计算机网络及应用
基于中央定位服务器的 P2P 网络聊天系统

班级: 自 72
姓名：程晔安
学号: 2017011627

实验时间 2019 年 12 月 17 日

目录

1 题目描述 **1**

2 总体设计 **1**

2.1 整体架构 . . . 1
2.2 窗口模块 . . . 2
2.2.1 登录窗口 . . . 2
2.2.2 通讯录窗口 . . . 3
2.2.3 聊天窗口 . . . 3
2.3 CS 交互模块 . . . 4
2.4 P2P 交互模块 . . . 5
2.4.1 发送数据 . . . 5
2.4.2 开始和结束监听 . . . 5
2.4.3 数据结构 . . . 5
2.5 其他模块 . . . 6
2.5.1 内部传输模块 . . . 6
2.5.2 保存模块 . . . 6

3 设计细节 **6**

3.1 (必做) 账号登录上下线 . . . 6
3.2 (必做) 维护通讯录, 查询好友是否在线 . . . 6
3.2.1 维护通讯录部分 . . . 6
3.2.2 查询在线部分 . . . 6
3.3 (必做) 基于 TCP 协议的 P2P 文字通信 . . . 6
3.3.1 发送消息 . . . 7
3.3.2 收到消息 . . . 7
3.4 (必做) 文件传输 . . . 7
3.5 (必做) 友好的用户界面 . . . 9
3.6 (选做) 基于 UDP 协议的 P2P 文字通信 . . . 9
3.7 (选做) 群聊 . . . 11
3.7.1 群聊信息的储存 . . . 12
3.7.2 群聊的文字通讯 . . . 13
3.8 (选做)P2P 文件分发 . . . 13
3.8.1 文件分发 . . . 14
3.8.2 接收主机逻辑 . . . 15
3.9 (选做) 其他功能 . . . 16
3.9.1 表情包图片发送 . . . 16
3.9.2 聊天记录保存 . . . 16

4 结果展示 17

5 实验总结 20

1 题目描述

本次项目要求实现的是一个基于中央定位服务器的 P2P 聊天系统, 其中聊天成员使用自己的学号作为账号, 可以通过中央定位服务器互相查询在线状态. 如果在线, 可以获得对方的 IP 地址, 基于 IP 地址进行进一步的聊天和文件传输功能.

具体来说, 在本项目中实现了的项目的要求如下所示:

1. (必做) 账号登录上下线
2. (必做) 维护通讯录, 查询好友是否在线
3. (必做) 基于 TCP 协议的 P2P 文字通信
4. (必做) 文件传输
5. (必做) 友好的用户界面
6. (选做) 基于 UDP 协议的 P2P 文字通信
7. (选做)P2P 文件分发
8. (选做) 群聊
9. (选做) 其他功能

2 总体设计

2.1 整体架构

本次设计使用 C# 语言实现,.NET Core 的版本是 3.0.1, 编写和调试采用的操作系统是 Windows10.

首先描述我的整体设计, 整体设计分为与中央服务器的交互模块 (CS Core), 与其他主机的交互模块 (P2P Core), 窗口模块.

整体的模块间交互逻辑模仿了互联网的架构, 做到了分层化的设计逻辑, 基本上实现了前端窗口程序和后端发送消息文件的分离. 如下所示为整体交互逻辑.

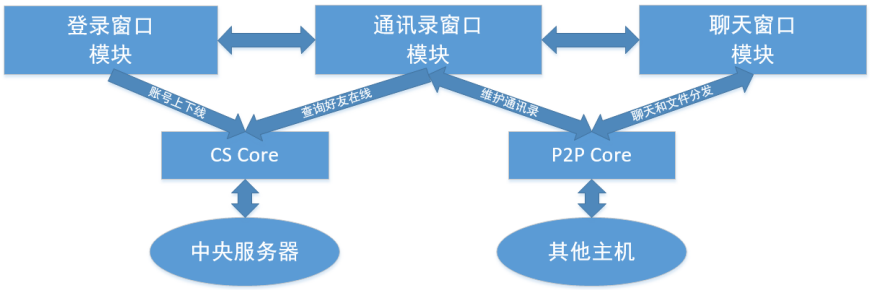

图 1: 模块间通信

下面分别描述各个模块的实现.

2.2 窗口模块

窗口模块分为登录窗口, 通讯录窗口和聊天窗口. 三者之间的交互逻辑如下所示:

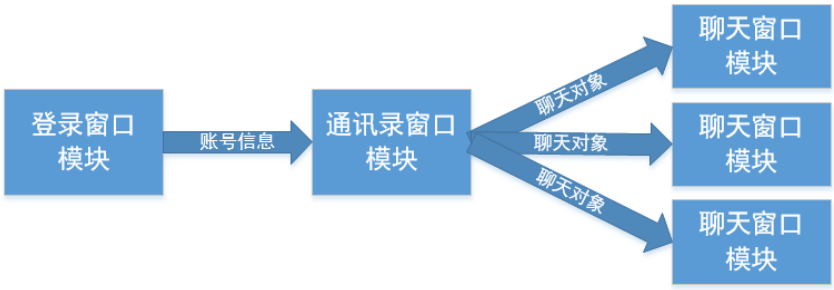

图 2: 窗口间通信

在本次任务中, 我模仿 QQ 的设计, 给予用户同时与多个用户聊天的功能. 这些窗口都与通讯录窗口直接关联, 通讯录窗口放置了交互的主要逻辑.

2.2.1 登录窗口

登录窗口如下所示.

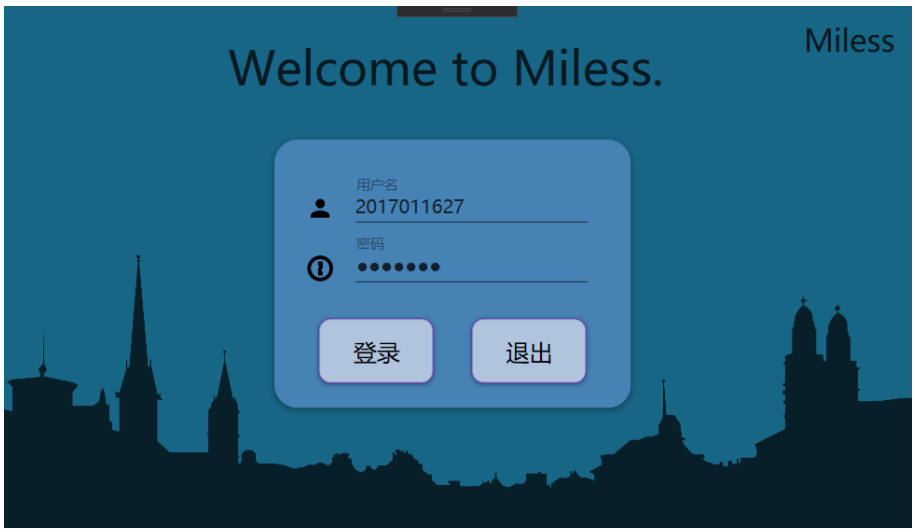

图 3: 登录窗口

首先, 登录窗口通过 CS Core 进行查询得到用户输入的账号和密码是否正确, 如果正确即传入正确的 ID 信息, 进行登录, 将这些信息转到通讯录窗口中.

