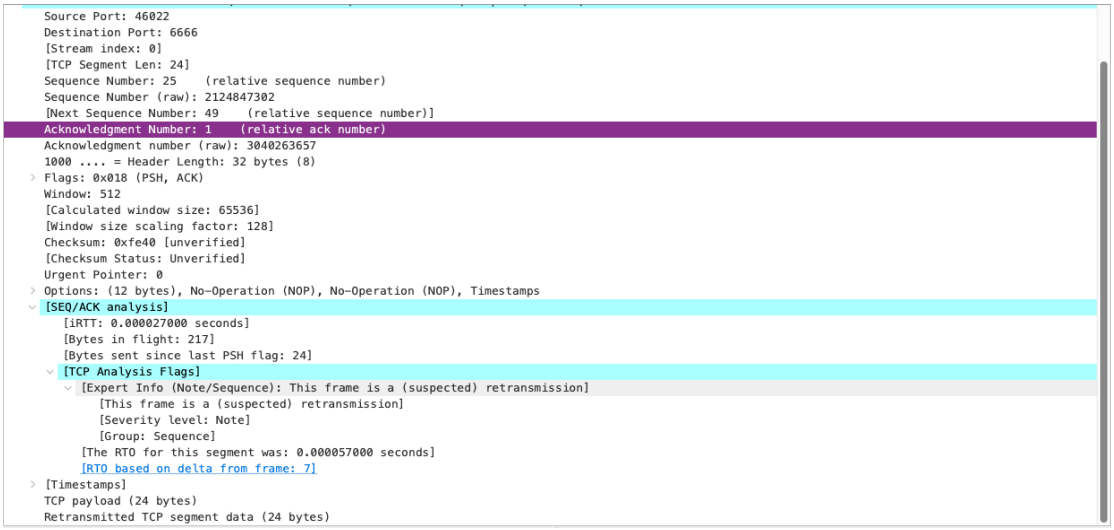
1.3.5 抓包结果分析

数据包	内容

1	第一次握手
2	第二次握手
3	第三次握手
4	客户端发送第一条消息（序列号 1）
5	服务器返回第一条消息（序列号 1）的 ACK
6	客户端发送第二条消息（序列号 25）
7	客户端发送后八条消息（序列号 49）和 FIN 消息（第一次挥手）
8	因为第二条消息（序列号 25）丢失，服务器重发接收到第一条消息（序列号 1）的 ACK
9	客户端重发第二条消息（序列号 25）
10	服务器发送接收到后八条消息（序列号 49）的 ACK（第二次挥手）
11	接收到 49 号分组的 FIN 消息后，服务器返回 FIN 消息（第三次挥手）
12	第四次挥手，连接终止



8 号分组



9 号分组

2. 开发基于 UDP 协议的可靠通信协议软件

我们的软件是基于 Ping 消息和三个字段（SEQ、ACK、FIN）来实现可靠性的。为了区别 Ping 消息和携带数据的常规消息，我们将它们分别称为 Ping 消息（PING_MSG）和常规消息（REG_MSG）。Ping 消息只携带消息类型和发送方地址，常规消息携带消息类型、发送方地址、标志位（flag）、标志位的值、和要传输的数据。详见 rudp_msg.h 中的注释。

在接下来的语境中，我们假设服务器（Server）有 10 条消息要发送给客户端（Client）。但是因为客户端也会发送（ACK）消息给服务器，所以在它主动发消息的情况下，它也可以被视作是服务器端。总之，我们可以认为数据的发送方为服务器端，接收方为客户端。因为在编写代码时是站在服务器的视角，所以代码注释中并没有使用“发送方”和“接收方”这种说法。

另外，我们的协议设计参考了 TCP 协议，但是因为比较简单，在某些场景下不能做到绝对可靠。

2.1 协议机制

2.1.1 建立连接

首先，当服务器有数据要发送给客户端时，它会向客户端周期性地不断发送 Ping 消息。当客户端接收到了服务器的 Ping 消息后，得知服务器有消息要发给自己，便会开始向服务器周期性地发送 Ping 消息。当两边都互相收到 Ping 消息后，我们就认为连接建立起来了。

