基于中央定位服务器的 P2P 网络聊天系统设计
计网大作业

自 62 韦毅轩

2016011422

weiyx16@mails.tsinghua.edu.cn

2019 年 1 月 4 日

目录

1 概要

这次大作业的主要任务是基于中主的中央定位服务器，自行编写客户端，实现 P2P 网络聊天系统的设计，并完成一系列功能。程序主界面如 Fig.1所示。

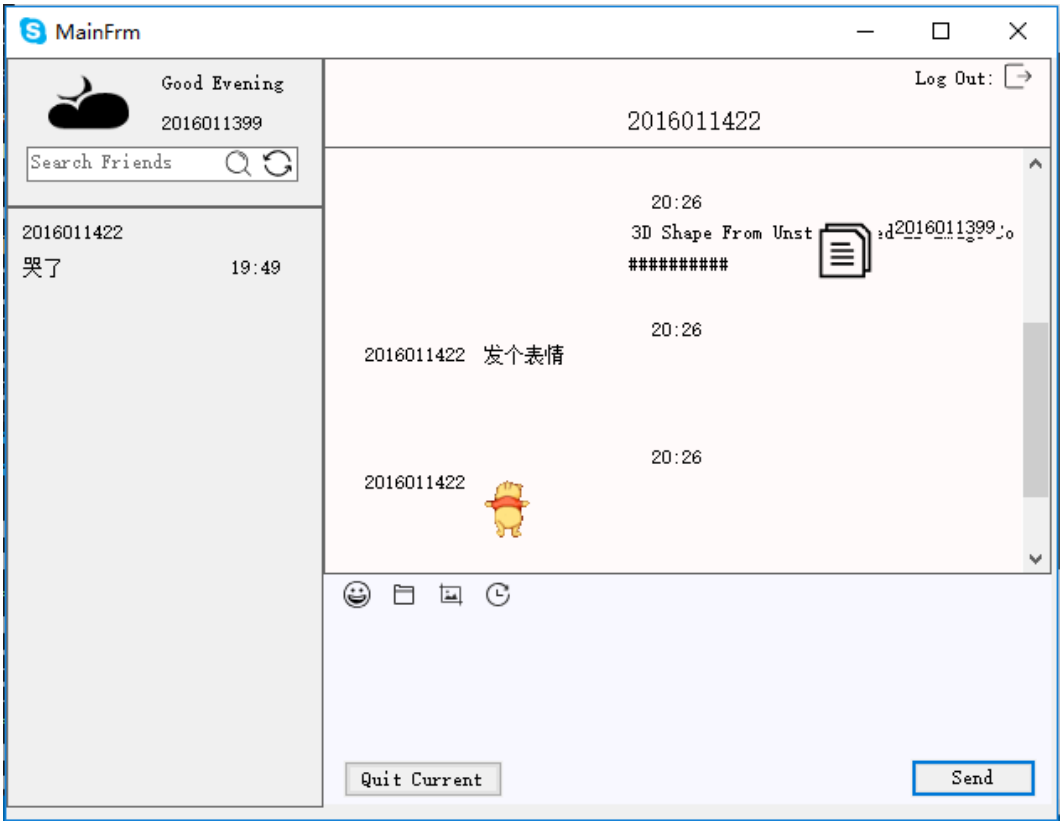


图 1: 整个界面仿照微信设计。左边上方显示当前用户名、一些必要的欢迎信息以及好友搜索界面，左边下方显示之前与谁聊过天，以及最后一条消息与最后一条消息对应的时间，便于用户继续搜索以前聊天过的好友进行聊天，右边上方显示了当前聊天的好友以及退出按键，右边中间则在聊天中显示双方互发的消息，右边下方的四个按键分别可以发送表情/发送文件/调用截图功能/查询与当前用户的聊天记录，四个按键下方即为文本输入框，点击右下角 Send 即可发送，左边的 Quit 键则可以退出当前的聊天，接受新的消息发送。

2 需求分析

2.1 必做任务

1. 账号登录上线/下线，需要该客户端能和主楼的服务器进行消息发送与接收，并对接收的消息进行分析。

2. 通讯录（查询好友是否在线），需要该客户端能发送好友用户名给服务器，并解析

好友的 Ip 地址，以便后续通信。

3. P2P 通信，需要客户端能通过 Ip 地址，实现与好友的通信，需要同时实现消息的接收与发送两部分的功能。
4. 10Mb 以上的文件传输，同样通过 P2P 实现。
5. 友好界面，主要要符合用户的交互习惯，因此采用类似微信设计，比较习惯。

2.2 选做任务

1. 群聊功能，需要实现一对多的消息传送。
2. 聊天记录查询，用户可以查询与当前用户的聊天记录，以本地文件实现。
3. 语音/动态表情，用户可以发送 Gif 等来更好地表达。
4. 其他功能，合理自由发挥。

3 总体设计

3.1 实现框架

本次大作业通过 C#+Vs2015+.Net Framework4.5.2 实现。C# 的界面编写较为友好，且用 C# 实现网络编程的典例比较多，查询起来有据可循，比较方便，故选用 C# 编写。

3.2 网络通信方法

主要使用了 C# 中的 System.net.sockets 包中集成的 TcpClient 类实现，封装了一系列的方法，实现起来较为方便。通过其建立连接之后，再采用 NetworkStream 来进行通讯数据的写入与读出。文件的读写则通过 FileStream 来实现。整个架构与作业要求中的相同，都是混合式的架构，通过服务器获得 Ip 地址，再通过 Tcp 连接来进行 P2P 的传送。

由于这次实现每个人与每个人通信的协议不同，无法进行联机测试，我都是在自己的电脑上通过登录两个账号来进行测试的。我为每个账户分配了一个独立的端口，即学号后 5 位加上 10000，一是保证每个用户端口不会冲突，二是保证不和计算机现有端口发生冲突。

3.3 功能设计

针对上述的需求分析，并结合自身能力与时间，我主要实现了如下几个功能 Fig.3.3。

功能名称	具体子功能
用户登录上下线	• 提供了服务器IP与端口更改的接口 • 提供了基本的账号有效性判断 • 保证同台电脑同账户只能登录一次
查询好友在线及好友列表	• 输入好友账号判断当前好友是否在线 • 记录之前聊天过的好友并显示最后状态
P2P通讯	• 实现二者点对点的发送与消息监听 • 只要不退出当前对话，就不会被第三方的消息打断，保证通讯质量 • 实现图片/文字的混合发送与显示 • 实现动态表情的发送与显示 • 实现大文件传输及传输状态显示
查询聊天记录	• 实现通过对本地聊天记录文件的解析，进行聊天记录的查询与现实
其他功能	• 调用系统截图工具或在客户端内自动截图实现功能，并发送图片
友好界面	• 基于微信电脑版进行外观设计，提供上述功能的接口与用户交互