2.2.2 通讯录窗口

通讯录窗口如下所示.

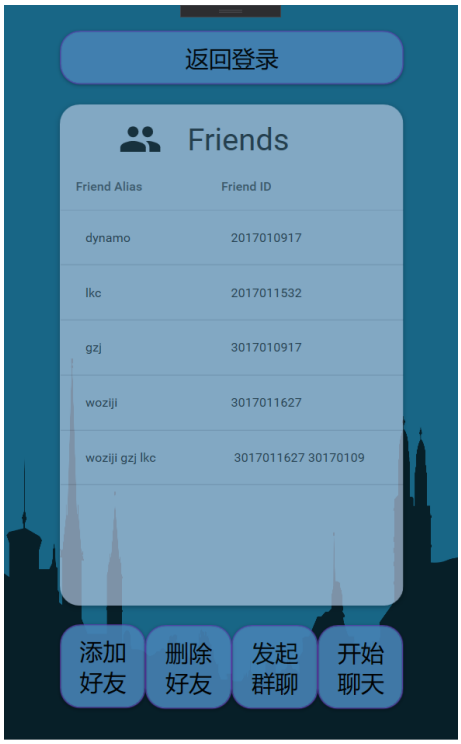

图 4: 通讯录窗口

通讯录窗口是整个逻辑的核心窗口逻辑. 仿照 QQ 的设计, 我利用通讯录窗口储存通讯录信息和已经打开的聊天窗口的信息. 本窗口中有添加, 删除好友, 创建群聊和开始聊天功能.

添加好友功能的实现是通过 CS Core 与中央服务器进行交互, 在检测对方的账号合法之后即可将其加入到通讯录中.

开始聊天功能的实现方法是: 首先通过 CS Core 检测好友是否在线, 如果在线即传入聊天对象信息, 打开聊天窗口开始聊天.

开始群聊功能的实现逻辑是首先选择聊天成员, 向所有成员发送”Greetings” 信息, 向所有成员告知群聊的诞生, 然后开始群聊.

2.2.3 聊天窗口

聊天窗口如下所示:

图 5: 聊天窗口

聊天窗口实现的功能主要是进行 TCP 文字通讯,UDP 文字通讯, 文件发送. 由于这三个这三个功能主要为后端完成, 我在之后的” 设计细节” 部分详细说明.

2.3 CS 交互模块

本部分的主要功能是完成与中央服务器的通信. 公布的交互指令如下所示:

服务器地址：（之后公布） 端口：（之后公布）		
通信类型	**客户端发送指令**	**服务器返还指令**
登录	用户名：本人学号，密码：net2019 例：“2017011000_net2019”	“lol”
查询好友状态	“q+好友学号” 例：“q2019011001”	IP 地址（在线）/“n”(不在线)
下线	”logout+本人学号” 例：“logout2019011000”	“loo”

图 6: 与公共服务器的交互指令

按照交互的指令, 利用 C# 官方提供的 Socket 类, 将客户端的指令进行编码, 对其回应进行解码, 即可实现登录, 查询好友状态和下线操作.

值得说明的是, 本次编程的时候, 我采用了单例模式 (Singleton pattern) 进行编程, 即确保一个类在一整个项目中只有一个实例, 这样就不会产生在项目中各个实例进行相互冲突的情况出现. 尤其是在 P2PCore 的设计中, 如果不使用单例模式, 很有可能启动了多个监听器线程, 导致程序运行过慢.

2.4 P2P 交互模块

P2P 交互模块是整个交互过程的核心. 为了使代码由更强的复用性, 我自己定义了数据结构, 再进行传输. 值得说明的是, 我实现了 P2P 文字通讯的 TCP 传输和 UDP 传输. 这里采用了单例模式.

P2P 模块中的主要功能是发送数据, 开始和结束监听. 下面分别进行介绍.

2.4.1 发送数据

发送数据的过程是: 首先利用 CS Core 获得的聊天对象的 IP 信息和规定的端口. 如果为 TCP 传输, 则和对方建立 TCP 连接, 再通过流传输数据. 如果为 UDP 传输, 就直接利用 IPEndPoint 直接进行传输.

2.4.2 开始和结束监听

首先建立两个新线程分别监听 TCP 报文和 UDP 报文. 由于端口为事先指定, 只需要在这两个端口进行轮询, 如果 TCP 流中有数据或者接收到了 UDP 数据, 按照之前规定的数据结构进行解包, 传入 interThreads(之后介绍) 即可.

在实际编写过程中, 我另开了一个线程用作 UDP 报文超时的轮询线程. 具体用法见具体” 实现 UDP 文字通讯” 部分应用实现中的分析.

2.4.3 数据结构

首先, 我自己创造了数据协议, 如下所示:

名称	数据类型	传输时大小	说明
BeginMatch	uint	4B	起始端校验和
Type	enum	4B	传输数据类型
SrcID	string	4B	发送方账号
DstID	string	4B	目的方账号
FileName	string	4B	文字内容/文件名
FileNameLength	int	4B	文件名长度
GroupIDLength	int	4B	群组名称长度
GroupID	string	动态	群组名称
FilePart	byte[]	动态	文件编码
EndMatch	uint	4B	结束端校验和

图 7: 数据协议

在实际传输过程中, 按照上图中的顺序将数据转为字节数组, 最后通过 TCP 进行传输即可.

2.5 其他模块

除了上述主要模块, 还有其他模块进行辅助操作, 主要有内部传输线程模块和保存模块. 下面分别进行说明:

2.5.1 内部传输模块

只有监听线程, 可以将收到的数据进行解包, 但是无法进行转存. 在本项目中, 我将所有收到的数据包放在一个 List 中, 再使用另一个线程上轮询这个 list, 如果有数据就拿出进行处理. 这样的处理可以实现信息的分层使用, 既提高了安全性, 又体现出了前后分离, 分层的方法特点.

