Documentation:

SIP2P – Session Initiation Protocol in Peer-to-Peer Overlay Networks – an Experimental Implementation
Table of Contents

Table of Contents .................................................................................................................. 3

List of Figures ......................................................................................................................... 5

Acronyms.............................................................................................................................. 6

1 Abstract .................................................................................................................................. 7

2 Session Initiation Protocol – SIP ............................................................................................ 8
   2.1 Definition ........................................................................................................................ 8
   2.2 SIP URL .......................................................................................................................... 8
   2.3 SIP Network elements .................................................................................................... 8
   2.4 How it works .................................................................................................................. 9
      2.4.1 Registration............................................................................................................ 9
      2.4.2 Call setup.............................................................................................................. 10
   2.5 Features ........................................................................................................................ 11

3 Peer-to-peer networks ............................................................................................................. 12
   3.1 Dynamic hash tables ...................................................................................................... 13

4 SIP & P2P ............................................................................................................................... 14
   4.1 Approach used for the prototype .................................................................................. 14
      4.1.1 Processing of the REGISTER message .............................................................. 15
      4.1.2 Processing of the INVITE message ................................................................... 16

5 Implementation ....................................................................................................................... 17
   5.1 Use case diagram ......................................................................................................... 17
   5.2 Classes........................................................................................................................... 18
      5.2.1 Class diagram ....................................................................................................... 18
      5.2.2 SipServer ............................................................................................................. 18
      5.2.3 SipProcessor ........................................................................................................ 19
      5.2.4 KadCppApi .......................................................................................................... 19
      5.2.5 Database .............................................................................................................. 20
      5.2.6 Logger 20
      5.2.7 main 20
## Table of Contents

5.3 Configuration file ..........................................................................................20

5.4 Used libraries ..............................................................................................21
   5.4.1 KadC 21
   5.4.2 socketcc ..............................................................................................21

5.5 Prototype limitations and necessary enhancements .....................................22

5.6 Possible enhancements ................................................................................23
   5.6.1 Instant messaging – SIMPLE ..............................................................23
   5.6.2 Authorization.........................................................................................23

6 Conclusion .......................................................................................................24

7 References .......................................................................................................25

8 Appendix ..........................................................................................................27
   8.1 Source code ..............................................................................................27
      8.1.1 sip2p.h 27
      8.1.2 KadCppApi.h ....................................................................................27
      8.1.3 Database.h ......................................................................................29
      8.1.4 Logger.h ............................................................................................30
      8.1.5 SipServer.h ......................................................................................30
      8.1.6 SipProcessor.h ...............................................................................31
      8.1.7 sip2p.cpp .........................................................................................34
      8.1.8 KadCppApi.cpp .................................................................................37
      8.1.9 Database.cpp ....................................................................................39
      8.1.10 Logger.cpp .......................................................................................41
      8.1.11 SipServer.cpp ................................................................................42
      8.1.12 SipProcessor.cpp ...........................................................................43
      8.1.13 sip2p.ini ..........................................................................................52
      8.1.14 Makefile ..........................................................................................53
List of Figures

Figure 1: SIP registration .................................................................9
Figure 2: SIP call setup ...............................................................10
Figure 3: REGISTER message processing .....................................15
Figure 4: INVITE message processing .............................................16
Figure 5: Use case diagram ..........................................................17
Figure 6: Class diagram ..............................................................18
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>Authentication, Authorization and Accounting</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain name server</td>
</tr>
<tr>
<td>KadC</td>
<td>C library which implements the Kademlia algorithm</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP</td>
</tr>
</tbody>
</table>
1 Abstract

This document describes an experimental implementation of the Session Initiation Protocol (SIP) with peer-to-peer overlay networks.

SIP is an application layer protocol that is used for establishing multimedia sessions between two or more participants. It is mostly used for Voice over IP sessions, but it also can support transport of any other multimedia, such as images, presentations or videos [Rose02].

While traditional internet applications rely on client-server architectures, there are some disadvantages to this approach, such as in the case of a network of equal participants. A relative new approach is the peer-to-peer (p2p) architecture. A purely p2p architecture consists of equal peers, which act as clients and servers simultaneously and use defined algorithms for communication between them without having to contact a central server.

Fundamentally SIP is designed to work in a client-server environment. The goal of this document is to show SIP in a P2P architecture.

I would like to thank my advisor, Dr. Alexander for continuous support and reviews in my project.
2 Session Initiation Protocol – SIP

This chapter introduces SIP including its usages, the layout and names of SIP messages, standard message flows and routing mechanisms.