图 2: 客户端主要功能

3.4 界面设计

主要界面分为两个，一个是登录界面，一个是主聊天的界面。

登录界面 如下：

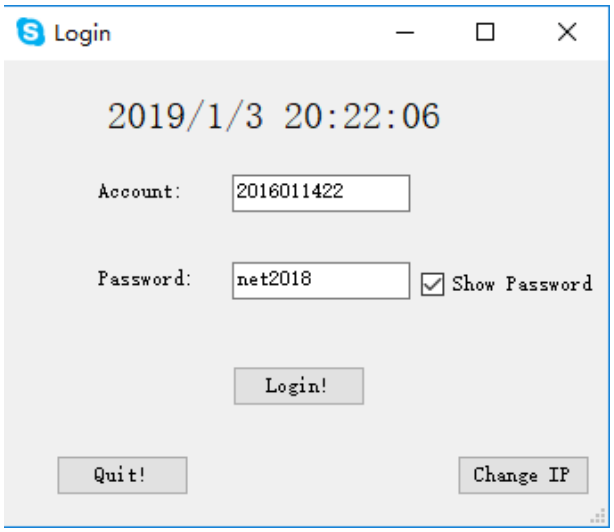


图 3: 登录界面

在登录界面中显示当前的时间，也是在提醒我进度完成的情况。由于没办法像微信那样真正用二维码扫描来进行登录，所以采用了比较简单的 QQ 登录模式。

主界面 如下：

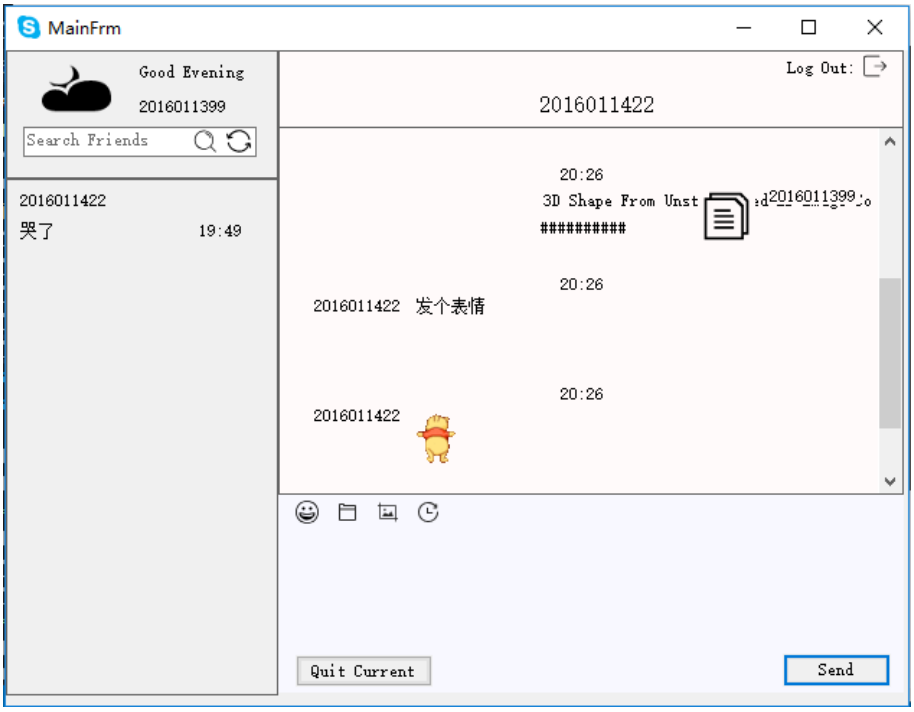


图 4: 主界面

主界面仿照微信设计，各个模块的主要功能已经在概要部分进行了描述。

其他界面 整个客户端各个主要界面及互相调用方法如下：

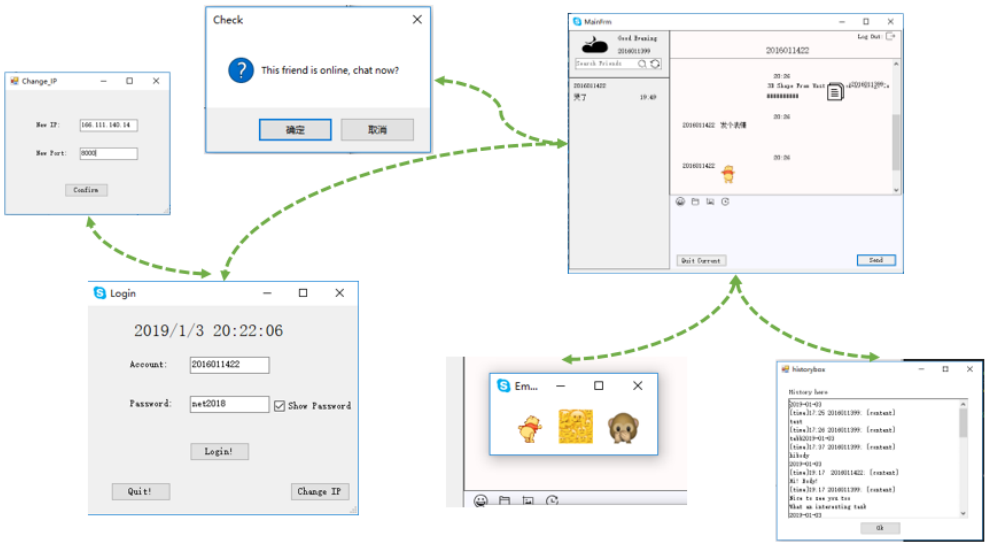


图 5: 各界面交互

3.5 程序框图

整个程序的主要逻辑如下。在 P2P 通讯中，A 用户可以直接向 B 用户发起连接，而免去繁琐的验证。若 B 用户正在与 C 用户通讯，则 A 用户无法连接到 B 用户，这样使得状态管理起来较为简单，也能让每次通讯都不会被打断。而单方面退出并不会影响另一方面继续发送，也就是说 A 与 B 聊天，A 退出聊天，返回空闲状态，若 A 没有退出或与 C 聊天，则 B 还可以继续给 A 发送消息，A 会回到当前聊天中，这样也比较合理。