2.5.2 保存模块

本次实验的所有保存和聊天记录历史模块都由本模块完成. 具体来说, 我利用对话双方的名称作为标识, 利用.txt 文本文件利用某种固定格式进行聊天记录的保存和重新放入.

3 设计细节

下面针对作业要求中的每个得分点进行项目的说明.

3.1 (必做) 账号登录上下线

本部分主要要求能够创造套接字与中央服务器进行通信. 在项目中, 我使用 c# 官方的 Socket 类进行实现, 再将输入按照 UTF8 编码进行解包.

3.2 (必做) 维护通讯录, 查询好友是否在线

3.2.1 维护通讯录部分

维护通讯录部分, 首先在每次登录过后, 我会根据登录的信息读入对应账号的通讯录文本文件, 之后通过 C# 中的 TreeView 控件与后端的 ObservableCollection 进行绑定, 完成通讯的实时显示. 删除和添加好友之后, 将通讯录的变化首先保存在内存中, 在退出登录以后将内存中的通讯录覆盖写入文本文件, 即完成了通讯录的维护.

3.2.2 查询在线部分

在每次打开对话框时, 我都会对好友进行在线检测. 如果不在线则无法进行聊天. 这样的处理的原因是本次实验只提供了中央服务器, 没有办法在不知道对方 IP 地址的情况下给对方进行留言. 通过这样的方式实现了查询好友在线的功能要求.

3.3 (必做) 基于 TCP 协议的 P2P 文字通信

本部分是本次作业的主要逻辑部分. 我通过发送消息和收到消息两个部分分别说明.

3.3.1 发送消息

发送消息的系统流程图如下所示:

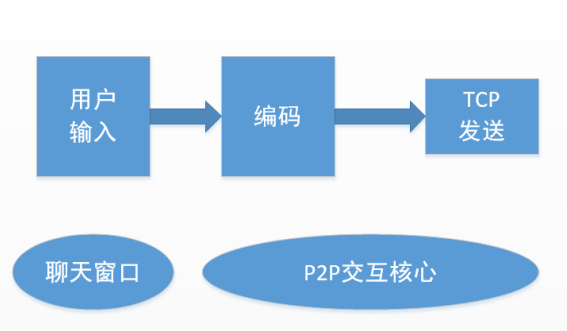

图 8: 发送消息

其中前端窗口从用户输入框获得用户输入的信息, 将其包装成我的数据格式 myDatagram, 其中加入发送方和目标方的 ID, 信息类型等等信息. 之后调用 P2P 交互核心的 Senddata() 函数, 即可通过 Tcp 将 MyDatagram 转成的字节数组通过流传输的方式进行传输.

3.3.2 收到消息

收到消息的算法流程图如下所示:

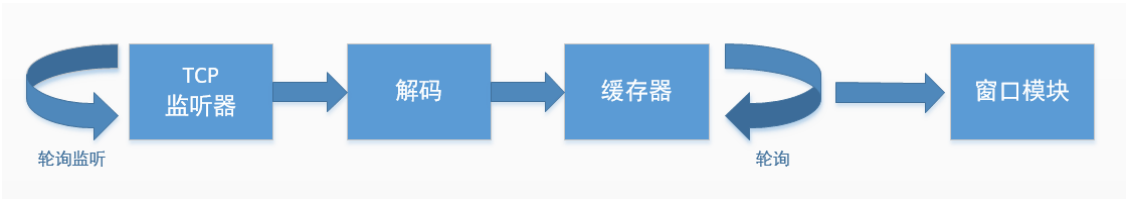

图 9: 收到消息

值得指出的是, 本部分会使用新的两个线程, 这可能会增大应用程序的运行负担. 这两个线程如下所示:

- Tcp 轮询监听器. 用来轮询监听发送到指定端口的报文 (数据流)
- 接收数据轮询读入器. 用来轮询检测已经接受到的报文.

3.4 (必做) 文件传输

文件传输版块, 我使用自己定义格式的 MyDatagram 进行实现, 进行传输的过程中可以认为文件名为之前的文字传输中的文本部分. 比较重要的是在传输过程中, 需要记录文本文件被编码成二进制之后的长度, 并作为数据报的一个部分一起进行传输. 只有这样做才可以让接收方按照长度成功解码.

在聊天界面点击文件图案即可进行文件传输. 文件大小定义为不要大于 16MB. 在进行传输之后, 在接收方会选择是否进行接收. 如果拒绝, 则会丢掉相应的数据包, 如果同意, 则可以保存在本地的对应路径下, 用户可以进行查看.

如图所示为进行文件传输的场景. 可以看到成功完成了文件传输.

