Power Peering
A peer to peer network as a Smtp/Pop3 overlay.

Omar Moling, Anton Dignös
Free University of Bozen

This report is about the Power Peering application, developed for the course Computer Networks, held by Prof. Gabriele Gianini, 2008/2009.

1 Introduction

Power Peering is an application aimed to be a peer to peer network application on top of the Smtp and pop3 protocol. Its main features are the one of a common P2P system, as joining the network, querying resources, requesting resources and offering such.

Common P2P applications are based on level 4 protocols as TCP/IP directly, thous they require special ports of the client to be open, to work. This application instead uses protocols part of the application layer, to communicate, this protocols are Smtp for sending and Pop3 for receiving messages.

Although it seems to be an overhead to use those message (plain/text) oriented protocols, its usage has a main advantage. The required ports of this protocols are already open for mail messaging use and mails are even protected by the privacy law.

The rest of this report is organized as follows ...

2 Application Overview

As already introduced the Power Peering application has as a requirement all needed functionalities of a P2P system. Figure 1 shows the use case diagram of the application, including all needed functionalities for the user to work in an abstract way. For a better understanding of the systems internal functionality, sequence diagrams for the most important requirements have been developed. Figure 2(a) shows the systems sequence by invoking a network discovery, which is realized by the internal PING message send to adjacent clients. These clients forward the the message of the requesting client and return a PONG message to it. Figure 2(b) shows instead the sequence of actions performed by the system when a QUERY of a user is performed. The QUERY is as the PING send

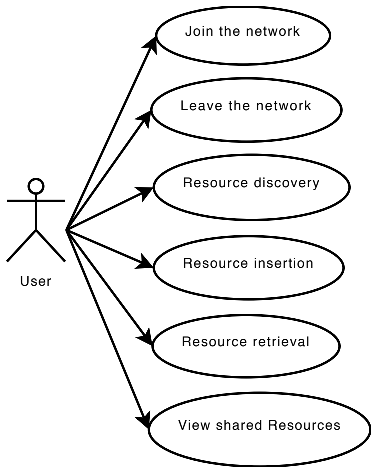
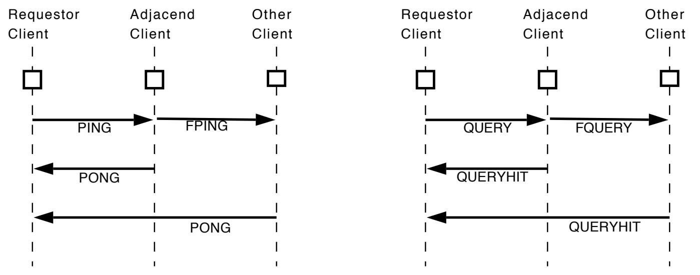


Figure 1: Use Case Diagram.

to adjacent/intermediate clients, which forward the message to their adjacent client. The response instead is always returned directly to the requestor.



(a) Network joining. (b) Resource discovery

Figure 2: Sequence Diagrams.

The term adjacent client at this point needs an explanation, clients do not store all other clients of the network and they only keep a subset of them. This requires a mechanism for forwarding messages as a client can not communicate with all others. These subset of clients to store is evaluated randomly to avoid partitioning of the clients, on a PING or PONG request of the network discovery a client evaluates randomly whether to keep the reference to this client or not when its client list is already full.

As a client which forwards a message does not know, which of its adjacent client already got it, messages which need forward contain a field time to live (TTL) which is decreased by forwarding. A message is not forwarded any more when its TTL expires.

3 Application Architecture and Design

The architecture of the system is divided into 4 packages, reflecting the components of the application.