具体程序的细节与程序框图有一定的不同，即在具体实现中，加了很多出错检测/对应错误操作的提示等等，限于程序框图大小有限，在此并没有完全绘出。

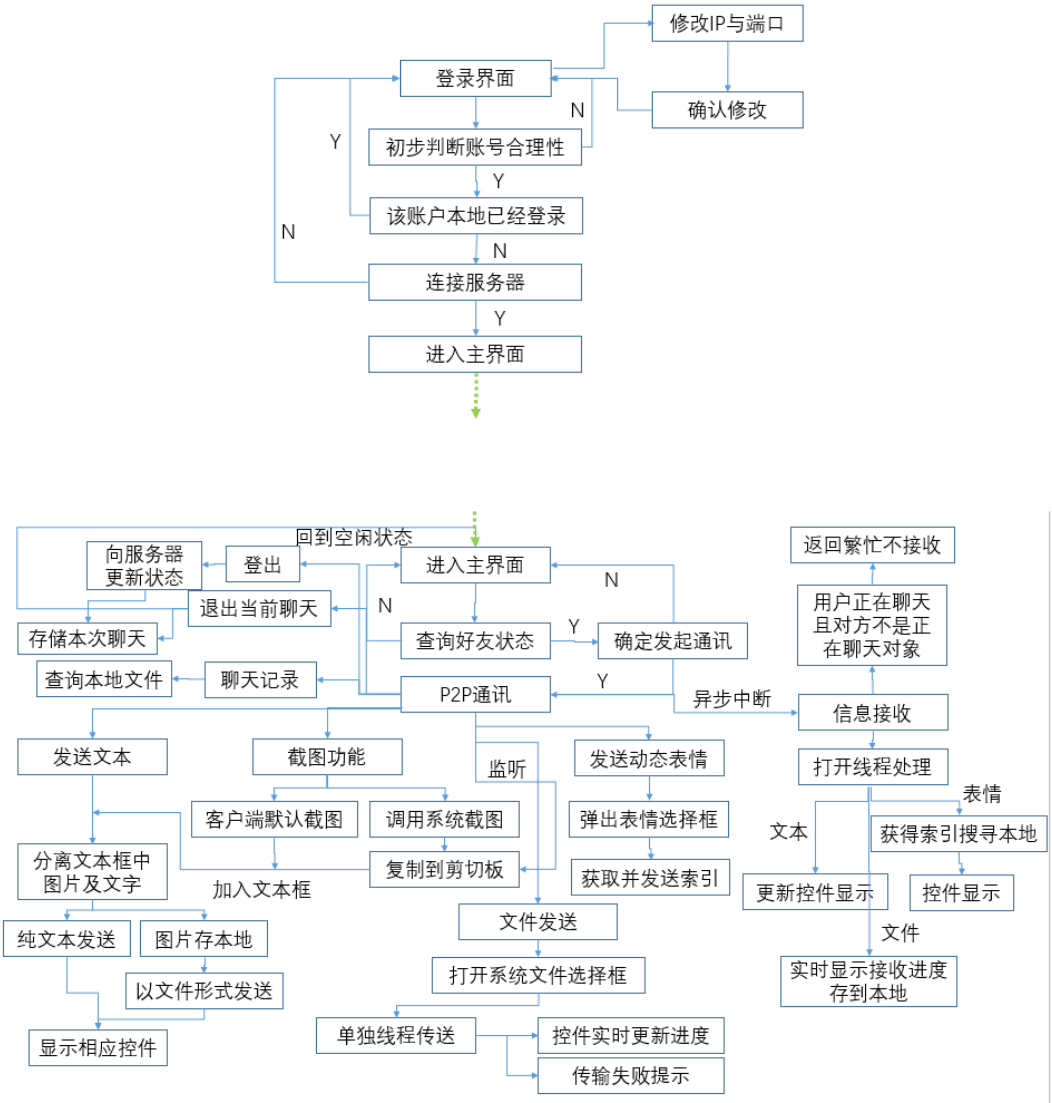


图 6: 主要程序框图

4 具体设计

在这里我分模块专门讲一下我的具体设计思路。

4.1 账号登录

登录界面比较简单，只需要输入账户和密码即可。在登录键内加入了一些基本的有效性判断，要求账号密码非空，且账户要求是 10 位，以 20160 开头。同时为登录用户分配端口，如果端口重复，则判断用户为反复登录，拒绝本次登录。利用 C# 自带的 MessageBox 窗口，发生错误时弹出对应的提示信息，并要求用户进行重新登录。比如重复登录报错如下：

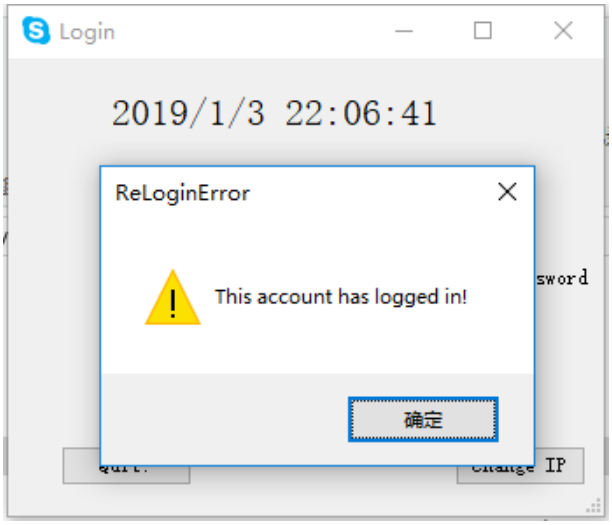


图 7: 本地重复登录弹出对应信息

考虑到当服务器 IP 地址与端口发生改变时，用户重新使用下的方便性，设计了 Ip 地址与端口更改的界面，允许用户指定服务器的相关配置信息。

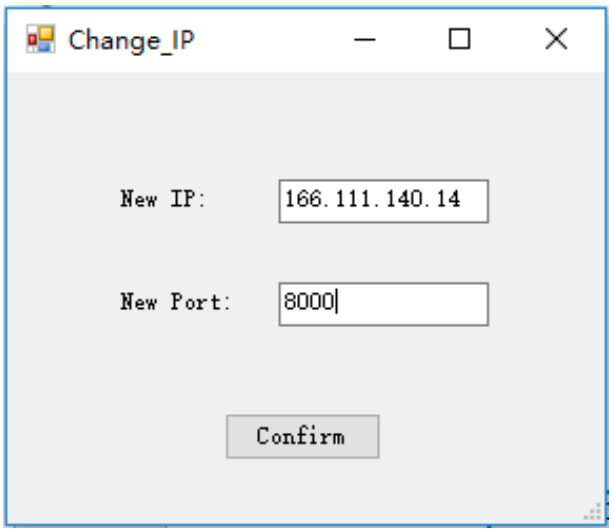


图 8: 修改 Ip 和端口

初步判断合法性之后，通过 TCP 与服务器建立连接，将用户名与密码组合发送给服务器。若连接失败，弹出服务器繁忙信息，若连接成功，获取服务器的返回值，并加以判断，如果返回值为”lol” 则确认登录，进入主界面。

4.2 账号退出

账号退出在主界面中通过点击右上角的 log out 标志实现。

图 9: 账号退出

点击退出键后，弹出对话框，要求用户确认是否退出。

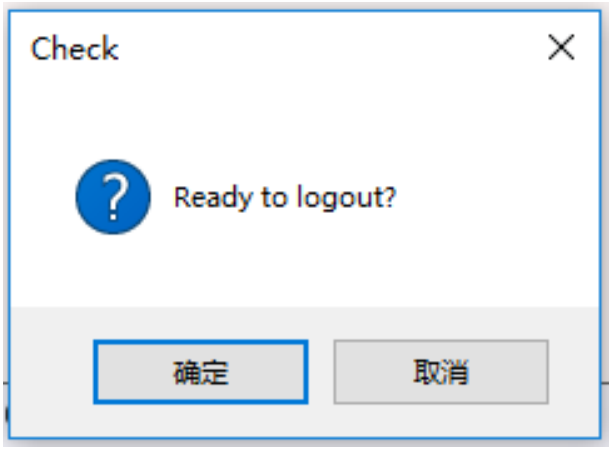


图 10: 账号退出确认

用户确认后，通过 TCP 与服务器建立连接，发送 logout+ 学号至服务器，接受服务器返回”loo” 则下线成功，程序退出，若出现其他异常，则返回对应提示信息。