2.1 Definition

SIP was firstly defined in RFC 2543 which was later revised to RFC 3261. SIP is a signalling protocol and it is used to establish and maintain sessions with one or more participants. These sessions may be VoIP sessions or multimedia sessions and conferences. Per definition SIP is not restricted only to TCP/IP Protocols. [Rose02]

2.2 SIP URL

The identification of users is based on a SIP URL. SIP URLs are similar to email addresses. They consist of a user and a domain name separated by an at sign, e.g. Caller@iptel.org. Here the user name is Caller, and the domain is iptel.org. That means that the server at iptel.org is responsible for forwarding requests to and from Caller@iptel.org.

2.3 SIP Network elements

Although it is possible for two SIP clients to directly reach each other by using the IP address of the other party, a basic SIP network consists of following elements: User agents and proxy servers, registrar and redirect servers. The functionality of these three server types may be contained in a single server. Registrar servers are used for registering users and redirect servers just redirect calls to another location. Proxy servers, in turn do call routing besides providing caller authentication, authorization and accounting (AAA) services. Lastly user agents (UAs) are the actual endpoints. They may take the form of a program running on a computer, or be physical telephones which implement SIP protocol [SiKu07].
2.4 How it works

2.4.1 Registration

A user must register in order to be able to receive and initiate SIP sessions using a SIP URL. The only exceptions are direct IP to IP calls. These calls are initiated by sending REGISTER message to a SIP registrar server. A user sends the callee SIP URL in the To-field, and its IP address in the Contact-field. If necessary, the registrar sends the information to a location server followed by a confirmation to the user that the registration was ok.

Figure 1: SIP registration, source [SiKu07], modified
2.4.2 Call setup

Figure 2: SIP call setup, source [SiKu07], modified

A simplified SIP call setup is depicted above. For the sake of simplicity some of the messages are omitted, such as TRYING and RINGING. The numbers in red circles indicate the order in which the messages are sent.

Here a user called caller@otherdomain.com is trying to call user user@somedomain.com from previous example. The caller’s user agent of Caller sends an INVITE message to the proxy server running at default port 5060 at otherdomain.com. If needed the proxy server resolves the address of somedomain.com by sending a DNS query and forwards the INVITE message to that server. The proxy server at somedomain.com knows the location of user and forwards the INVITE message again. If user is able to receive calls, a RINGING message will be sent through the proxies to caller@otherdomain.com indicating that the call setup was successful and that user’s phone is ringing.

When user accepts the call, then an ACK message is sent directly to caller@otherdomain.com without any proxies in between and the media session is started. Finally, when one of the parties terminates the session a BYE message is sent to the other one.
2.5 Features

Below is a list of some features supported by the SIP protocol:

- Call forwarding
- Callee and caller number delivery
- Personal mobility: Users may change their end terminals but they will still be reachable under the same sip URL.
- Terminal-type negotiation
- Caller and callee authentication
- Multicast conferences

[Schu07]
3 Peer-to-peer networks

Classic internet applications use client-server architectures. That means that a central server is providing some or all application logic, and clients contact servers in order to receive resulting information. For example, for a simple Web HTTP service this is logical architecture because a HTTP server provides the information and all clients receive it directly from there, i.e. it is a one-to-many communication.

As for SIP, it is currently also using a client-server architecture. Central servers are responsible for registering and locating users and for establishing SIP sessions. But telephony, no matter if over IP or not, is actually a form of many-to-many communication. Each user should be able to reach all other users in a particular network. While in traditional telephony there had to be some central point for routing calls, the internet provides mechanisms where the need for such central points disappears. This is realized by peer-to-peer networks.

P2P networks have following characteristics: Self-organisation, symmetric communication and distributed control. "It automatically adapts to the arrival, departure and failure of nodes" [RiMo06]. In pure p2p networks there are no servers and basically no clients. All peers have same rights and all are responsible for routing the messages within the network to their final destinations. In client-server systems the services and resources where provided by a centralized entity, whereas in p2p networks they are provided by the peers themselves [Singh05]. Not all peers in a network must provide those services, but some of them do. This was it will be ensured that collectively the peers can provide the particular services [Brya06].

P2P networks emerged when file sharing applications, such as Napster, Gnutella and others appeared on the internet. Some of them were pure p2p networks, and some used client-server-structure for some tasks, e. g. user management. In early systems a so called “flood search approach” was used for locating a file in a network. A peer searching for a file sends a message to its neighbours. If the neighbour had the file, it would send the requestor a positive response. If not, the message would be forwarded to next neighbours. And
these do not know about the outcome of the previous request and would sent responses to the initial requestor even if the file was already found. That approach proved to be inefficient and therefore dynamic hash tables have been introduced, which will be described in the next chapter [Brya06]. Current research focuses on other areas which may benefit from p2p networks. In order to use full potential of p2p networks, the robustness of such networks is to be demonstrated [RiMo06].