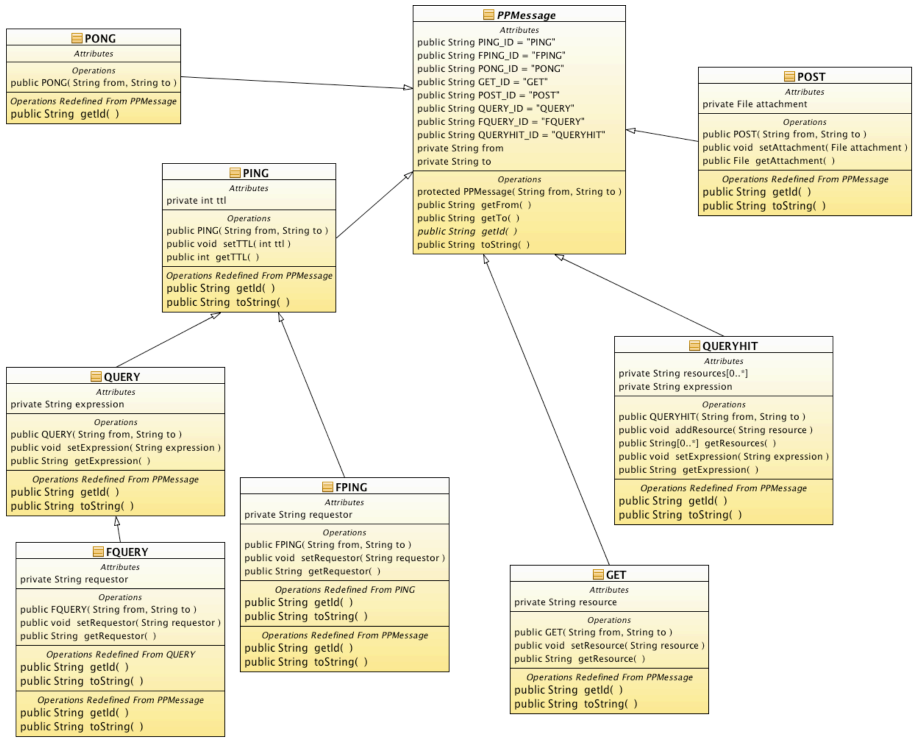

PPMessage
Attributes
public String PING_ID = "PING"
public String FPING_ID = "FPING"
public String PONG_ID = "PONG"
public String GET_ID = "GET"
public String POST_ID = "POST"
public String QUERY_ID = "QUERY"
public String FQUERY_ID = "FQUERY"
public String QUERYHIT_ID = "QUERYHIT"
private String from
private String to
Operations
protected PPMessage(String from, String to)
public String getFrom( )
public String getTo( )
public String getId( )
public String toString( )
PONG
Attributes
Operations
public PONG(String from, String to)
Operations Redefined From PPMessage
public String getId( )
POST
Attributes
private File attachment
Operations
public POST(String from, String to)
public void setAttachment(File attachment)
public File getAttachment( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )
PING
Attributes
private int ttl
Operations
public PING(String from, String to)
public void setTTL(int ttl)
public int getTTL( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )
QUERYHIT
Attributes
private String resources[0..*]
private String expression
Operations
public QUERYHIT(String from, String to)
public void addResource(String resource)
public String[0..*] getResources( )
public void setExpression(String expression)
public String getExpression( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )
QUERY
Attributes
private String expression
Operations
public QUERY(String from, String to)
public void setExpression(String expression)
public String getExpression( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )
FPING
Attributes
private String requestor
Operations
public FPING(String from, String to)
public void setRequestor(String requestor)
public String getRequestor( )
Operations Redefined From PING
public String getId( )
public String toString( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )
FQUERY
Attributes
private String requestor
Operations
public FQUERY(String from, String to)
public void setRequestor(String requestor)
public String getRequestor( )
Operations Redefined From QUERY
public String getId( )
public String toString( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )
GET
Attributes
private String resource
Operations
public GET(String from, String to)
public void setResource(String resource)
public String getResource( )
Operations Redefined From PPMessage
public String getId( )
public String toString( )

Figure 3: Message Class Diagram.

Figure 5 shows the class diagram of the message package, these classes are used as data structures for the messages the system ins capable of sending and/or receiving.

Figure 4 are the classes of the io package and are responsible for the abstraction of the protocol overlay. Outside these package classes can use the PPMessages class to send and parse messages and the POP3 class to fetch messages, the MessageHandler Interface is used by the POP3 class witch's method is called for each message of the mail box. These allows the system to only pop messages which are of its interest, others are left as they are on the server.

Figure 5 shows the implementation of the actual client, and the main entry point of the application. A client can be implemented and extended by simply extending the PowerPeer class, and extending some of its methods which are of its interest. The UserInterface interface acts as a common point for communication with the users output.