相比 TCP 协议的三次握手机制，因为我们在建立连接时没有 SEQ 和 ACK 的传递，可能会导致连接残留的问题，也就是连续的两次连接请求可能会被视为一个。而且在传输完成后，我们的协议本应可以进入等待状态等待下一次连接，但是它却会因为收到了延迟的 Ping 消息而马上开始下一次的传输。

2.1.2 数据传输

与 TCP 协议类似，我们的传输依靠 SEQ 和 ACK 字段来确保可靠性。当客户端接收到了序列号为 seq，长度为 len 的消息，它会返回一条 ack = seq + len 的 ACK 消息。发送方发完消息后进入等待，如果它没有在 TIMEOUT 秒内收到返回的 ACK 消息，则触发超时重传。另外，与 TCP 协议一样，发送消息内容长度为零的消息（比如单纯的 ACK 消息）不会开启超时重传计时器。

另外，因为在发送时双方还在不停地给对方发 Ping，所以我们还维护了一个 ping_counter。如果连续 DOWN_THRESHOLD 个周期内没有接收到对方的 Ping 消息，就认为对方暂时下线了，直到下一次收到 Ping 消息之前，发送方为了节约资源不会继续重传。

有了这个机制，我们可以加入另一个机制：如果某一方下线了，然后马上又重新上线，或者是出现了几次这种情况，另一方可以认为此时网络延迟太高了，可以将 DOWN_THRESHOLD 和 TIMEOUT 适当拉长。因为时间有限，笔者并没有实现这个功能，并且也只实现了等停协议，没有实现回退 N 或滑动窗口协议。

2.1.3 终止连接

与 TCP 一样，我们采用四次握手机制终止连接。在服务器发完消息后，它会发送

FIN 消息给客户端。客户端返回 ack = seq + 1，并视情况（看自己是否还有消息要发给服务器）设置 FIN 标志位为 1.任意一方在发送了一次 FIN 消息，并且收到了它的 ACK 后，就终止连接。

为了提升效率，客户端的第一个 FIN 消息是由最后一条消息捎带的。

2.2 代码结构

本次实验代码基于实验二的第二题改进。借用了实验二中的 transport.h、transport.cpp、user_cmd.h、user_cmd.cpp，将 dv_msg.h 和 dv_msg.cpp 重写成了 rudp_msg.h 和 rudp_msg.h，以及重写了 main.cpp。接下来简述各部分实现的功能，具体描述详见代码注释。

2.2.1 rudp_msg.h 和 rudp_msg.cpp

我们在这里定义了三个标志位和消息类型。实际上，在 TCP 协议中，SEQ 是必然存在的标志位，但是在我们的方案中，为了与 Ping 消息区别开，我们的 SEQ 也设置成了可选的标志位。我们设置了 insert_header()来插入消息头（源地址和消息类型）、insert_flag_field()来插入标志位、以及 insert_message_content()来插入消息内容。

```
#define PING_MSG 0
#define REG_MSG 1  // regular message

#include <cctype>
#include <string>

// RUDP_Msg flags
const unsigned int SEQ = 0x4;
const unsigned int ACK = 0x2;
const unsigned int FIN = 0x1;
```

消息类型及标志位

除此之外，ACK 标志位只有在服务器发送的第一条消息中没有设置（打印出的值为 -1），其余消息均携带了 ACK 消息。FIN 标志位若没有设置打印 0，设置了则打印 1。其余编码解码部分与实验二类似。

2.2.2 main.cpp

这是我们的主程序。在开始时可以指定一个参数，若是“server”则运行服务器端，“client”则运行客户端。它们的区别在于服务器端会初始化 10 条消息发送给客户端，而客户端默认消息队列（messages_to_send）为空。另外，它们还会交换一下源地址和目标地址。

我们定义了三种用户操作：

- TOGGLE_DROP：启用/禁用丢弃掉接收到的常规消息（但是不会丢掉 Ping 消息），以模拟网络故障或干扰过大消息不完整；
- TOGGLE_DOWN：启用/禁用模拟节点掉线，此时终端不会接受或发送任何消息；
- EXIT：退出程序。

为方便调试，默认情况下 drop_incoming_reg_messages 为 true，node_down 为 false，双方都接收不到常规消息，但是可以建立连接。我们会利用上面的两个 TOGGLE 命令来