### 3.1 Dynamic hash tables

There are many algorithms which enable p2p networks possible. It depends on the type of application which tasks should be centralized and which should be done by peers. For our telephony application it is important to store user data, i.e. the mappings between their user names and IP addresses., which can be done by using distributed hash tables. There are two algorithms which have been examined for our application, Chord and Kademia.

Chord is a p2p protocol which uses routed queries to locate a key in a network with a minimal number of hops. The number of hops grows logarithmically with the number of nodes. The nodes in a Chord network are organized in a circle, so each node has a successor and a predecessor. When searching for a key, the messages are always forwarded circularly, hence the distance between the current node and the destination node is halved [Morr03]. In order to guard against node failures Chord sends keep-alive messages to its successors entries and continuously updates them [Baset05].

Kademlia is another DHT implementation. When connecting to a Kademlia network, a node must know the IP address of at least one another node participating in the network. This can be done by saving the nodes list locally when leaving the network or by retrieving it from some a server-supernode. Unlike Chord Kademlia uses XOR-metric to compute the distance between two peers and there are no successor lists. The first entry in the routing table is the immediate neighbour [Baset05]. The KadC library is a C implementation of Kademlia and it was used for our application for storing the IP addresses of SIP users in a DHT cloud.
4 SIP & P2P

As described in chapter 2 the current SIP architecture uses central servers in order to persist information about users and to route calls to desired endpoints. There are basically two approaches for combining SIP and p2p: The first is to replace the location service in SIP by a p2p protocol (SIP-using-P2P). And the second approach is to implement p2p protocol itself using SIP messages (P2P-over-SIP) [SiSu06]. In the first case p2p is used only for lookups and updates of mappings between SIP URLs and users IP addresses. In the second case SIP messages are extended to convey information needed for routing the messages to the desired endpoints. In this chapter we will focus on the first. In the following chapter a small application will be introduced, implementing a modified version of SIP-using-p2p.

4.1 Approach used for the prototype

A server is created and a listening UDP socket is initialised on the SIP default port 5060. The IP address of the server will be inserted as that of a SIP server in any SIP client. Actually that server acts as a SIP proxy server, but it is only used as a proof of concept for the idea.

Receipt of a REGISTER message will be used to publish the user data binding in the p2p network.

When making calls, the address of the called party will be searched for in the p2p network, the INVITE message will be modified accordingly and the request will be forwarded to that destination. If necessary, the application acts as a proxy forwarding the following SIP messages between the caller and the called party.
4.1.1 Processing of the REGISTER message

The processing of a REGISTER message is quite simple. Upon receiving a REGISTER message, the user data will be published in Kademlia network, and a SIP OK response will be sent to the sender. The format for publishing user data is following: The key is the user name. And the value is composed out of following components: User name, IP address and the port on which the SIP client is running. For example if the user name is "Marcus", and his SIP client runs on IP address 208.167.23.150 on port 20000, then the published value is Marcus@208.167.23.150:20000. If the port equals to the SIP default port 5060, then the port number and the preceding colon are omitted.
4.1.2 Processing of the INVITE message

When processing the INVITE message, the application acts as a SIP proxy server. Yet, it does not only forward the INVITE message to some destination, it rewrites the value of the “To:” field. Firstly, a TRYING message is sent to the caller. Then it extracts the existing value from the To-field. The extracted value should be the name of the user, which was published upon receiving a REGISTER message (s. above). A search in Kademlia is done, and the value of the To-field is replaced by the retrieved 3-tuple user, IP address and port. According to the above example, if the To-field originally contained “Marcus”, it would have been replaced by Marcus@208.167.23.150:20000. The rest of the INVITE message is copied and sent to the destination. The following RINGING and OK messages are just forwarded to the caller while the data plane is handled by the SIP clients.
5 Implementation

The sample application is implemented in C++ and for the moment only runs on Linux systems. It utilizes two libraries: KadC and socketcc, which are described later in the text.

5.1 Use case diagram

![Use case diagram]

Figure 5: Use case diagram

Our implementation supports only two use cases: registration and simple call set-up. A user agent will be able to register in our p2p network and to call other users in the same network. Possible enhancements are listed in 5.6.
5.2 Classes

5.2.1 Class diagram

The class diagram above is not complete. It just depicts the methods needed for adequate comprehension. Some methods have parameters not present in the diagram, and some parameters are passed as pointers, constant references etc. For more details, please refer to the source code (s. Appendix).