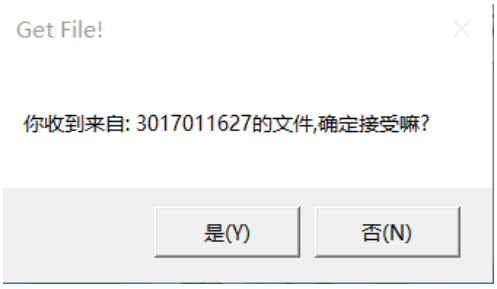

图 10: 收到文件提醒

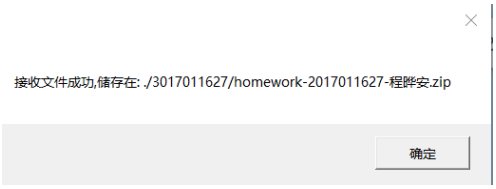

图 11: 接收成功提醒

图 12: 收到文件

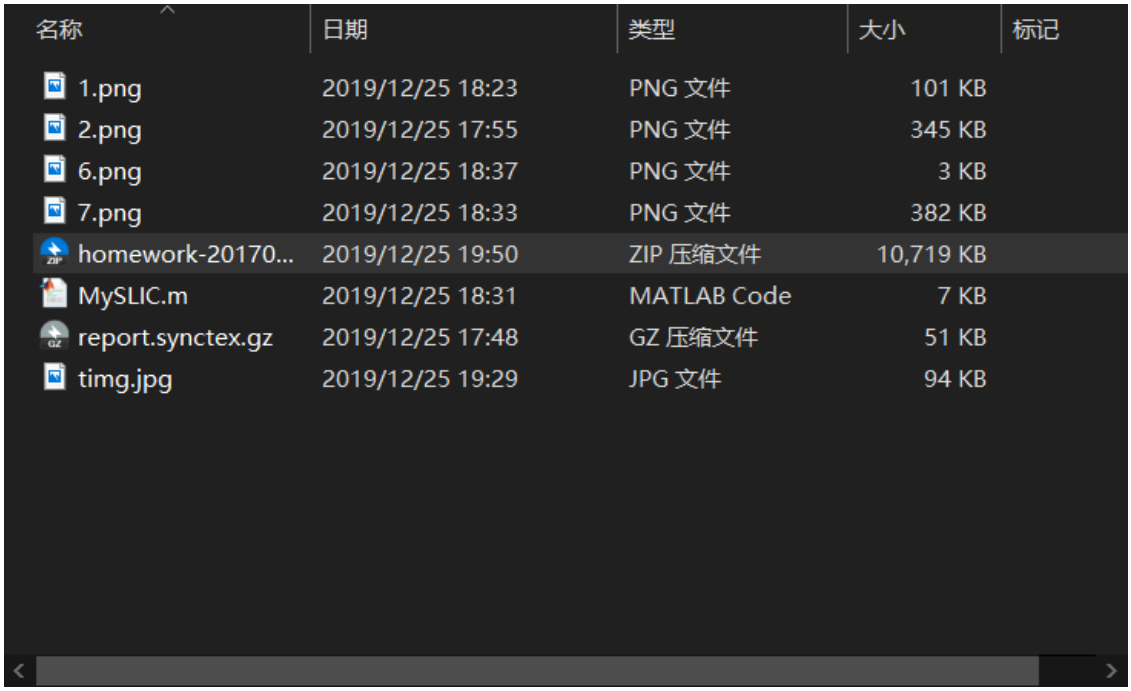

图 13: 本地传输

3.5 (必做) 友好的用户界面

本项目所有的界面截图都已经在上方的展示过程中展示. 我利用谷歌的 Meterial Design 工具箱优化了界面设计, 尽量做到扁平化和现代风格.

3.6 (选做) 基于 UDP 协议的 P2P 文字通信

正如上文所提到的, 本次代码结构为分层实现, 因此本部分的实现过程其实只需要更改为 Udp 发送和轮询接收部分, 不用改上层 (窗口层) 的命令, 也不用更改中层 (包装数据报格式), 即可以完成. 整体的逻辑图如下所示.

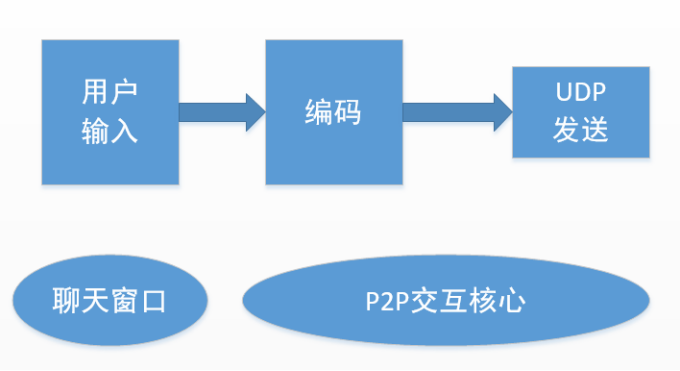

图 14: Udp 发送

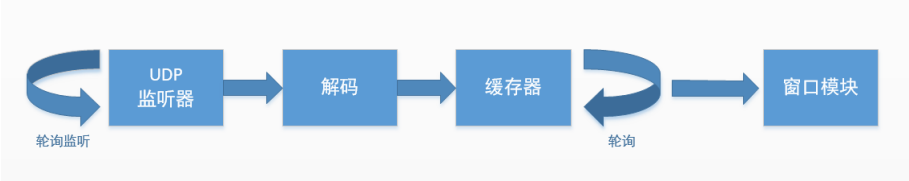

图 15: Udp 接收

具体实现使用的是 C# 自带的 UdpClient 类, 发送的过程是使用 UdpClient.Send 指令完成, 在已知数据, 数据长度, 目标主机的 IPEndPoint, 即可进行传输. 接受的过程也与 Tcp 的对应过程类似, 首先创造一个新的线程进行轮询接收, 接收到以后还是放入接收信息队列, 等待后方继续处理.

值得说明的是, 处理 UDP 报文有数据不稳定, 数据没有发送方信息的特点, 这里由于自己定义的数据协议有前后两个校验和, 一般情况下可以保证数据的正确性. 而数据没有发送方信息的缺点也被自己定义的数据协议中包含的发送方 ID 信息所弥补.

处理 UDP 报文有数据不稳定的部分逻辑如下所示:

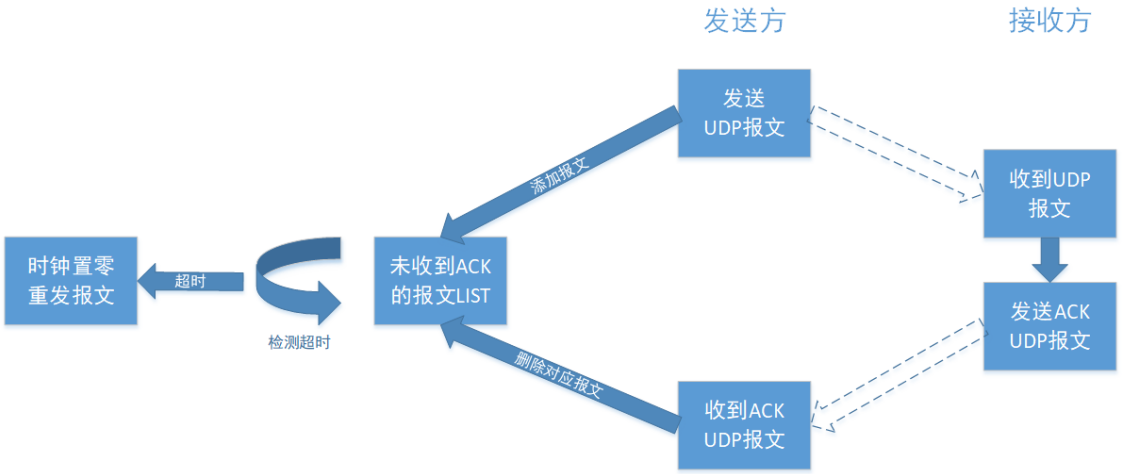

图 16: 处理 UDP 发送不稳定

为了处理 UDP 报文容易产生丢包的缺点, 我在 UDP 的发送方将发送的 UDP 报文进行储存, 并内置一个计时器. 然后在开始监听后再额外创建一个线程对储存了的 UDP 报文的时钟进行轮询检查. 如果时钟超过某个时段, 则重置时钟, 重发报文. 如果在这个超时时间内收到了 ACK 报文, 则确定 UDP 报文已经收到, 将储存的相应报文删去, 即完成了 UDP 的传输. 在实际利用 UDP 传输的过程中, 如果收到 ACK 报文, 程序会弹窗显示” 收到了 ACK 报文”, 如下所示.