对我们的程序进行三次测试。（见 2.3 节）

接下来进入主循环。大致的流程是，先检查用户输入、然后判断当前是否处于连接状态，若是，依次检查并执行：是否可以发送下一条消息、对方是否还在线、上一条消息是否需要超时重传。接下来，接受消息、根据消息类型及内容进行相应的判断（见代码 314 行的 switch-case 语句）、删除接受到的消息。

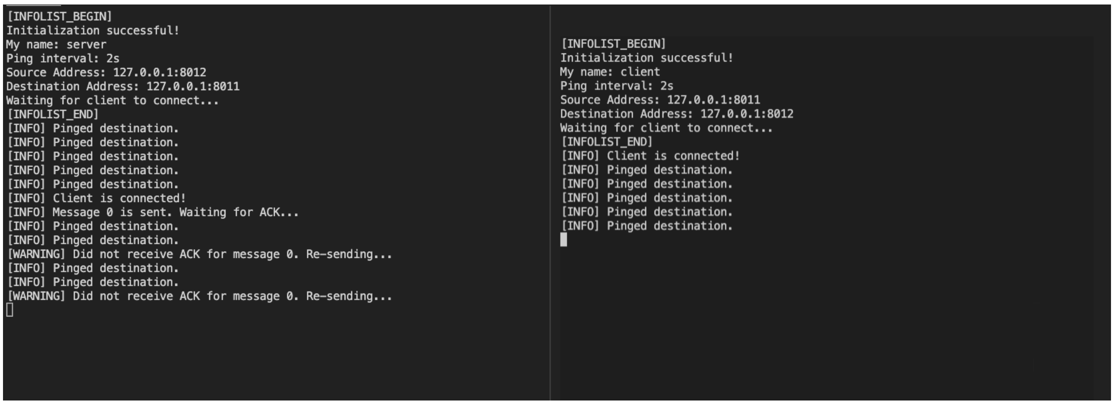
其余部分的逻辑详见代码注释，在这里不再赘述。

2.3 测试结果

下面的截图中左侧为服务器端，右侧为客户端。

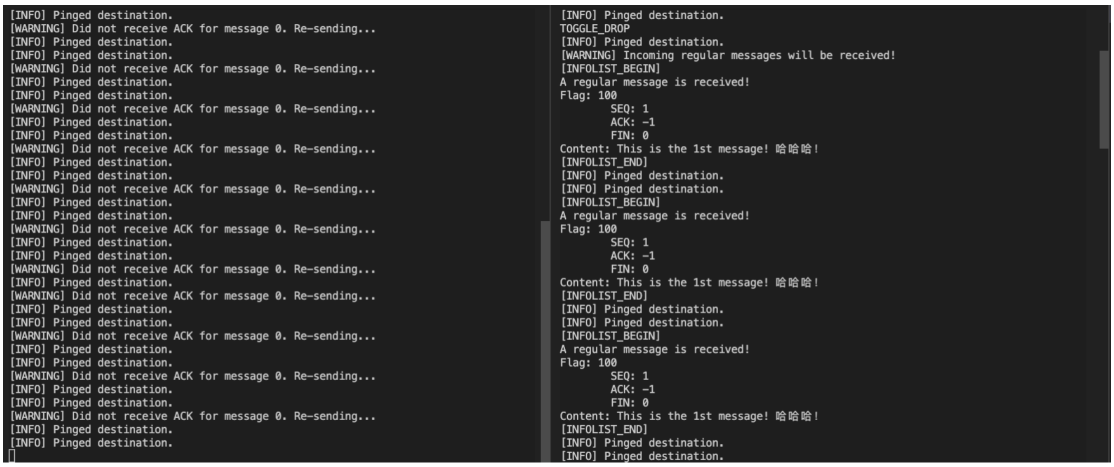
2.3.1 超时重传

a. 开启服务器，等待一会，再开启客户端：



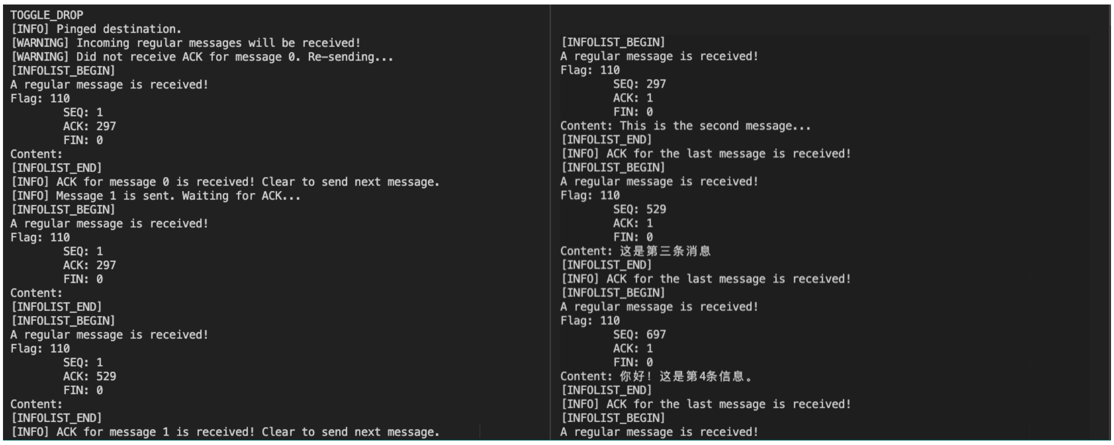
可以看到，服务器因为有消息要发给客户端，它马上开始发 Ping，客户端上线后，因为收到了服务器的 Ping，所以也开始给它 Ping。很快，服务器就发现客户端上线了。它给客户端发送了 0 号消息，然后随后又超时重传了几次。

b. 在客户端启用接受常规消息：



此时客户端可以接收到服务器发来的消息，但是服务器因为收不到客户端的 ACK，所以一直在超时重传。

c. 在服务器启用接受常规消息



```
TOGGLE_DROP
[INFO] Pinged destination.
[WARNING] Incoming regular messages will be received!
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 1
        ACK: 297
        FIN: 0
Content:
[INFOLIST_END]
[INFO] ACK for message 0 is received! Clear to send next message.
[INFO] Message 1 is sent. Waiting for ACK...
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 1
        ACK: 297
        FIN: 0
Content:
[INFOLIST_END]
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 1
        ACK: 529
        FIN: 0
Content:
[INFOLIST_END]
[INFO] ACK for message 1 is received! Clear to send next message.
```

```
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 297
        ACK: 1
        FIN: 0
Content: This is the second message...
[INFOLIST_END]
[INFO] ACK for the last message is received!