4.3 查询好友

查询好友通过点击左上角的文本框，输入好友的学号，点击右边的搜索键进行搜索。

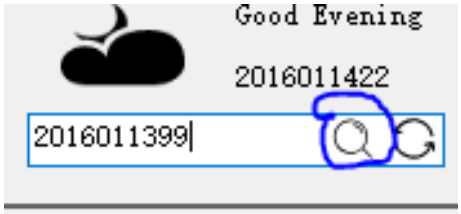


图 11: 查询好友在线状态

再次通过建立 TCP 与服务器连接，发送 q+ 查询学号至服务器，若服务器返回”n” 则说明好友不在线，无法进行通讯，若返回 IP 地址，则说明好友在线，将此 IP 地址进行存储，以便后续通信时使用。好友不在线则弹出对应提示，若好友在线则询问是否开始聊天。

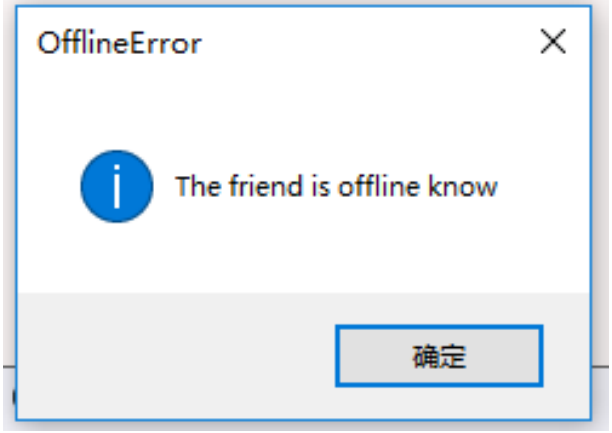


图 12: 查询好友，好友下线时提示

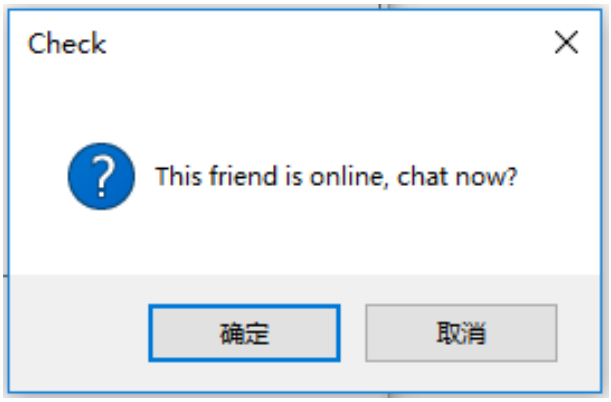


图 13: 查询好友，好友在线，询问是否发起聊天

4.4 点对点通讯

用户查询好友之后决定发起聊天，双方即可开始点对点通讯，通讯过程主要分为发送和接受两个部分，以下进行原理说明。（注：这里限制了用户给自己发送消息，因为这样做的意义不大）

双方握手流程 首先在这里解释双方建立通讯的整个流程，这个流程在所有传输过程中适用。在一方用户 A 打算向另一方用户 B 发起通讯/建立通讯后发送文本或文件时，首先将自己的学号信息进行打包发送，确认 B 用户目前的状态，B 用户若处于空闲，即暂未和别人聊天中，或 B 用户之前正在和 A 聊天，只是聊天继续，那么即返回”Hallo” 信息，同时存下 A 用户的账户信息，更新界面。A 用户收到”Hallo” 信息之后，才正式开始信息传送。这是确保了 B 用户中途退出与 C 聊天，但 A 不知情的情况下，BC 聊天的稳定性。

信息的接收通过线程监听 + 异步触发来实现。由于接收发送的端口号是固定的，

所以接收方只需监听所有连接到固定端口号的 TCP 连接即可。一旦监听到 TCP 连接，打开一个新的线程进行信息的处理，同时继续保持监听状态。

第三方介入聊天 首先我们看一下当两方（2016011422 与 2016011399）已经发起聊天时，第三用户 2016011413 再尝试向 2016011422 发起聊天，在 1413 账户同学界面就会出现如下错误：

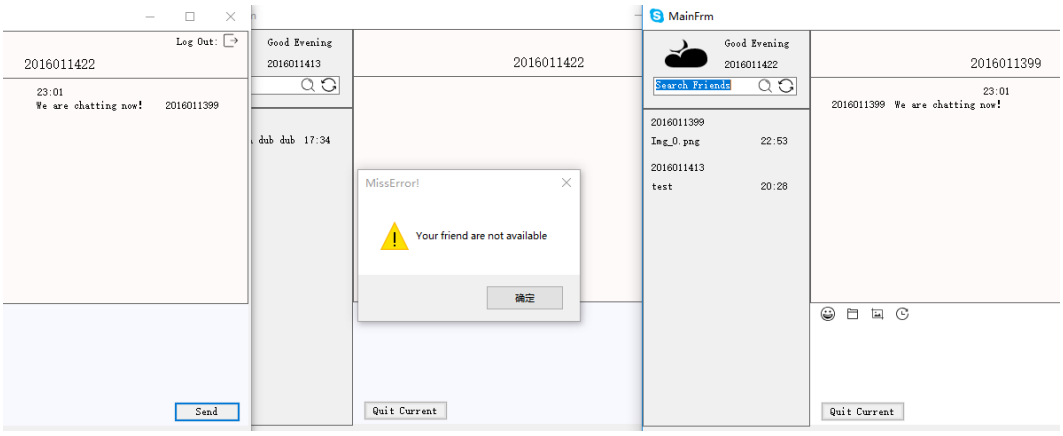


图 14: 双方聊天时第三方发送的情况

通过这样的方式，保证了已连接的两者稳定通讯，当 2016011422 退出当前聊天之后回到空闲状态，此时第三方用户即可发起连接。

文本框内容发送与接收 在客户端的文本框中可能同时存在图片与文字两种内容，其中图片由截图部分功能产生，文字为用户输入的，关于截图部分功能会在后面进行阐述。对于图片信息，点击发送时，自动将其保存在本地，后以文件形式发送。对于文本信息，在其头上加上”/**Text**/” 的头项，便于接收方进行各类信息的区分。在具体实现中，当信息混合时，优先发送文本信息，再发送图片信息。以下为一个例子：