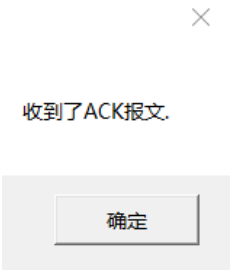

图 17: 收到 ACK

收到以后则代表本次发送是成功的.

值得指出的是, 由于 ACK 报文也有可能丢失, 其实本部分可以仿照 TCP 的是设计思路设计更多的保障传输安全的功能. 但是由于设计时间有限, 且网络状况较好 (一般没有出现 UDP 丢包情况), 这样的处理在使用过程中已经完全足够.

由于设计时间有限,Udp 传输功能不支持文件传输和群聊.

3.7 (选做) 群聊

群聊部分的实现分为群聊信息的储存和群聊的文字通讯. 下面分别进行描述. 下面是群聊的一个场景.

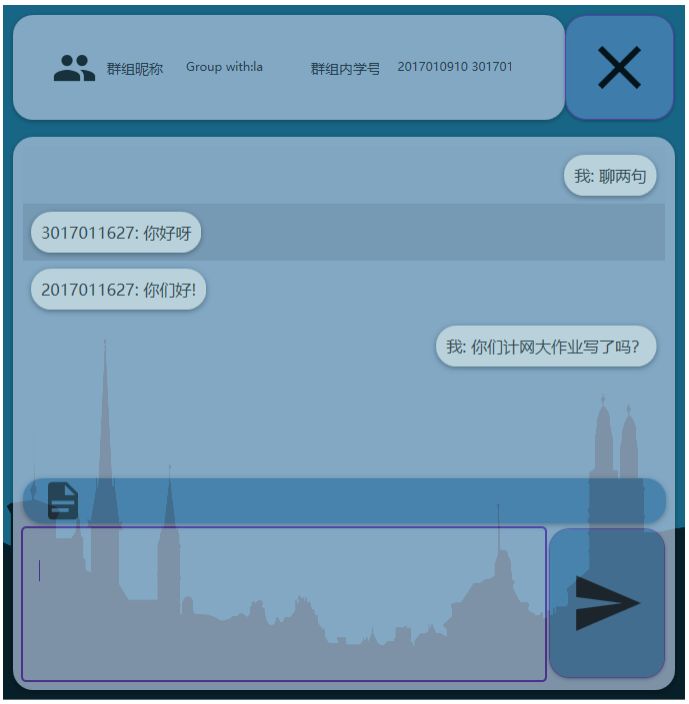

图 18: 群聊 1

图 19: 群聊 2

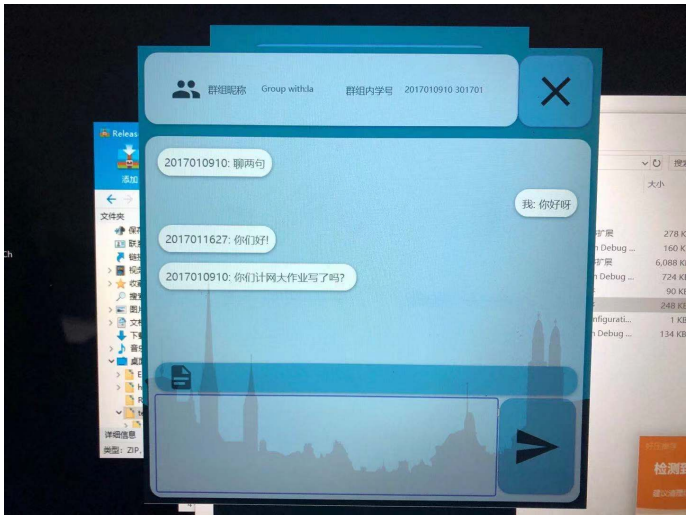

图 20: 群聊 3

3.7.1 群聊信息的储存

首先, 我将一个群聊看做一个特殊的” 好友”, 其 ID 信息即为群聊中所有用户 (包括自己) 的 ID 的字符串组合. 因此认为有相同成员的群聊为相同的群聊. 这样的设定之后, 就可以将一个群聊正常地储存在通讯录中.

在页面操作层面, 用户点击” 发起群聊” 按钮, 在列表中选择多于两个的好友, 再次点击” 确认” 即可创建群聊.

3.7.2 群聊的文字通讯

在确认进行群聊之后, 仿照 QQ 的设计, 创建群聊的时候并不用征得被拉入群聊的用户的同意, 且群聊的信息会直接出现在被拉入群聊的用户的通讯录中. 为了实现这个功能特点, 我在创建群聊窗口的时候, 首先向所有群聊中的成员发送一个固定格式的 Greetings 文字, 然后向群聊中的所有成员进行发送. 如果成员在线, 成员在解包过程中收到相应内容的 Geetings 文字, 就会直接创建出对应的群聊信息到通讯录中. 下面是实际操作中的场景.

图 21: 收到群聊邀请

之后的正常文字通讯, 和发送 Greeetings 文字的方式相似, 每发一条消息, 其实是向所有的成员发送消息, 而和前面无法给离线的成员发送消息的理由一样, 离线的成员无法收到群聊消息.

需要指出的是, 群聊功能不支持 Udp 通讯.

3.8 (选做)P2P 文件分发

整体流程分为两步进行. 首先是文件的分割和分别发送部分, 其次是各个接收主机之间的逻辑部分. 下面分别进行说明.

为了实现本部分的功能, 我在原有的数据报格式上新加了如下所示的表项.

名称	数据类型	传输时大小	说明
BeginMatch	uint	4B	起始端校验和
Type	enum	4B	传输数据类型
SrcID	string	4B	发送方账号
DstID	string	4B	目的方账号
FileName	string	4B	文字内容/文件名
FileNameLength	int	4B	文件名长度
GroupIDLength	int	4B	群组名称长度
GroupID	string	动态	群组名称
GroupFileIDLength	int	4B	传文件群组名称长度
GroupFileID	string	动态	传文件群组名称
GroupFIleIndex	int	4B	传文件群组编号
FilePart	byte[]	动态	文件编码
EndMatch	uint	4B	结束端校验和

图 22: 添加表项

添加了这三个表项之后, 就可以唯一确定某一次 P2P 文件传输中文件和各个主机.

3.8.1 文件分发

本部分流程图如下所示.