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 529
        ACK: 1
        FIN: 0
Content: 这是第三条消息
[INFOLIST_END]
[INFO] ACK for the last message is received!
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 697
        ACK: 1
        FIN: 0
Content: 你好！这是第4条信息。
[INFOLIST_END]
[INFO] ACK for the last message is received!
[INFOLIST_BEGIN]
A regular message is received!
```

此时双方开始正常通信。直到下图的四次挥手：

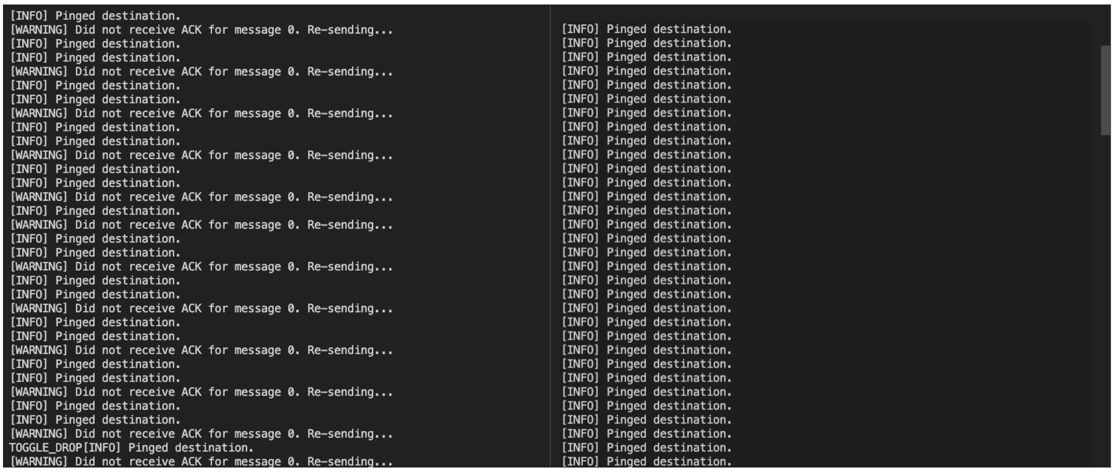
```
[INFO] Message 8 is sent. Waiting for ACK...
[INFOLIST_BEGIN]
A regular message is received!
Flag: 110
        SEQ: 1
        ACK: 1897
        FIN: 0
Content:
[INFOLIST_END]
[INFO] ACK for message 8 is received! Clear to send next message.
[INFO] Message 9 is sent. Waiting for ACK...
[INFOLIST_BEGIN]
A regular message is received!
Flag: 111
        SEQ: 1
        ACK: 1898
        FIN: 1
Content:
[INFOLIST_END]
[INFO] Transmission finished! Waiting for the next transmission...
```

```
Content: 这是第九条，也就是倒数第二条消息
[INFOLIST_END]
[INFO] ACK for the last message is received!
[INFOLIST_BEGIN]
A regular message is received!
Flag: 111
        SEQ: 1897
        ACK: 1
        FIN: 1
Content: Message No.10. 最后一条消息。
[INFOLIST_END]
[INFOLIST_BEGIN]
A regular message is received!
Flag: 111
        SEQ: 1898
        ACK: 2
        FIN: 1
Content:
[INFOLIST_END]
[INFO] Transmission finished! Waiting for the next transmission...
```