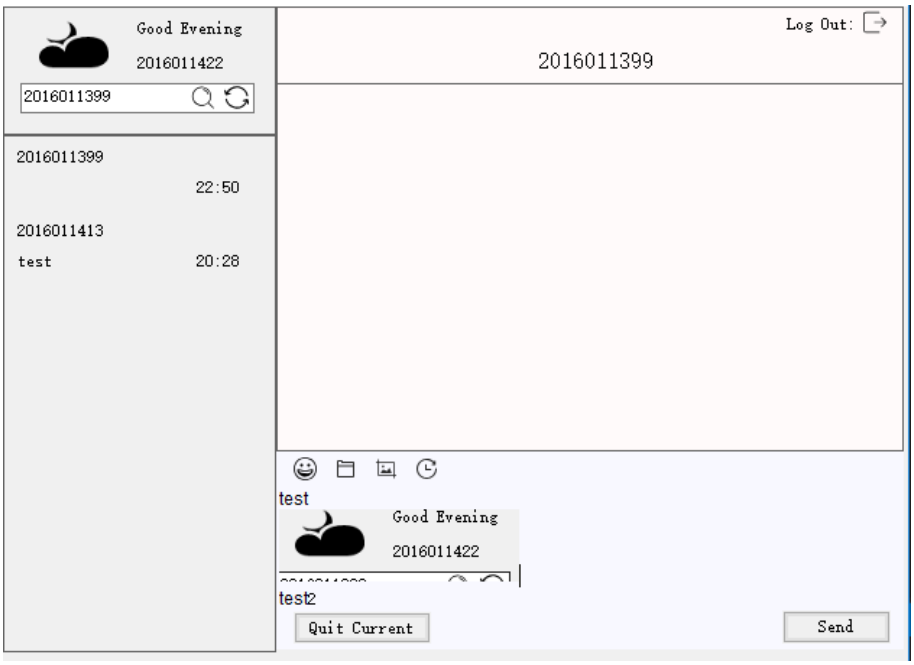


图 15: 图片文本混合发送前

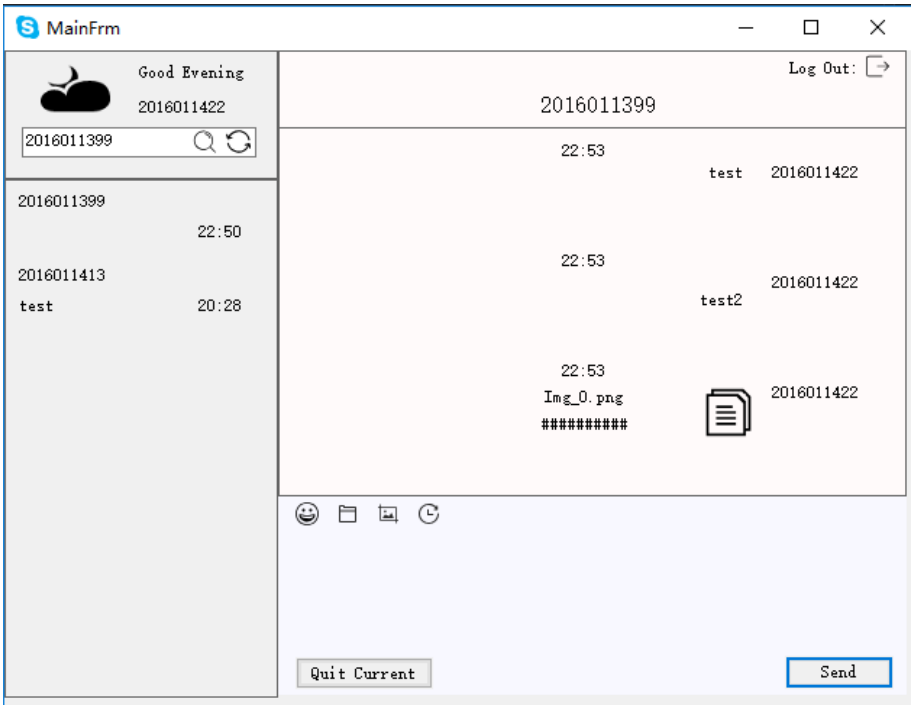


图 16: 图片文本混合发送结果

若聊天中对方下线，则同样会报错，因为我在每次发送前都会判断对方当前是否可到达。

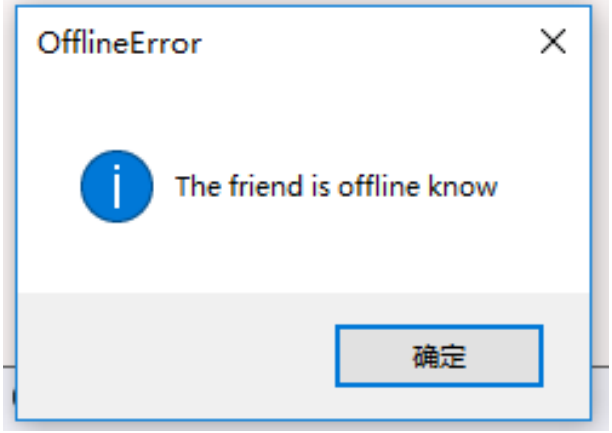
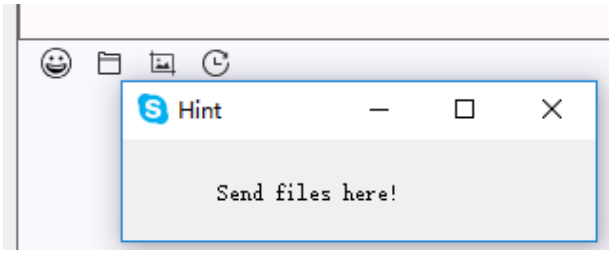