5.2.2 SipServer

SipServer is the class which handles the SIP request handling. It creates a listening UDP socket on the default (or specified) port. Upon receiving a request, it forwards the request to SipProcessor’s only public method `processRequest`. 
5.2.3 SipProcessor

SipProcessor is the class which implements the SIP functionality. The constructor needs a pointer to a UDPServerSocket and a pointer to an IPAddress object. Both are used to send responses to the calling party. Depending on which request is received, one of the subroutines `processRegister` or `processInvite` is called. If an error occurs, then one of the functions sendXXX sends the appropriate SIP error response to the caller. At the moment the methods just send simple strings back to the caller, which is not in-line with the SIP protocol’s error codes. This should be updated, so that SIP conforming responses are sent.

The `processRegister` method the extracts the user data from the Contact line. It is then published under the given user name, and a SIP OK response is sent back to the caller.

The `processInvite` method is a little bit more sophisticated. First it sends a TRYING message to the caller. This is done before forwarding the INVITE message, so not to cause a timeout on the calling client. Then it uses the header field “To” to get the name of the called party. A search for the name is done in Kademlia, and the value from the To-Header is replaced by the data found in Kademlia. The rest of the INVITE message is simply copied. Then it opens a socket to the called party and sends the rewritten INVITE message. If it is received properly, the client sends a TRYING and afterwards a RINGING message. TRYING can be ignored at this stage, as it was already sent previously, but the RINGING message will be passed through to the caller without any modifications. When the called party picks up, an OK message is sent to our application which is forwarded to the caller. Upon termination of the call by either the caller or callee, the resources are released and the SipServer waits for the next SIP request.

5.2.4 KadCppApi

The implementation of the Kademlia algorithm is done in the KadC library. The KadCppApi is a C++ wrapper of the KadC API and implements most useful
functions. The most important are publish and search which respectively publish and search for data in the Kademlia network.

### 5.2.5 Database

The Database is another wrapper around the KadCppApi used for persistence throughout the application. It is used to enable simple replacement of Kademlia if this should be necessary, as p2p-networks should be flexible concerning the protocol maintaining the DHT with user data [Baset05]. It implements the singleton pattern, and has only two methods put and get, which should be self explanatory.

### 5.2.6 Logger

Finally, the Logger class is used for unified logging. The constructor takes a string, which will precede all debug messages. This is used to mark which class sent a given debug message. A method called debug is overloaded several times in order to be able to call the same method with different parameters.

### 5.2.7 main

The main method, i.e. the program entry point is implemented in file called sip2p.cpp. It just reads the settings file and stores stanzas in global variables if needed. These are defined in the header file "sip2p.h".

### 5.3 Configuration file

Different settings can be put in a configuration file, which will be read when the program starts. The default name for that file is “sip2p.ini” but it may be changed and passed as an argument to the program.

Comments start with a '#' sign. All lines starting with a hash key will be ignored. Actually if a key name cannot be interpreted, the line containing that key will also be ignored. Also be careful.

The most important settings which must be set for the daily use are S2P_SERVER_IP and S2P_SERVER_PORT. They contain the ip address,
(also the IP address of the localhost) and the port under which the UDP Server will be started. They always must be set. The rest is just used for debugging purposes.

The KADC_INI_FILE specifies location of the configuration file for KadCapi, the DHT library used in this project. It should be always kadc.ini, but for debugging purposes, or when running more than one process of sip2p, different ports can be specified for KadCapi to use in two different configuration files.

If DEBUG option is set to TRUE (it is case sensitive, so TRUE, true, or True... are actually FALSE), then the program behaves differently from normal mode. For details, please see the source code.

Between the lines [DUMMY_P2P_USERS] and [DUMMY_P2P_USERS_END] you may put names of some dummy users and their SIP URLs. These may be real SIP accounts, which are not stored in Kademlia. If preceded by a hash sign #, these dummy users will also be ignored.

If the flag DUMMY_PUBLISH is TRUE, then the dummy users will be published in the Kademlia network. Also if the DUMMY_FAST_SEARCH is set to TRUE, then first the names from the configuration file will be searched, and not Kademlia. That way you may speed up the development process, without having to wait for Kademlia searches, if you are sure they work.

### 5.4 Used libraries

#### 5.4.1 KadC

KadC is an implementation of the Kademlia protocol written in C. It is available from http://kadc.sourceforge.net/. It defines a simple API for storing and retrieving key value pairs from the network.

#### 5.4.2 socketcc

socketcc is a library that contains standard socket API. It supports both TCP and UDP sockets. It is available from http://sourceforge.net/projects/socketcc/.
5.5 Prototype limitations and necessary enhancements

The implemented prototype just supports successful calls and registrations and there is no real exception and error handling. Below is a list of limitations with a short explanation:

- **Blocking server**
  - The server does not support multiple requests. Only when a request is fully processed, the server is ready for the next request.

