P2P Chat Application (P2PC)
System Functional Specification

CMP2204 Term Project Spring 2019

1 Introduction

1.1 System Purpose

The purpose of Service Discovery for our P2P Chat Application (P2PC) is to automatically detect all available users in the network for the chat host to initiate a conversation. The purpose of Chat Client and Server are to exchange messages between two processes on two remote hosts.

1.2 Definitions, Acronyms, and Abbreviations

Throughout this document, the terms in boldface below are to be interpreted as defined:

shall	This term indicates an obligatory requirement that must be met to comply with the specification.
may	This term indicates an item that is truly optional.

1.3 Operational Scenarios

The following are the use cases supported by the P2PC:

Logging-in: Upon connecting to the Local Area Network, P2PC starts listening for all P2PC services in the LAN. Each detected user is stored in a dictionary. The end user is able to display the list of online users.

Sending a message: User can view all available users in the network. (Implementation details for this functionality are to be figured out by the developer.) The end user specifies one username to chat with, a TCP session is opened with the corresponding IP address, and the user-typed message is encrypted and sent over this TCP connection. After this, TCP session is closed.

Receiving a message: When a new TCP connection request is received, Chat_Listener immediately accepts this connection request and receives, decrypts and displays the message. The sender name must be displayed with the message, so that the end user can figure out whom the message is from.

Message history: User can view the message history (information related to chatted user, date/time, and messages).

2 Requirements

2.1 Service Discovery Requirements

Req. #	Requirement
2.1.0-A	P2PC Service Listener **shall** listen for UDP broadcast messages on port 5000. The broadcast IP address should be configured programmatically, by replacing the last (4^{th}) decimal number with 255 in the host IP address. For example, if your IP address is 192.168.2.34, you should broadcast to IP address 192.168.2.255.
2.1.0-B	Upon receiving a message, P2PC Service Listener **shall**: *(i)* parse the message contents using a JSON parser in Python, *(ii)* get the UDP broadcast sender's IP address using recvfrom() method. The *username* from the JSON message, and the fetched *IP address* **shall** be stored in a dictionary. The dictionary **may** be a local text file that can be shared between the Chat Client and Service Listener components.
2.1.0-C	When launched, P2PC Service Announcer **shall** first ask the user for its username, and store it locally.
2.1.0-D	P2PC Service Announcer **shall** periodically send broadcast UDP messages announcing its service. The period **shall** be once per minute.
2.1.0-E	P2PC Service Announcer's periodic broadcasts **shall** contain a JSON that looks like: '{"username": "Ece", "ip_address":"192.168.1.15"}'. Service Announcer **shall** be able to retrieve its host's IP address programmatically, to insert into this message.
2.1.0-F	P2PC Service Listener **may** display each detected user on the Terminal (*e.g.,* "Ece is online"), which would provide better user experience.

2.2 Chatting Requirements

Req. #	Requirement
2.2.0-A	P2PC Client **shall** display a list of online users' names. The flow **may** depend on developer's choice.
2.2.0-B	P2PC Client **shall** provide a mechanism for the user to send a message to any online user.
2.2.0-C	P2PC Client **shall** initiate a TCP session with the specified user's IP address.
2.2.0-D	P2PC Server **shall** listen for TCP connections on port 5001.
2.2.0-E	P2PC Server **shall** accept TCP connection request before it times out, and **shall** successfully receive the message sent from the end process.
2.2.0-F	P2PC Server **shall** display the received message and its sender. P2PC server **shall** display the sender name instead of the IP address. (This may require you to maintain two dictionaries for IP and name lookup, respectively.)
2.2.0-G	P2PC Server **shall** dump all received messages in a Chat log (a text file) under the same directory.
2.2.0-H	P2PC Client **shall** close a TCP session upon sending the message.
2.2.0-I	After a TCP session is closed, P2PC Client **shall** persist; the Chat Client process **shall not** terminate.
2.2.0-J	P2PC Server and Client **may** be implemented as different threads in the same process. In that case, end user may both write messages to send, and view messages received, in one (same) Terminal window.

2.3 Performance Requirements

Req. #	Requirement
2.3.0-A	P2PC **shall** run on Python 3.
2.3.0-B	P2PC **may** spawn different threads for Service Listener, Service Announcer, Chat Server and Chat Client. In that case, it **may** operate on a single Terminal window, providing easier use.
2.3.0-C	Service Listener must be able to detect and display up to 50 users.
2.3.0-D	The Chat application **shall** be able to send to and receive from any user, with no perceivable delay.
2.3.0-E	The P2PC application **may** be operated in 4 Terminal windows. Alternatively, if you are developing a multi-threaded application one Terminal window may be sufficient. In the latter case, you **shall** provide guidelines to user for which key to press for which action.
2.3.0-F	Any unspecified configuration is a plus – setting busy status, displaying error message when message can't be delivered to user, etc.