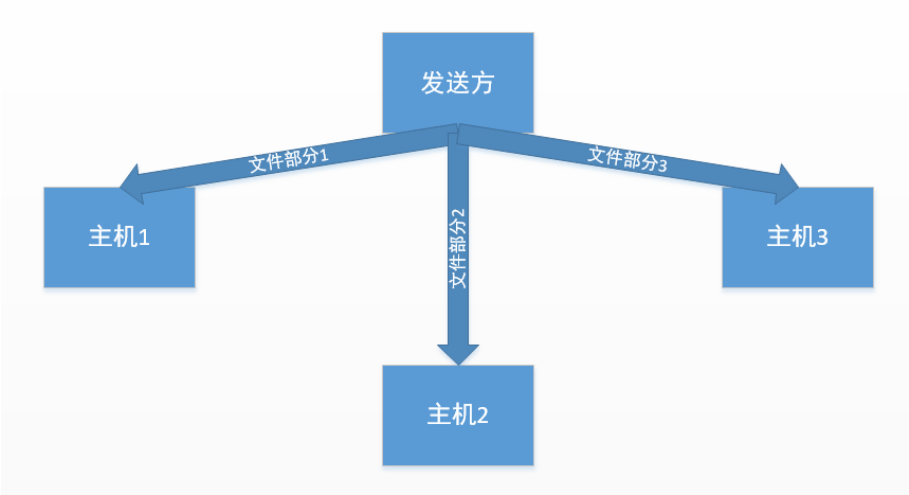

图 23: 文件分发主机逻辑

首先, 发送主机在群组中进行遍历, 找到在线的用户, 将其 ID 信息包装在数据报中 (要求顺序), 之后按照在线用户的总数依次将文件进行切割, 将切割部分和对应的部分的编号发给接收主机, 即完成了文件的上传. 这样, 上传主机虽然发送了多个数据报, 但是总共只上传了一次文件. 在之后的操作中均不包含发送主机. 因此为 P2P 操作.

3.8.2 接收主机逻辑

本部分的逻辑如下所示:

首先, 在接收主机第一次收到数据报文时, 我设计了如下数据结构来存储接收主机所拥有的数据信息.

- (string) GroupID: 为群组号. 这是基本的验证环节.
- (List of string) P2PFileIDList: 由发送主机发送的报文中的 GroupFileID 得到, 用来了解在本次文件分发中所有的其他主机信息, 以之后进行对应部分的请求.
- (int[]) P2PFileIndexPool: 用来记录哪些文件部分已经收到, 哪些部分还没有.
- (byte[]) P2PFileBuffer: 用来储存已经有的文件部分. 值得说明的是, 由于在接收主机在收到文件部分的时候并不知道整个文件的大小, 我这里采用收到的文件部分乘以一共的文件部分个数加一来作为这个缓存器的大小.
- (byte[]) P2pMyOriginalPart: 用来记录自己本来的文件部分, 以备发给请求的对象.
- (int) MyP2PIndex: 记录自己的本来的文件部分的编号.

在有了这些容器之后, 我们就可以按照如下的逻辑顺序进行操作, 直到所有文件收齐.

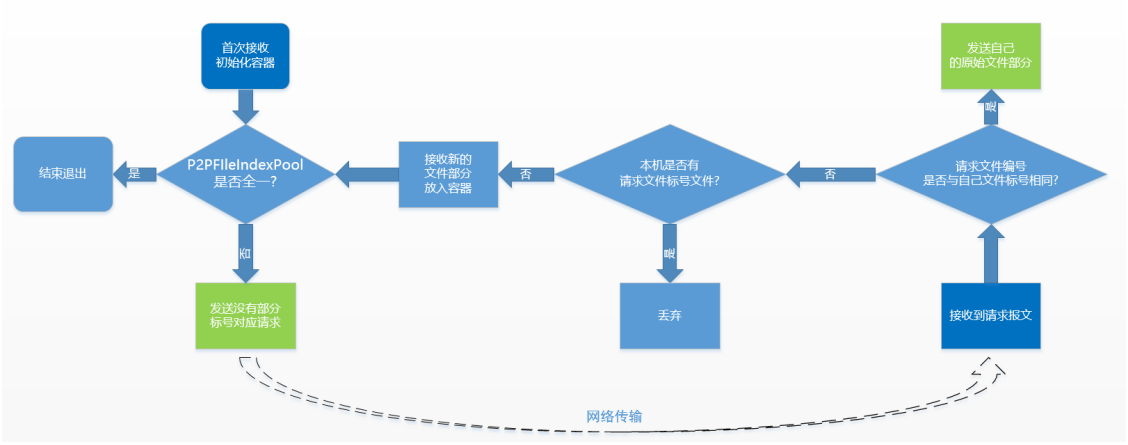

图 24: 文件分发

1. (首次接收) 初始化上述容器, 将对应元素放入容器中.
2. 检查自己的 P2PFileIndexPool, 如果有某一位为 0, 说明此位的文件部分还没有, 跳转第三步; 如果全为 1, 则说明文件已经收齐, 结束退出保存.
3. 向所有 P2P 文件传输的其他节点发送请求报文, 请求缺少的部分.
4. 如果收到 P2P 文件传输请求, 检查其文件编号, 如果本机没有部分文件, 则默认不作操作, 如果为本机原始的文件部分对应的文件标号, 则发送文件部分.

如图所示为两个主机收到不同的文件部分的截图.