- **Program stops if the called party do not answer**
  - The program relies on receiving the TRYING, RINGING and OK or CANCEL message from the called party. If this is not the case, the program waits and has to be restarted in order to receive new INVITE messages. This could be changed by using threads.

- **Multiple registrations**
  - When a REGISTER message is received, the OK response is immediately sent to the caller and the data is published in the DHT cloud. No check is done if a user with the same name already exists. If it is the case, then the old data will be overwritten in Kademlia.

- **Update of Kademlia peers**
  - For successful invoking the program, a list of Kademlia peers is stored in kadc.ini file. This file is used by the KadC library and should be updated at regular intervals. This should happen automatically when the KadC resources are released. There is also an API method in the library for that purpose. Yet, both cause segmentation faults, which could not be corrected by the author.

- **Error codes**
If an error occurs, e.g. a user is not found, an invalid request is received etc, only simple strings are sent back to the caller. This is not SIP compliant and should be updated, so that valid SIP error responses are generated.

5.6 Possible enhancements

Future work on the application may of course include solutions to the limitations described above.

5.6.1 Instant messaging – SIMPLE

It would be interesting to extend our application to support instant messaging. RFC 3428 describes a SIP message called MESSAGE. The MESSAGE method itself does not have a notion of a session, each message stands alone, and so the instant messaging capabilities would have to be implemented by the clients [Camp02].

5.6.2 Authorization

At the moment only direct IP to IP calls are possible. That means we can map names only to IP addresses, but not for example to other real SIP URLs. To be able to do that the authorisation would have to be supported by our server. In order to do this, the processing of the REGISTER message should be extended.
6 Conclusion

As this document and the application show, it is relatively simple to implement basic SIP functionality in a p2p network. The author’s opinion is that the best approach is to replace the location service in SIP with a p2p protocol (SIP using P2P) [SiSu06], which renders the location and proxy servers in SIP dispensable.

The sample application did not address other important issues. Technical issues with need for improvement are discussed in chapter 5.6. Yet, security, stability, legal issues and ability of p2p networks to defend themselves against attacks are not discussed in this document and form areas for further research.
7 References


[Camp02] Campbell et.all, “SIP Extentsion for Instant Messaging”, RFC 3428


8 Appendix

8.1 Source code

8.1.1 sip2p.h

```cpp
#ifndef SIP2P_H
#define SIP2P_H

#include <string>
#include <map>

// Data read from settings file = GLOBAL VARIABLES
int s2pServerPort;
std::string s2pServerIP;
std::string kadcIniFile;
bool DEBUG = false;
std::map<std::string, std::string> dummyUsers;
bool DUMMY_PUBLISH = false;
bool DUMMY_FAST_SEARCH = false;

// Name of the log file
std::string logFileName = "sip2p.log";

// Function definitions
bool readSettings(char* fileName);

#endif //SIP2P_H
```

8.1.2 KadCppApi.h

```cpp
#ifndef KADCPPAPI_H
#define KADCPPAPI_H

extern "C" {
#include "int128.h"
#include "rbt.h"
#include "KadCapi.h"
#include "KadClog.h"
};

#include <string>

#include "Logger.h"

/**
 * C++ representation of KadCapi.
 */
class KadCppApi {
```
private:
    KadCcontext context;
    Logger * log;

public:

    /** Full constructor */
    KadCppApi(
        const std::string & inifilename = "kadc.ini",
        int leafmode = 1,
        int init_network = 0);

    static KadCppApi* getInstance();

    /** Destructor */
    virtual ~KadCppApi();

    /** Close */
    void close();

    /** Returns number of k-nodes */
    int getKNodesCount();

    /** Returns number of contacts */
    int getContactsCount();

    /** Returns firewall status */
    int getFirewallStatus();

    /** Returns our IP adress as seen by other peers. */
    unsigned long int getExternalIP();

    /** Returns our local hash ID. */
    std::string getHashId();

    /** Returns our UDP port number. */
    unsigned short int getUDPport();

    /** Returns our TCP port number */
    unsigned short int getTCPport();

    /** Prints a status to output. */
    void printStatus();

    // KadC style search
    int publish(const std::string & key, const std::string & value);

    /** search.
     * Returns the found string, or 0 if not found.
     */
    std::string search(const std::string& key);

    /** Updates the values in the ini file. */
    void updateIniFile();
8.1.3 Database.h