SEQ/ACK	内容解析
1897/1	服务器发送了第十条消息，并捎带了 FIN 消息，进行了第一次挥手
1/1898	客户端回复了 ACK 消息，并且捎带上了自己的 FIN 消息，第二、三次挥手
1898/2	服务器返回了客户端的 FIN 消息的 ACK，第四次挥手

2.3.2 客户端断连并重连

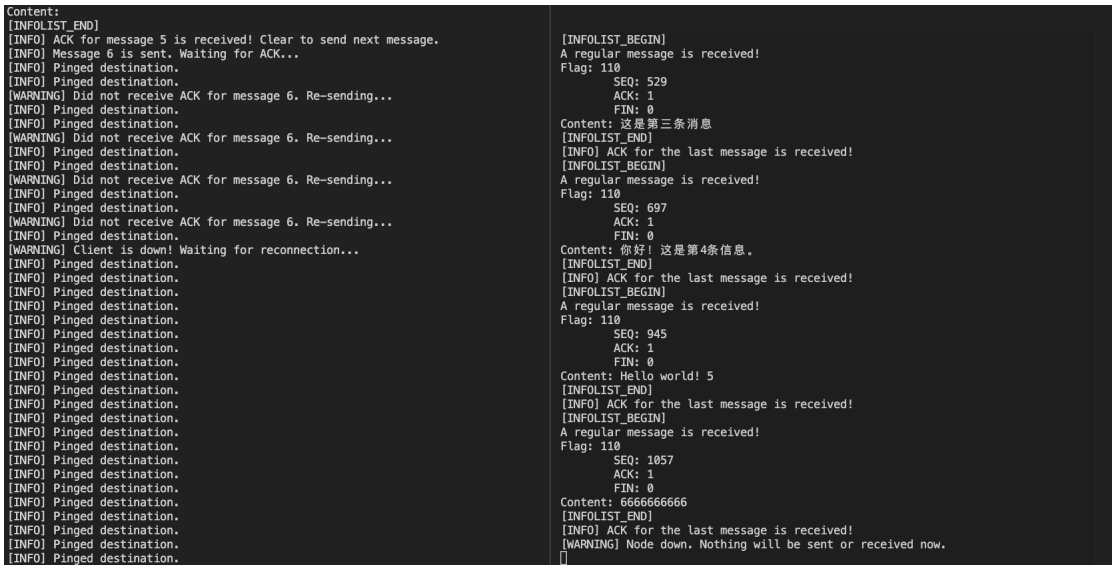
1. 开启两端，并在服务器端启用接受常规消息



```
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
[INFO] Pinged destination.
[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
TOGGLE_DROP[INFO] Pinged destination.
[WARNING] Did not receive ACK for message 0. Re-sending...
```