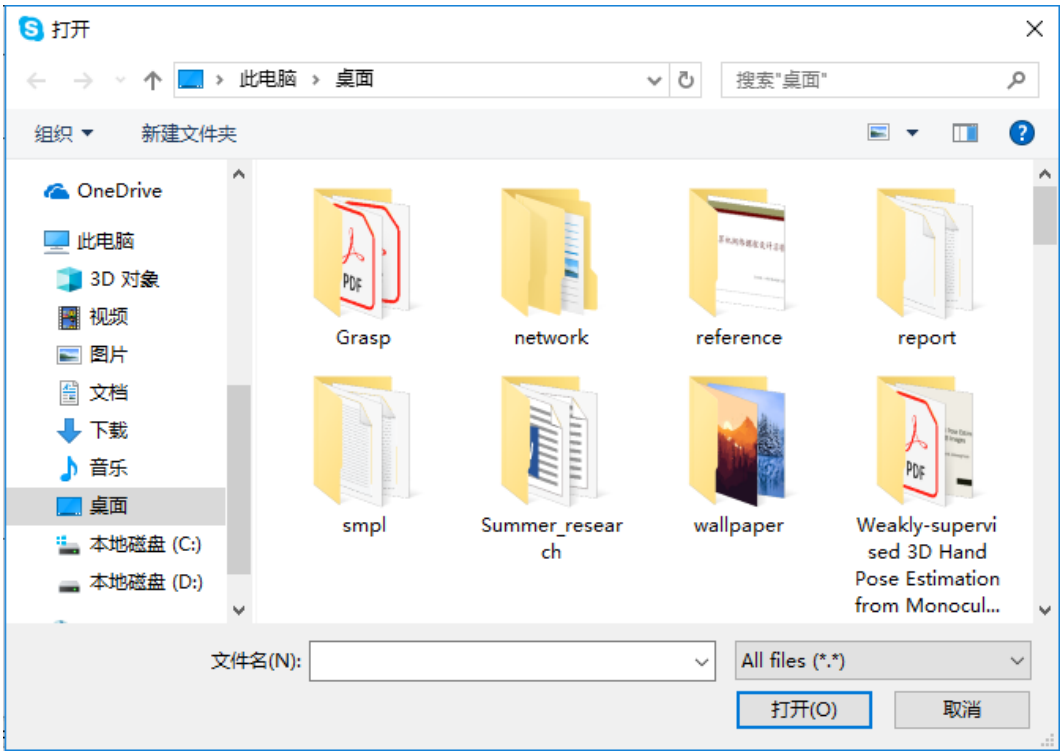
图 17: 聊天过程中对方下线，则无法继续发送

对接收方而言，接收到对方传递过来的信息之后，查看 NetworkStream 流的前几位信息，如果与”/**Text**/” 相符合，则确定是文本信息，交给主线程在聊天界面上显示相应的控件。

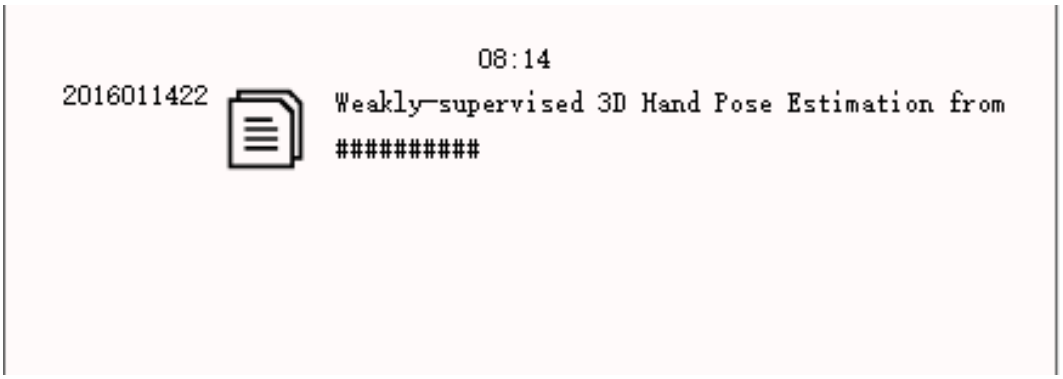
文件发送与接收 点击界面上的 UI 交互即可发送文件。



点击后自动弹出系统文件选择的窗口，进行任意文件的选择。



点击确认后即可发送文件，发送文件效果如下：

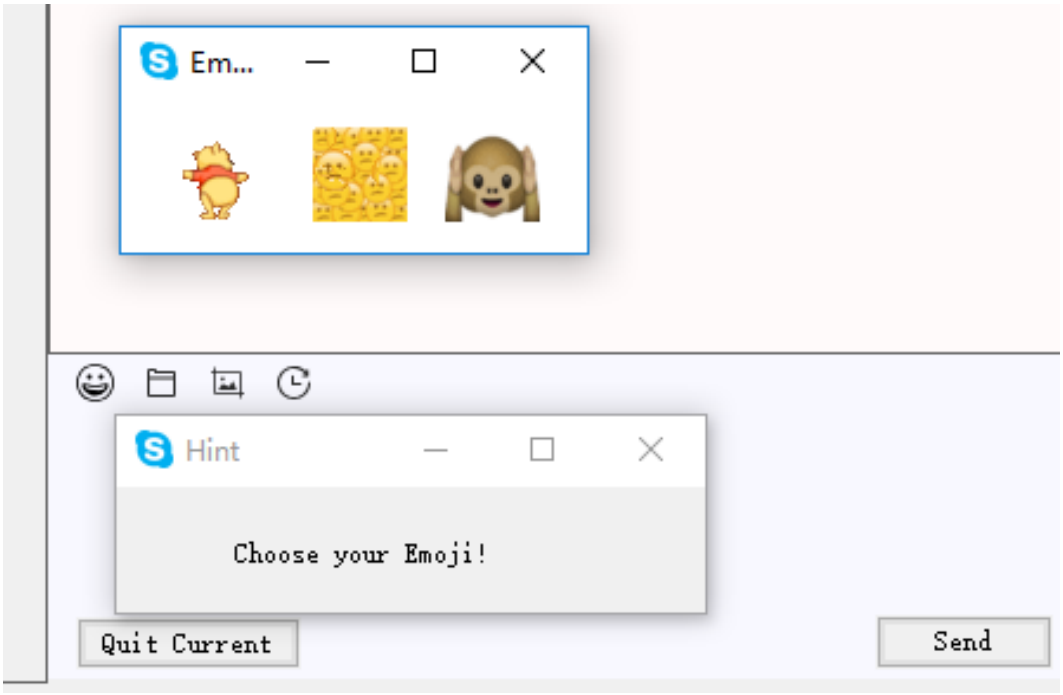


显示了发送人，发送文件名，一个固定图标，以及实时更新的发送进度信息。文件发送基于 FileStream，从本地读取文件数据，再基于 TCP+NetworkStream 流实现网络传送，并单独开了一个线程进行处理，以防主程序卡死。我将文件分段，每次发送 1Kb，一方面确保了传输可靠性，可以捕捉到用户在途中退出的意外情况，另一方面也有助于接收方和更新方分别同步显示文件发送/接收的进度。在文件传送开头，我加上了前缀"/**begin-file-transport**/"，保证接收方解析到这样的开头时调用文件接收处理的函数处理。为了让对方能够知晓文件传输的进度，则需要其在文件传输结束前即可知道文件的大小，且由于文件名的信息没有包含在文件的内容中，需要进行单独发送，所以我在文件前缀"/**begin-file-transport**/" 之后加上"/*FileName*/" 和文件

名，”/*FileLength*/” 和文件大小（以 Kb 计）。

对接收方而言，解析到传输信息开头为约定前缀后，调用文件处理的函数进行处理，先根据约定信息获得文件名与文件长度，根据文件名在本地新建文件，如果文件名重复，则根据顺序分别加上 (1),(2)... 等，保证传输鲁棒性。接着利用 NetworkStream 进行接收，利用 FileStream 进行文件写，且每接收 10% 的数据更新一次控件，显示当前进度，因为我的进度显示是以十位”#” 来实现的，即每 10% 加上一个”#”。