#ifndef DATABASE_H_
#define DATABASE_H_

#ifdef DATABASE_H_
#endif // DATABASE_H_
8.1.4 Logger.h

```cpp
#ifndef LOGGER_H_
#define LOGGER_H_

#include <string>
#include <iostream>
#include <fstream>

/**
 * Utility class for logging.
 * The methods should be self-explainatory.
 *
 * It writes the output to the standard output
 * and to a logfile called "sip2p.log".
 */

class Logger
{
 public:
 Logger(const std::string& senderName);
 virtual ~Logger();
 void debug(const std::string& msg);
 void debug(const std::string& msg, int number);
 void debug(const char * msg);
 void debug(const char * msg, int number);
 void debug(int i);
 private:
 std::string name;
 static std::ofstream outFile;


};
#endif /*LOGGER_H_*/
```

8.1.5 SipServer.h

```cpp
#ifndef SIPSERVER_
#define SIPSERVER_

#include "Logger.h"
#include "socketcc.h"

/**
 * The SIP UDP Server.
 * The instances of this class create a listening socket
 * on default port 5060, or another one, and just forwards
 * the received messages to the SipProcessor.
 * SipServer uses the socketcc library for creating sockets.
 */

class SipServer {
 public:
 // The interface:
 /**
 * Default constructor creates listening UDP socket on port 5060.
 ```
SipServer();
/**
 * This constructor takes the port number where to listen
 * to for SIP requests.
 */
SipServer(int port);
/**
 * Destructor releases the resources.
 */
virtual ~SipServer();
/**
 * Starts the server.
 * Must be called after construction.
 */
void start();
private:
/**
 * The buffer size for receiving and sending requests.
 */
const static int BUFFER_SIZE = 65535;

/* Logger */
static Logger* log;
/* Port number */
int serverPort;
};
#endif /*SIPSERVER_*/

8.1.6 SipProcessor.h

#ifndef SIPPROCESSOR_H_
#define SIPPROCESSOR_H_

#include "Logger.h"
#include "socketcc.h"
#include <string>

/**
 * Main class for SIP request processing.
 */

class SipProcessor{
 public:
 /**
 * Constructor
 * Gets a pointer to a UDPServerSocket, and the IP and the port of the caller.
 */
 SipProcessor(UDPServerSocket* sSocket, IPAddress* callerIP, int cPort);
 /**
 * Destructor
 */
virtual ~SipProcessor();

/**
 * Processes a SIP request. The main entry point for request processing.
 * Searching and publishing and request forwarding is done here and by subrou-
 * tines.
 */
void processRequest(const std::string& request);

private:
    // Static constants
    const static std::string REGISTER;
    const static std::string INVITE;
    const static std::string ACK;
    /** SIP Header. */
    const static std::string SIP_HEADER_OK;
    const static std::string SIP_HEADER_TRYING;
    const static std::string SIP_HEADER_MOVED_TEMPORARILY;

    // Request buffe size
    const static int BUFFER_SIZE = 65535;

    // Fields
    /** Logger. */
    Logger* log;
    /** Server socket */
    UDPServerSocket* serverSocket;
    /** Socket open to the callee if needed. */
    UDPConnectedSocket* calleeSocket;
    /** IP address of the caller and callee */
    IPAddress* callerIPSource, calleeIPSource;
    /** Opened ports to caller and callee. */
    int callerPort, calleePort;
    /** The original message send. */
    std::string originalRequest;
    /** Prepared contact data from original request.
     * 0 -> name
     * 1 -> ip address
     * 2 -> port
     */
    std::string responseContact[3];

    // Private methods

    /**
     * Processes a SIP REGISTER request. The user data from the request
     * will be stored in kademlia network, and the user will be from then
     * on available for P2P SIP calls.
     */
void processRegister();
/**
 * Processes a SIP INVITE request. It searches for the specified
 * name in TO: field in kademlia network, and searches for user data.
 * If not found, a NOT FOUND response will be sent to caller.
 * If found, the data in original request are replaced by the one
 * found in kademlia network, and the prepared INVITE request will
 * be sent to the found user.
 * Then the needed message forwarding between caller and callee
 * will be done, and afterwards the function exits.
 */
void processInvite();
/**
 * If an ACK message is received directly on the listening port,
 * then it just needs to be forwarded to the party in the TO field.
 */
void processAck();
/**
 * Sends a response to caller indicating a server error.
 */
void sendServerError();
/**
 * Sends a response to caller indicating that request is not supported.
 */
void sendRequestNotSupported();
/**
 * Sends a response to caller indicating an invalid request.
 */
void sendInvalidRequest();
/**
 * Sends a response to caller indicating that a user could not be found.
 */
void sendNotFound();
/**
 * Publishes data from REGISTER message in the kademlia network.
 */
publishUserData();
/**
 * Prepares the INVITE message. It searches for user data in kademlia
 * and replaces the needed field with data found in kademlia network.
 * If any error happens, an error message is sent to requestor,
 * and an empty string "" is returned.
 */
const std::string prepareInvite();
/**
 * Creates the TRYING message. Because it takes some time to find
 * the user data in kademlia, the TRYING message will be instantly
 * sent to the caller. Then after TRYING and RINGING is received from
 * the callee, the RINGING message will be forwarded to the caller.
 */
const std::string createTryingMessage();
8.1.7 sip2p.cpp

```cpp
#include "sip2p.h"
#include "Logger.h"
#include "Database.h"
#include "SipServer.h"

#include "time.h"
#include <iostream>
#include <sstream>
#include <fstream>
using namespace std;

/** Logger */
static Logger * log = new Logger("sip2p");

/** Main */
int main(int argc, char* argv[]) {
   //
   // Read data from configuration file
   //
   // Get file name
   char* confFile = "sip2p.ini";
   if (argc == 1) {
      log->debug("No configuration file specified -> default sip2p.ini will be used.");
   } else {
      confFile = argv[1];
   }

   // Read settings. If error, exit
   if (!readSettings(confFile)) {
      log->debug("Could not read settings. Program will exit!");
   }
   return 0;
}
```
exit(1);
}