The class diagram shown in Figure 6 illustrates the commands used by the

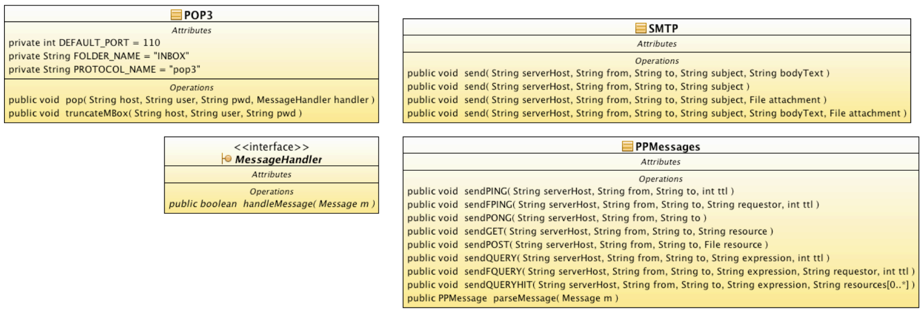

POP3
Attributes
private int DEFAULT_PORT = 110
private String FOLDER_NAME = "INBOX"
private String PROTOCOL_NAME = "pop3"
Operations
public void pop(String host, String user, String pwd, MessageHandler handler)
public void truncateMBox(String host, String user, String pwd)
SMTP
Attributes
Operations
public void send(String serverHost, String from, String to, String subject, String bodyText)
public void send(String serverHost, String from, String to, String subject)
public void send(String serverHost, String from, String to, String subject, File attachment)
public void send(String serverHost, String from, String to, String subject, String bodyText, File attachment)
<<interface>>
MessageHandler
Attributes
Operations
public boolean handleMessage(Message m)
PPMessages
Attributes
Operations
public void sendPING(String serverHost, String from, String to, int ttl)
public void sendFPING(String serverHost, String from, String to, String requestor, int ttl)
public void sendPONG(String serverHost, String from, String to)
public void sendGET(String serverHost, String from, String to, String resource)
public void sendPOST(String serverHost, String from, String to, File resource)
public void sendQUERY(String serverHost, String from, String to, String expression, int ttl)
public void sendFQUERY(String serverHost, String from, String to, String expression, String requestor, int ttl)
public void sendQUERYHIT(String serverHost, String from, String to, String expression, String resources[0..*])
public PPMessage parseMessage(Message m)

Figure 4: IO Clas Diagram.