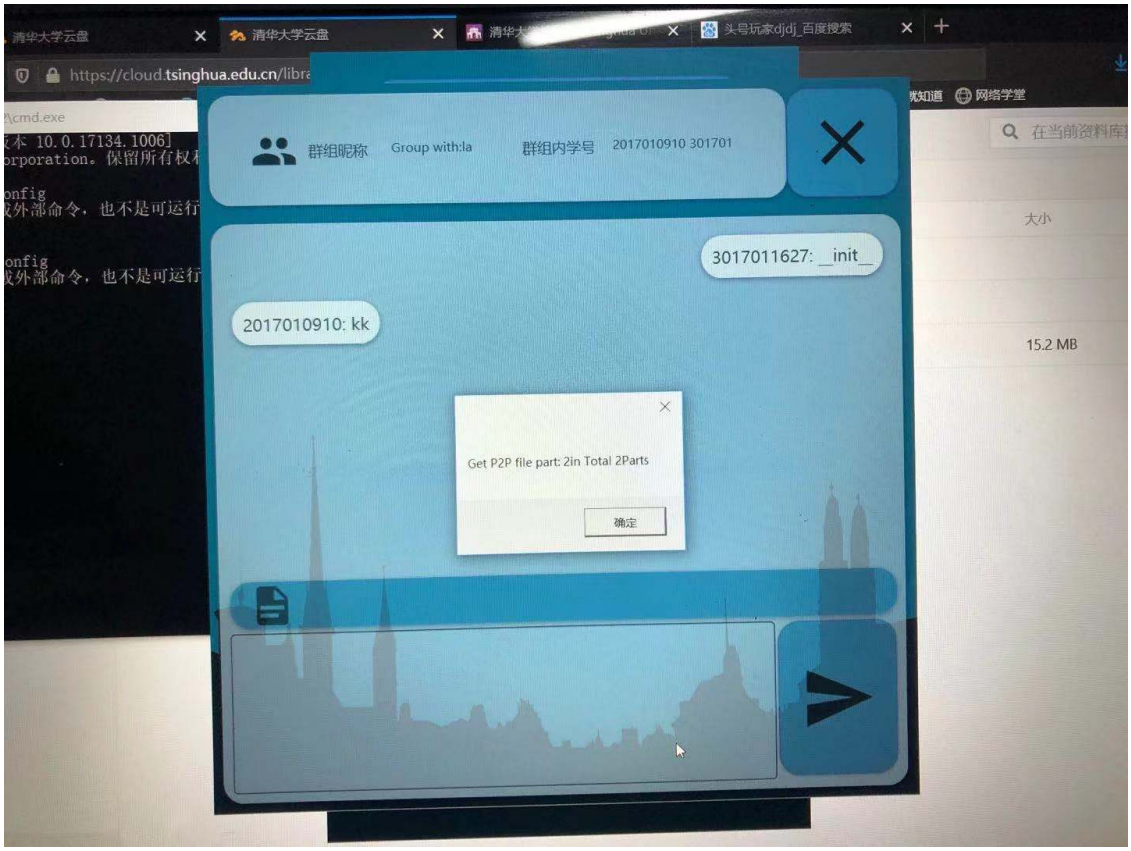

图 25: 文件部分 1

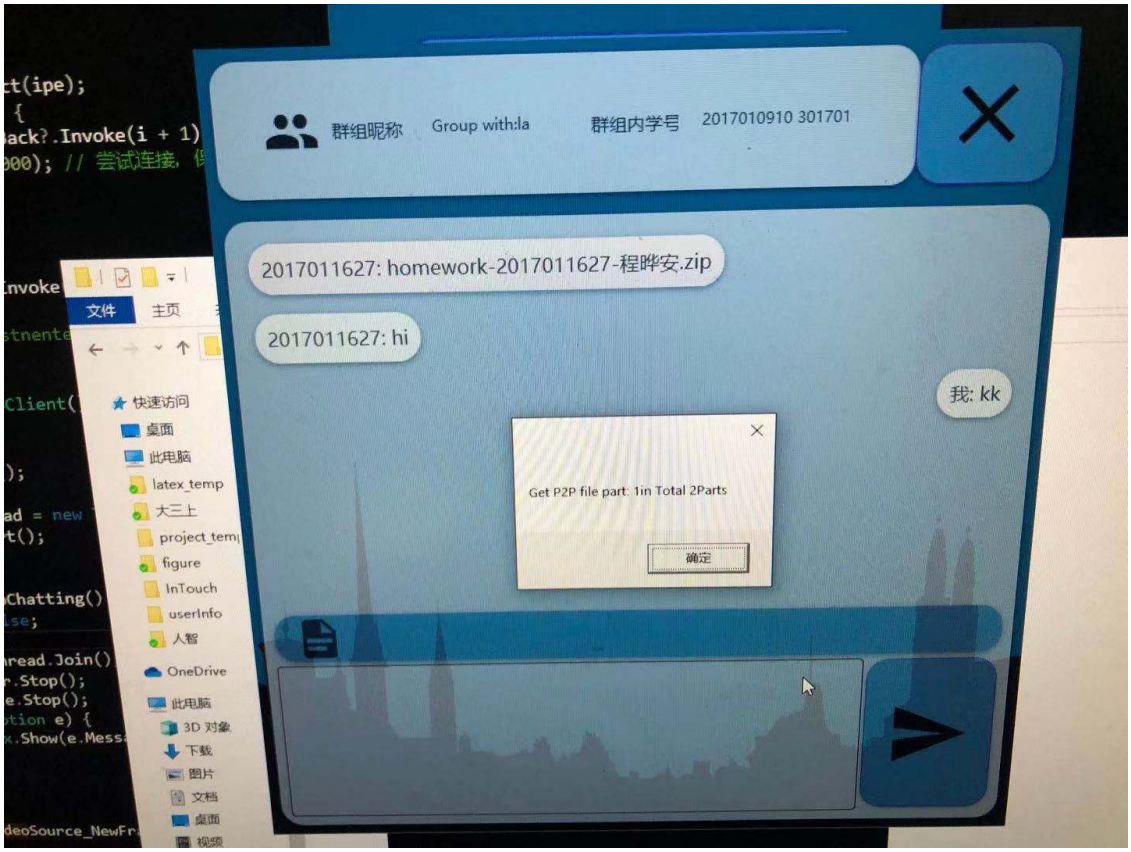

图 26: 文件部分 2

3.9 (选做) 其他功能

在要求内容的基础上, 我又实现了表情包图片的发送和聊天记录保存两个功能, 作为本次项目中的创新点. 下面分别进行说明.

3.9.1 表情包图片发送

在定义数据格式的过程中, 我发现图片文件与一般文件在转换为字符数组之后是没有区别的. 因此我们可以使用传输文件的方式来传输图片. 在传输之后, 我利用官方提供的 BitmapImage 图像格式将保存的图片进行了图片的实时显示. 值得说明的是, 由于设计时间有限, 表情包的显示只能支持双人聊天, 不支持群聊.

使用方法是直接点击传输文件的图像进行选择, 如果选择的文件是图像文件, 则会在聊天过程中直接显示出来. 下面是效果图.