// Init database
Database::getInstance();

// Start the server
try {
SipServer server(s2pServerPort);
server.start();
} catch (SocketException &e) {
    string msg = string((const char*) e);
    log->debug("ERROR: " + msg);
}

// Release memory
Database::close();
delete log;
return 0;
}

/** Reads setting from configuration file */
bool readSettings(char* fileName) {
    const string SERVER_PORT = "S2P_SERVER_PORT";
    const string SERVER_IP = "S2P_SERVER_IP";
    const string KADC_INI_FILE = "KADC_INI_FILE";
    const string KEY_DEBUG = "DEBUG";
    const string DUMMY_USERS_BEGIN = "[DUMMY_P2P_USERS]";
    const string DUMMY_USERS_END = "[DUMMY_P2P_USERS_END]";
    const string KEY_DUMMY_PUBLISH = "DUMMY_PUBLISH";
    const string KEY_DUMMY_FAST_SEARCH = "DUMMY_FAST_SEARCH";

    ifstream inFile(fileName);
    if(inFile.is_open()) {
        string line;
        while(! inFile.eof()) {
            getline(inFile, line);
            if (line.find("#") == 0) {
                // Comments start with '#'
                continue;
            }
            if (line.find(SERVER_PORT) != string::npos) {
                // read port
                int index = line.find('=');
                string portString = line.substr(index + 1) ;
                s2pServerPort = atoi(portString.c_str());
            } else if (line.find(SERVER_IP) != string::npos)  {
                // read ip adress
                int index = line.find('=');
                s2pServerIP = line.substr(index + 1).c_str();
            } else if (line.find(KADC_INI_FILE) != string::npos) {
                // read kadc ini file
            }
        }
    }
    return false;
}
int index = line.find('=');
kadcIniFile = line.substr(index + 1);
} else if (line.find(KEY_DEBUG) != string::npos) {
    // debug mode
    int index = line.find('=');
    if (line.substr(index + 1) == "TRUE") {
        DEBUG = true;
    }
} else if (line.find(KEY_DUMMY_PUBLISH) != string::npos) {
    // publish dummy users
    int index = line.find('=');
    if (line.substr(index + 1) == "TRUE") {
        DUMMY_PUBLISH = true;
    }
} else if (line.find(KEY_DUMMY_FAST_SEARCH) != string::npos) {
    // fast search dummies
    int index = line.find('=');
    if (line.substr(index + 1) == "TRUE") {
        DUMMY_FAST_SEARCH = true;
    }
} else if (line.find(DUMMY_USERS_BEGIN) != string::npos) {
    // Read dummy users
    for (getline(inFile, line); line != DUMMY_USERS_END; getline(inFile, line)) {
        if (line.find("#") == 0) {
            // Comments start with '#'
            continue;
        }
        int index = line.find('=');
        string key = line.substr(0, index);
        string value = line.substr(index + 1);
        dummyUsers[key] = value;
    }
}

inFile.close();