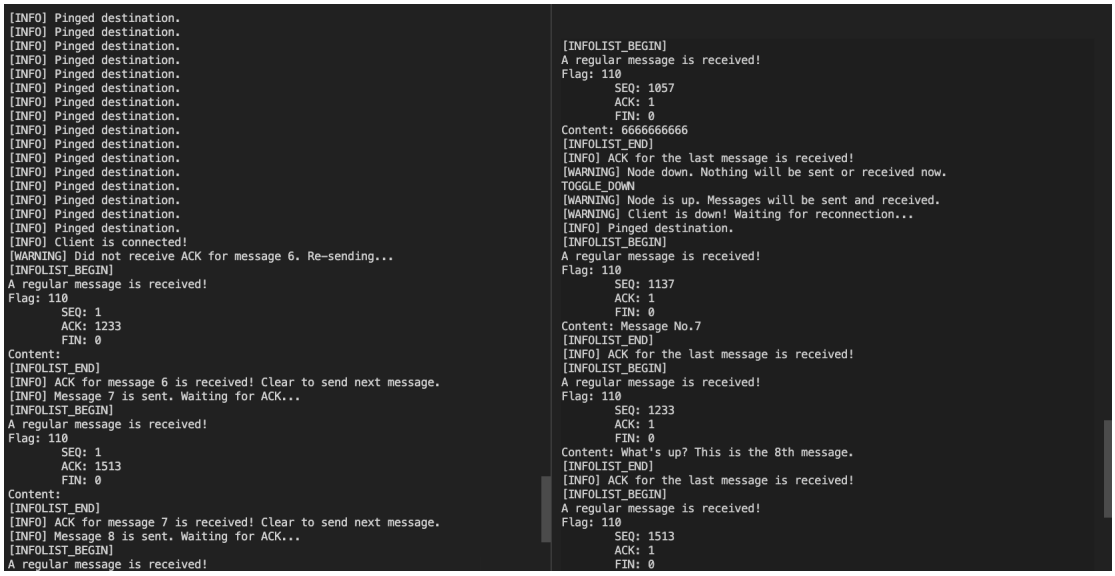
```
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
[INFO] Pinged destination.
```

2. 在客户端启用接受常规消息，然后马上将它下线（见 readme.txt）



此时客户端接收了 6 条消息，然后掉线了。服务器在重传了几次第六条消息后，意识到了客户端已掉线，停止了重传，但是它还在一直给客户端发 Ping，祈求重连。

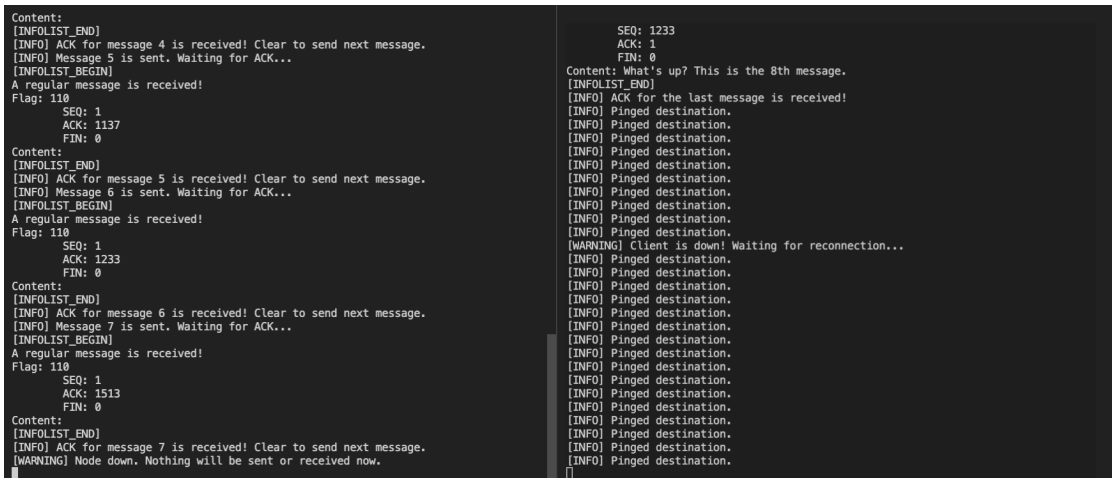
3. 重新将客户端上线



客户端上线后，它们马上重连，并完成了传输。

2.3.3 服务器断连并重连

与上面步骤一样，只不过角色互换了。如下图，服务器停止了传送，与 TCP 协议中一样，客户端也不会重传 ACK。



如下图，服务器重连，传输继续。