Assignment 1 - P2P Network

Group 10: Eduardo Castellanos - Thanh Bui

I. Introduction

In this assignment, we have developed a p2p application based on the protocol described in the assignment requirement. The application has all of the basic and advanced functions described in the assignment requirement:

- Send Join Request message in order to join the network.
- Handle Join Request messages.
- Send Query message with a search key and handle Query Hit messages.
- Publish a key/value pair in the network and respond to a Query message when its search key equals to the published key.
- Join multiple peers to expand the network.
- Send Type A Ping message and handle Type A Pong message to check if our neighbours are alive.
- Respond to Type A Ping message with a Type A Pong message.
- Send Type B Ping message and handle Type B Pong message to find more node information in the network.
- Respond to Type B Ping message with a Type B Pong message to tell our neighbour who else we know.
- Forward Query messages with loop avoidance to help other node to find a resource.
- Forward Query Hit message by following the reverse path of its corresponding Query message.

We also created a GUI for easy interaction between users and the application.

II. Our Solution

This section presents how we designed and implemented our program as well as how we handled each type of messages.

2.1. Technologies

We used Python 2.7 with the built-in libraries asyncore, and struct library to implement this assignment. In addition, PyQt4 was used to implement the GUI.

2.2. Application design

The class diagram of our application is shown in Figure 1. We have six types of message classes, which are derived from the base Message class: the ByeMessage, which represents the Bye message, the PingMessage/PongMessage which represent Ping/Pong message respectively, the JoinMessage, which represents the Join Request and Join Response message, and the QueryMessage/QueryHitMessage which represent Query and Query Hit message respectively.

The main handler of our application is P2PMain class which establishes a P2PListener instance accepting connections from other peers. For each connection, the P2PMain instance creates a P2PConnection instance to handle the connection. These P2PConnection instances use the message classes to handle incoming and outgoing messages.

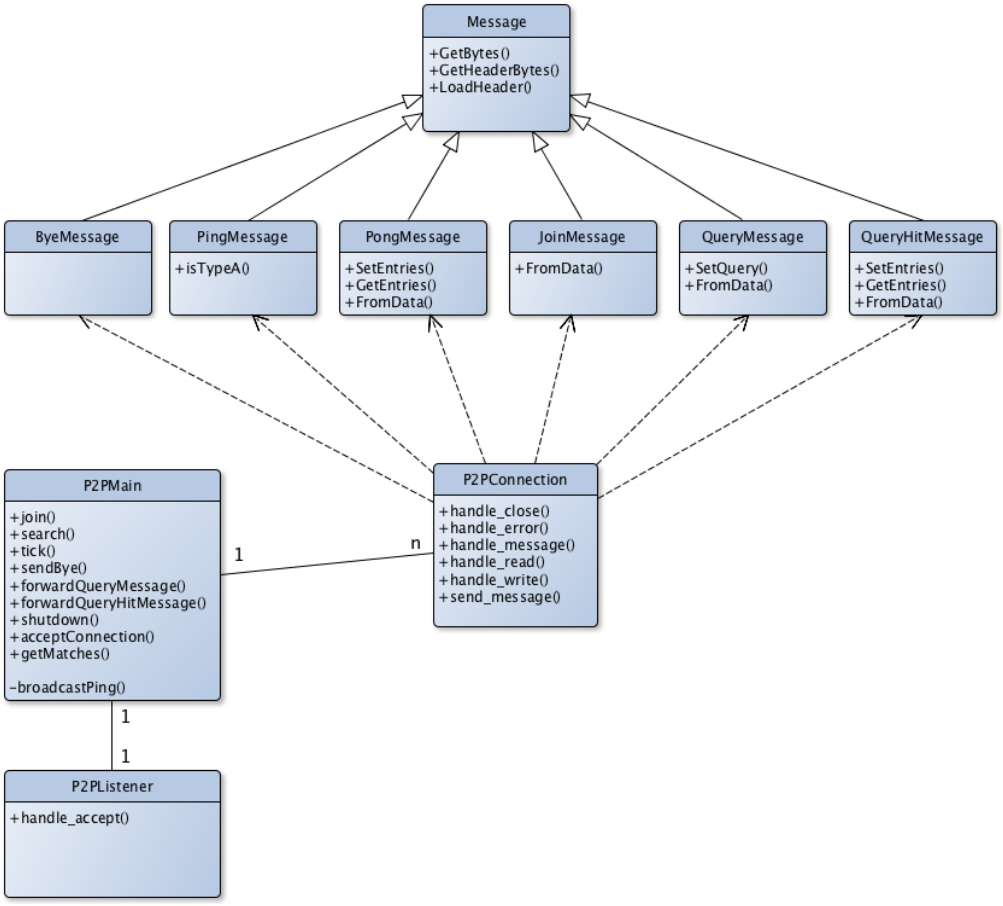

Figure 1 - Class diagram

2.3. Messages Handling

In this section, we describe how we handled each type of messages, including: Join message, Type A Ping/Pong message, Type B Ping/Pong message, and Query message.

Join message

When our application is started, it sends a Join Request message to the bootstrap to join the network. The connection is considered active only when the application has received the Join Response message from the bootstrap.

When our application receives a Join Request message from another peer, it opens a connection to send/receive packets to/from the peer and sends back a Join Response message to the peer.

We also store a list of peers in the application, so we can easily broadcast a message.

Type A Ping/Pong

Our application periodically sends a type A Ping message to its neighbours every 5 seconds to tell them that it is still alive.

It also responds to type A Ping messages with type B Pong messages. For the neighbours that are not updating their status by sending Ping messages, the application can detect but still keep the connection with them since they might not support Ping/Pong messages.

Type B Ping/Pong

Our application periodically sends a type B Ping message to its neighbours every 10 seconds to find more peers. If it receives a type B Pong message containing information about another node's peers, it extracts the information and sends Join Request messages to the peers to connect to them.

When our application receives a type B Ping message, it will randomly select 5 of its neighbours and sends the their information back to the sender in form of a type B Pong message.

Query message

Our application supports key searching with Query messages. If users want to search for a key, the application will wrap it in a Query message and then broadcasts it to all of its neighbours. In order to prevent query flooding, all of these Query message have the same message ID.

If the application receives a Query message, it will first stores the message ID and the information about the peer that sent the message. Then it checks for its local database to find the matching key and returns the result if found. It also broadcasts the Query message to all of its neighbours, except the peer that sent the message.

If the application receives a Query Hit message, it will look for the peer that sent the Query message with the same message ID in its database. If it finds a match, then there are two possibilities: (1) if it is the one that started the searching, it will display the result from the Query

Hit message, otherwise (2) it will forward the Query Hit message back to the peer that it received the corresponding Query message from.

III. Conclusion

In this assignment, we have developed a simple P2P application that uses a Gnutella-like protocol. Through this assignment, we learnt about the behaviour of a P2P application and how Gnutella works. Technically, we also learnt how to use Python to create client-server application and how to construct TCP packets.