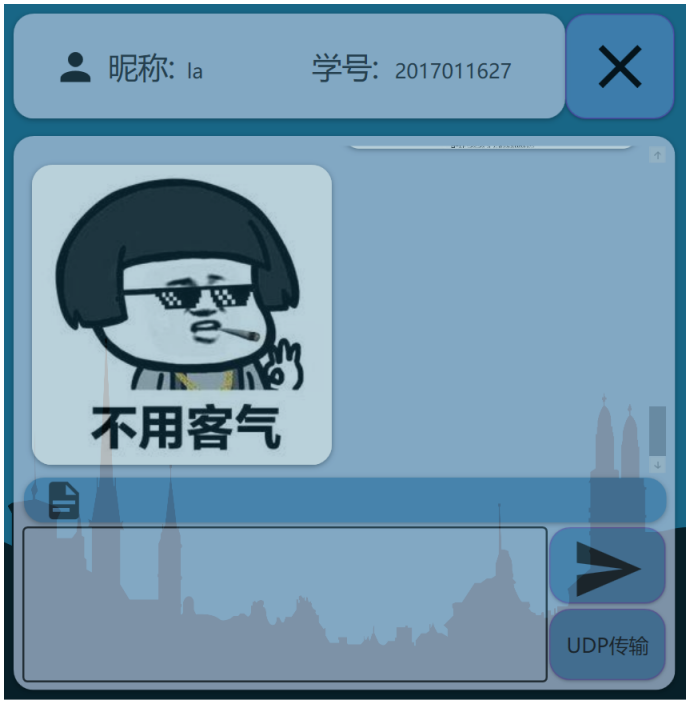

图 27: 表情包

3.9.2 聊天记录保存

在本次作业中, 为了更好地让聊天进行下去, 我实现了聊天记录的读入和保存. 由于时间关系, 我采用.txt 文本文件对聊天记录进行保存, 按照一定的保存规格 (各个信息用符号 $ 进行分割), 可以查看到用户与所有人的聊天详情. 包括发送每条信息的时间 (由于美观考虑没有加入到聊天显示中) 等等.

如下所示为一次聊天的聊天记录.

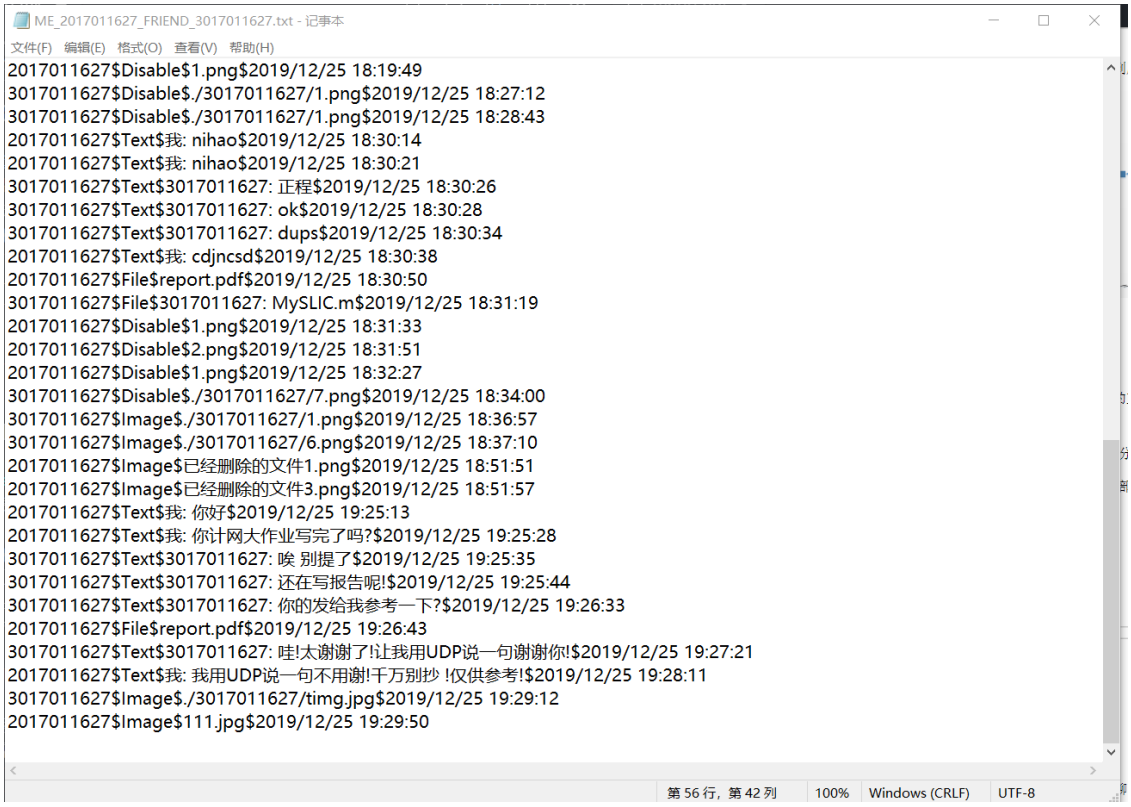

图 28: 聊天记录

4 结果展示

如下所示是我创造的一段比较完整的聊天的过程的截图.

图 29: 片段 1

图 30: 片段 2

图 31: 片段 3

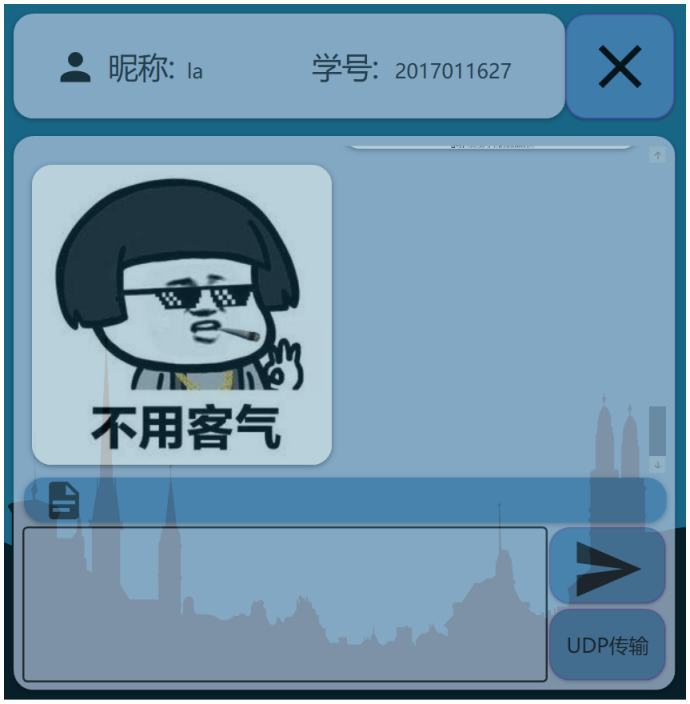

图 32: 片段 4

5 实验总结

在本次实验中, 我动手实现了基于中央定位服务器的 P2P 网络聊天系统. 我第一次系统地操作了套接字编程, TCP 和 UDP 传输等操作. 同时, 在自己定义数据报格式和进行解码编码传输的过程中, 我也更加深刻地了解了分层次网络传输和架构的相关概念. 不仅如此, 我还体验了做一个聊天系统所必要思考的程序内数据传输, 聊天程序和窗口逻辑等等. 这些都让我收获很多.