表情发送与接收 点击 UI 上的图标即可弹出表情选择窗口，用户点击对应表情即可发送。



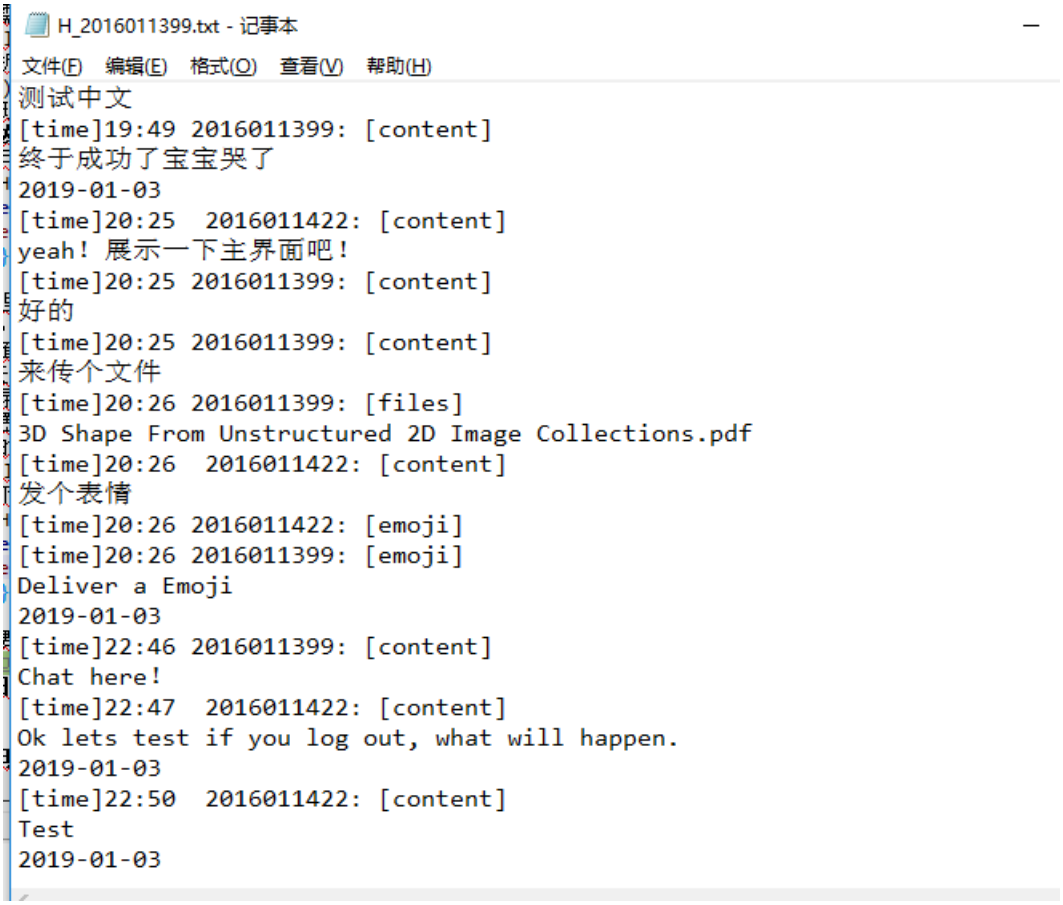
动态表情的发送原理与文本发送类似，只是显示界面有所不同。在这里我使用了比较取巧的方法，即保证交互双方本地的表情包相同，那样在传送表情时，先根据用户点击解析出用户选择的表情的索引，将此索引加上”/**Emoji**/” 的文件头后，以发送文本的方式进行发送。接收方在解析到约定好的信息流开头之后，即可调用专门的程序获取表情索引，调用本地的表情库，在主界面上显示相应的控件。通过这样的方式，可以比较好地基于现有的文本传输程序，做一个有效的扩展。

4.5 聊天记录保存与查询

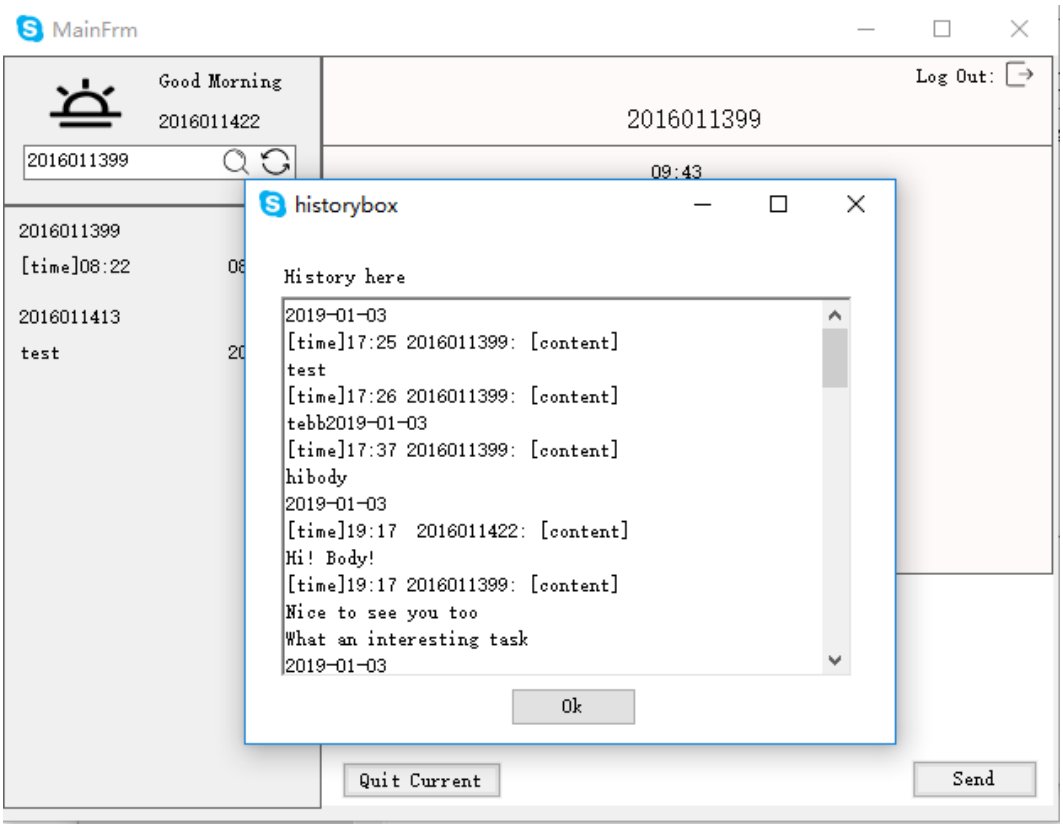
首先说一下聊天记录的保存，由于我的聊天都是基于单次对话的，所以在每次点击左下角”quit current” 按键退出当前对话，或者直接退出当前账号时，都会将本次聊天窗口内的所有记录保存到对应位置的 txt 里。路径为发送方自己的文件夹，文件名则根据聊天方来生成”H_ 聊天方”。每次保存信息时，都会先储存当天日期，再进行内容的