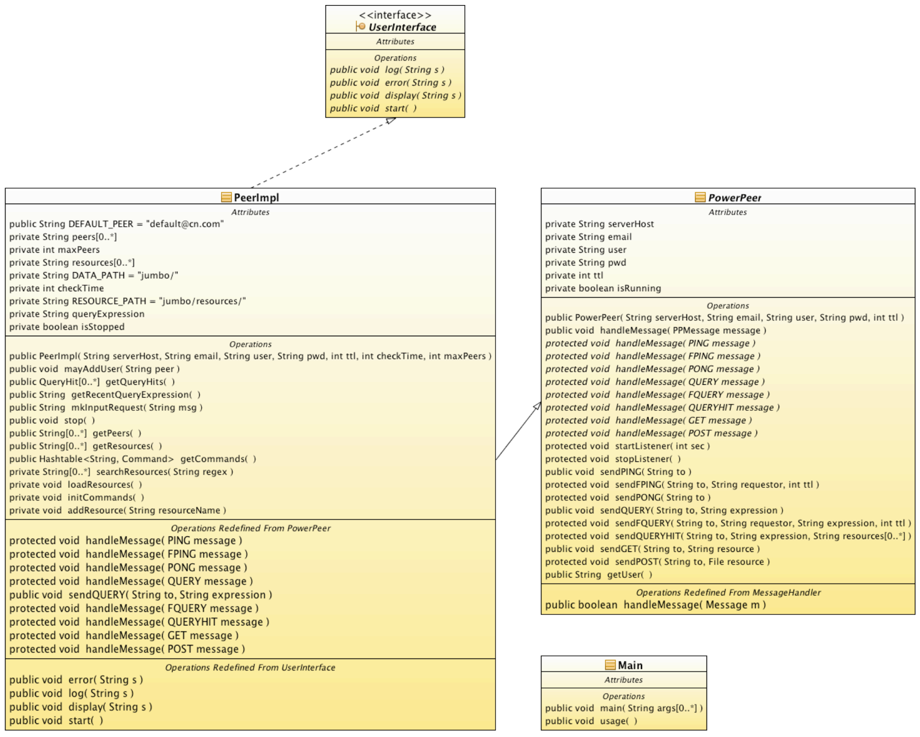

<<interface>>
UserInterface
Attributes
Operations
public void log(String s)
public void error(String s)
public void display(String s)
public void start( )
PeerImpl
Attributes
public String DEFAULT_PEER = "default@cn.com"
private String peers[0..*]
private int maxPeers
private String resources[0..*]
private String DATA_PATH = "jumbo/"
private int checkTime
private String RESOURCE_PATH = "jumbo/resources/"
private String queryExpression
private boolean isStopped
Operations
public PeerImpl(String serverHost, String email, String user, String pwd, int ttl, int checkTime, int maxPeers)
public void mayAddUser(String peer)
public QueryHit[0..*] getQueryHits( )
public String getRecentQueryExpression( )
public String mkInputRequest(String msg)
public void stop( )
public String[0..*] getPeers( )
public String[0..*] getResources( )
public Hashtable<String, Command> getCommands( )
private String[0..*] searchResources(String regex)
private void loadResources( )
private void initCommands( )
private void addResource(String resourceName)
Operations Redefined From PowerPeer
protected void handleMessage(PING message)
protected void handleMessage(FPING message)
protected void handleMessage(PONG message)
protected void handleMessage(QUERY message)
public void sendQUERY(String to, String expression)
protected void handleMessage(FQUERY message)
protected void handleMessage(QUERYHIT message)
protected void handleMessage(GET message)
protected void handleMessage(POST message)
Operations Redefined From UserInterface
public void error(String s)
public void log(String s)
public void display(String s)
public void start( )
PowerPeer
Attributes
private String serverHost
private String email
private String user
private String pwd
private int ttl
private boolean isRunning
Operations
public PowerPeer(String serverHost, String email, String user, String pwd, int ttl)
public void handleMessage(PPMessage message)
protected void handleMessage(PING message)
protected void handleMessage(FPING message)
protected void handleMessage(PONG message)
protected void handleMessage(QUERY message)
protected void handleMessage(FQUERY message)
protected void handleMessage(QUERYHIT message)
protected void handleMessage(GET message)
protected void handleMessage(POST message)
protected void startListener(int sec)
protected void stopListener( )
public void sendPING(String to)
protected void sendFPING(String to, String requestor, int ttl)
protected void sendPONG(String to)
public void sendQUERY(String to, String expression)
protected void sendFQUERY(String to, String requestor, String expression, int ttl)
protected void sendQUERYHIT(String to, String expression, String resources[0..*])
public void sendGET(String to, String resource)
protected void sendPOST(String to, File resource)
public String getUser( )
Operations Redefined From MessageHandler
public boolean handleMessage(Message m)
Main
Attributes
Operations
public void main(String args[0..*])
public void usage( )

Figure 5: Application Class Diagram.