储存。由于 txt 只能存储纯文本的信息，故我的聊天记录格式为 [time]00:00 [发送方账号]: [Emoji/Content/Files]，再根据具体发送的信息存储，如果是 Emoji，则只保存信息”Deliver a Emoji”，如果是 Files，则保存文件名，如果是文本信息，则保存发送的内容。每次将信息追加到之前的聊天记录之后。

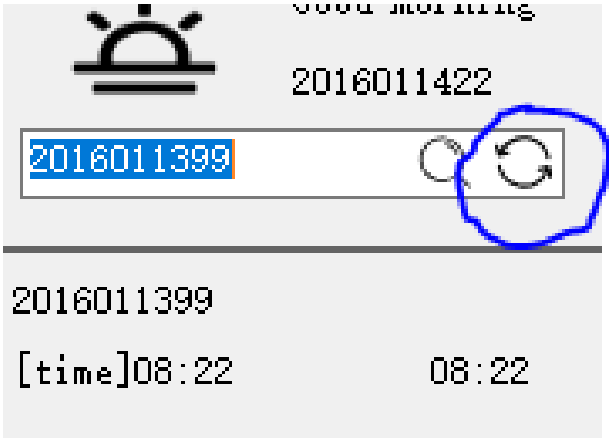
储存的文本内容如下：



聊天记录的查询主要分为两块，一部分是查询与单用户的聊天记录，另一部分是显示最近聊天过的好友列表以及对应的双方最后一条消息与发送时间（基于本地聊天记录）。效果如下：



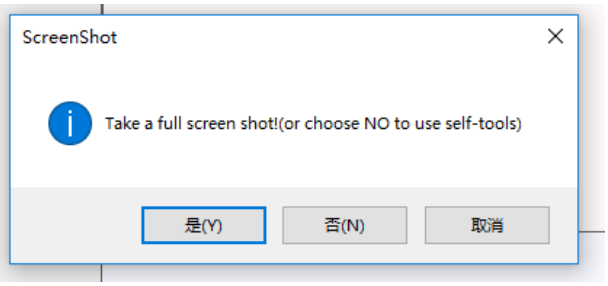
其中与单用户的聊天记录查询，通过点击交互界面上的历史按钮实现，点击之后，读取本地对应的 txt 文件，将其中内容显示在 richtextbox 中，若与对方没有聊天记录（即不存在本地文件）则弹出相应提示。而最近的好友列表则读取本地每一个 txt 文件的文件名获取好友账号，并读取最后一个发送时间与最后一条消息内容，形成对应控件，存放在左侧的容器中进行显示。这个列表的刷新并不是实时的，在每次登陆时会进行刷新，此外用户可以点击相应按钮进行手动更新。



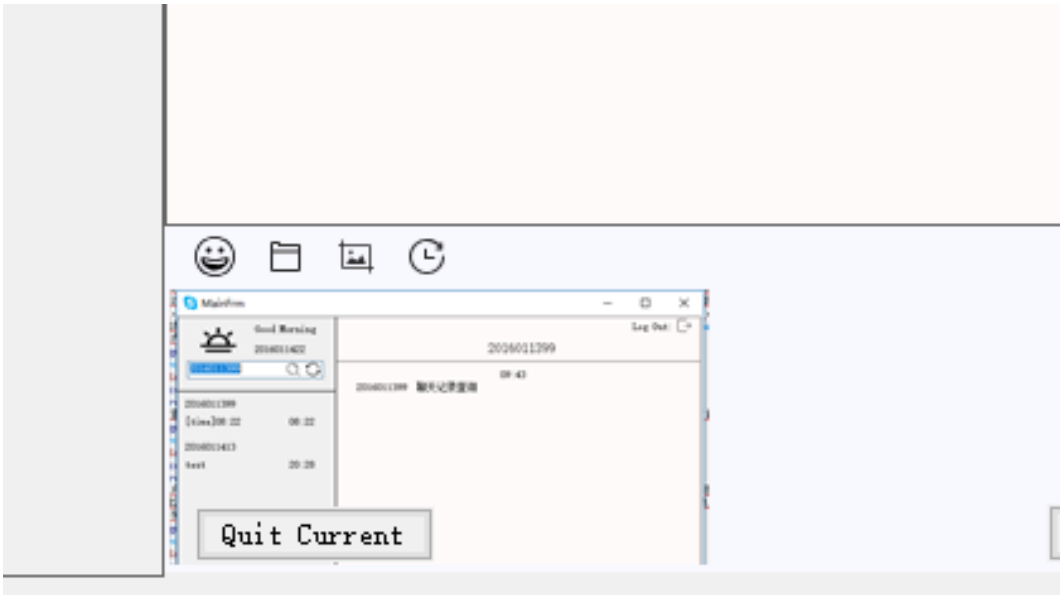
4.6 截图功能实现

我主要通过两个方式实现截图功能，一是自动截图，即利用 C# 自带的截图工具，对当前界面进行截图，这样的方式较为简单，但不能有其他更多的变化，二是利用系统自带的截图，截完图后可以将其直接复制进发送文本框中，点击发送即可，所以这样才会出现图片与文本在文本框中混合的情况，混合发送的解决方案已经在之前说明了。

点击截图 UI 按键之后弹出提示窗口：



点击是则进行全客户端的自动截图，点击否则可以通过系统自带的截图来实现，截图完后自动放入文本框中。



主要实现原理为利用了剪切板这一中介，在程序内实现对系统剪切板的侦听，如果选择了截图功能，且剪切板内容发生变化，则富文本框自动将其复制到自己的内容中。那么如何将截图图片复制到剪切板呢？使用系统自带截图工具比较简单，有自带的复制工具，如果使用的是自动的截图，则在程序内将其复制到剪切板中，并将其缩小为 1/3 之后显示在文本框中，因为要不然显示整个客户端实在是太大了！

至于图片的发送则基于文件传送的方式，在 Para.4.4中已经阐明。

5 实验总结

这次大作业，我们利用计网所学的知识完成了基于中央服务器的 P2P 通讯客户端的编写，迫于时间压力，我未能完成群聊部分，深觉遗憾。其实网络通讯部分并不是特别复杂，我们只接触到了应用层 + 传输层两个方面，对底层的传递的稳定性保证都是基于上层的异常处理来实现的，所以加了很多异常的判断，服务器连接失败/用户退出/自我发送/文件重名/表情导入失败等等，但是还有很多用户意想不到的使用很可能会导致一些意外的 bug，这样一想补丁实在太重要了。我的整个架构都想模仿微信来进行设计，发现只能实现它的一部分功能... 微信还是做的非常好的，要实现一些看起来简单的功能，需要数以百计的头发与心血。程序的鲁棒性/功能完善/界面交互友善提升都还有很多事情可以做，但是至少我发现通过这一个学期的学习和数个不眠夜晚的学习，我也能写一个简简简易版的微信了！这一点让我非常感激和激动，道阻且长，的确还有很多地方值得探索和努力，也感谢迈出的第一步。