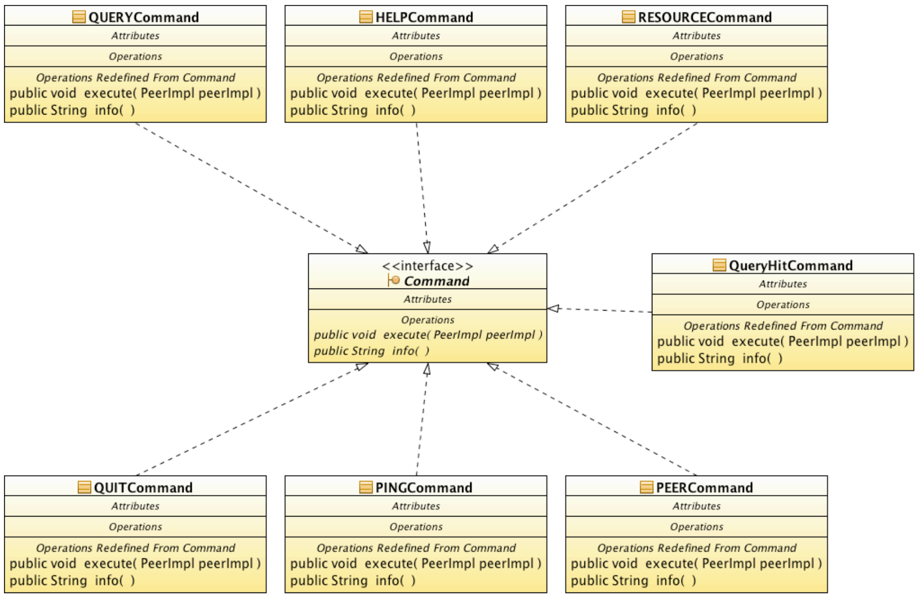


Figure 6: UI Commands Class Diagram.

Applications user interface. These commands are implemented and organized by the command design pattern.

4 Internal Protocol Specification

This section describes the messages of the system in detail and its coding/encoding as mail messages.

4.1 Ping Message

The ping message acts as the main network join message, which is send to an initial set of clients aiming to receive an answer who is on the systems network.

field	value	description
FROM	E-mail address	The requestor client.
TO	E-mail address	The destination client.
TTL	Integer value	Time to live.

4.2 FPing Message

The fping message is the forwarded message of a ping, aiming to enlarge the message by more adjacency levels.

field	value	description
FROM	E-mail address	The forwarding client.
TO	E-mail address	The destination client.
TTL	Integer value	Time to live.
REQUESTOR	E-mail address	The requestor client.

4.3 Pong Message

The pong message is mainly the answer to a PING/FPING messages requestor, saying that a client is actually in the system.

field	value	description
FROM	E-mail address	The requestor client.
TO	E-mail address	The destination client.

4.4 Query Message

The query message is the main resource retrieval message, containing a string as expression for which the requestor is asking for.

field	value	description
FROM	E-mail address	The requestor client.
TO	E-mail address	The destination client.
TTL	Integer value	Time to live.
EXPRESSION	String	The expression of the resource.

4.5 FQuery Message

The fquery message is the forwarded message of a query, aiming to enlarge the message by more adjacency levels.

field	value	description
FROM	E-mail address	The forwarding client.
TO	E-mail address	The destination client.
TTL	Integer value	Time to live.
EXPRESSION	String	The expression of the resource.
REQUESTOR	E-mail address	The requestor client.

4.6 Queryhit Message

The queryhit message is mainly the answer to a QUERY/FQUERY messages requestor, saying that a client has a resource matching the query expression.

field	value	description
FROM	E-mail address	The requestor client.
TO	E-mail address	The destination client.
RESOURCES	String list	The matching resources.

4.7 Get Message

The get message is part of the resource retrieval and is the request of a resource to get.

field	value	description
FROM	E-mail address	The requestor client.
TO	E-mail address	The destination client.
RESOURCE	String	The requested resource.

4.8 Post Message

The post message can be seen as the answere to a get message, containing the resource of interest of the get's requestor.

field	value	description
FROM	E-mail address	The forwarding client.
TO	E-mail address	The destination client.
ATTACHMENT	Base64 attachment	The resource of interest.

4.9 Message Encoding

As the system uses mail messages to transfer data, all messages have to be encodes as simple text mail messages. The encoding of these messages has been realized in a simple manner. The from and to field is used as such in the email, the subject is used as message specification i.e. which message it contains and the body is used for all the other content. To identify messages the header X-Mailer contains PowerPeering as a client.

5 Application Manual

5.1 Installation

The Power Peering application does not need an installation procedure, the only required files are the PowerPeering.jar, in the same directory the jumbo directory has to be present containing the directory resources for shared resources and an XML-file resources.xml containing the name of the shared resources.

5.2 Usage Instructions

The application can be started by the following command line command:

```
java -jar PowerPeering.jar serverHost email username password
```

This starts the system which allows to perform the following command:

command	action performed
ping	Send a ping to all adjacent clients.
query	Send a query to all adjacent clients.
hits	See a queryhits returned by the last query.
peers	See a list of all adjacent clients.
resources	See a list of all shared resources.
help	See the help of all commands.
quit	Quit the application and all its background threads.