// Print out what is read
log->debug("SIP Server IP: " + s2pServerIP);
log->debug("SIP Server port: " , s2pServerPort);
log->debug("KadcFile: " + kadcIniFile);
if (DEBUG) {
    log->debug(" -- DEBUG Mode on! --");
}
if (DUMMY_PUBLISH) {
    log->debug(" -- Dummy publishing on! --");
}
if (DUMMY_FAST_SEARCH) {
    log->debug(" -- Dummy fast search on! --");
}
log->debug("-----------------------------");
return true;
} else {
    log->debug("Could not find conf file " + string(fileName));
}
8.1.8 KadCppApi.cpp

```cpp
#include <iostream>
#include <sstream>
#include "KadCppApi.h"

using namespace std;

// Default values
string HASH_INDEX = "SIP2PEMIR";

KadCppApi::KadCppApi(const string& inifilename, int leafmode, int initNetwork) {
    KLogOpen("KadC.log"); // Removes KadC loggin f rom standard output.
    log = new Logger("KadCppApi");
    log->debug("Constructor called...");
    context = KStart(const_cast<char*>(inifilename.c_str()), leafmode, initNetwork);
}

KadCppApi::~KadCppApi() {
    log->debug("Destructor called...");
    KStop(&context);
    delete log;
}

void KadCppApi::close() {
    KStop(&context);
    log->debug("API closed...");
}

int KadCppApi::getKNodesCount() {
    return KGetnknodes(&context);
}

int KadCppApi::getContactsCount() {
    return KGetncontacts(&context);
}

int KadCppApi::getFirewallStatus() {
    return KGetfwstatus(&context);
}

unsigned long KadCppApi::getExternalIP() {
    return KGetextIP(&context);
}

string KadCppApi::getHashId() {
    //return KGetourhashID(&context);
```
return "not yet implemented";
}

unsigned short KadCppApi::getUDPport() {
    return KadC_getourUDPport(&context);
}

unsigned short KadCppApi::getTCPport() {
    return KadC_getourTCPport(&context);
}

void KadCppApi::printStatus() {
    log->debug("Status: ", context.s);
    log->debug("Number of kNodes: ", KadC_getnknodes(&context));
    log->debug("Number of contacts: ", KadC_getncontacts(&context));
}

int KadCppApi::publish(const string & key, const string & value) {
    log->debug("Publishing: " + key + " = " + value);
    string index = HASH_INDEX + key;
    string metalist = key + "=" + value;
    return KadC_republish(&context,
        const_cast<char*>(index.c_str()),
        const_cast<char*>(HASH_INDEX.c_str()),
        const_cast<char*>(metalist.c_str()),
        5, 10);
}

/**
 * Searches for a key
 */
string KadCppApi::search(const string & key) {
    log->debug("searching for key: " + key);
    // Create Hash for the key
    string index = HASH_INDEX + key;
    string foundString = "";

    // get rbt *
    void * voidPtr = KadC_find(&context,
        const_cast<char*>(index.c_str()),
        "", 5, 1, 10);

    // Iterate over rbt
    void * iter;
    KadCdictionary * pkd;
    for(iter = rbt_begin(voidPtr); iter != NULL; iter = rbt_next(voidPtr, iter)) {
        pkd = rbt_value(iter);
        KadCtag_iter kdIter;
        KadCtag_begin (pkd, &kdIter);
        for (int i = 0; i < kdIter.tagsleft; i++) {
            // Get Value of first tag
        }
    }
}
//return string(kdIter.tagvalue);
if (foundString == "") {
    foundString = string((kdIter.tagvalue));
}
    // KadCtag_next(&kdIter);
}

// Iterate again and free memory
for (iter = rbt_begin(voidPtr); iter != NULL; iter = rbt_begin(voidPtr)) {
pkd = rbt_value(iter);
rbt_erase(voidPtr, iter);
KadCdictionary_destroy(pkd);
} return foundString; // not found

// Update ini file
void KadCppApi::updateIniFile() {
    // TODO - simply cannot link the method name. I don't know why.
    //KadC_write_inifile(&context, NULL);
}

8.1.9 Database.cpp

#include "Database.h"
#include <map>
using namespace std;

// Init static values
Database* Database::instance = 0;

// Obtains the singleton instance
Database* Database::getInstance() {
    if (instance == 0) {
        instance = new Database();
        instance->init();
    }
    return instance;
}

void Database::close() {
    delete instance;
    instance = 0;
}

void Database::init() {
    // Wait for at least 20 knodes
    log->debug("Initialising database: Waiting for at least 20 kNodes..."); 
    int lastKnodesCount = -1;
    int x;
    while ((x = api->getKNodesCount()) < INIT_KNODES_COUNT) {
        if (x > lastKnodesCount) {
Appendix

```cpp
log->debug("- #kNodes = ", x);
lastKnodesCount = x;
}

// If DEBUG on, publish some dummy values in kademlia
extern bool DUMMY_PUBLISH;
if (DUMMY_PUBLISH) {
    log->debug("Publishing dummy values...");
    int countPublished = -1;

    extern map<string, string> dummyUsers;
    map<string, string>::iterator iter;
    for (iter = dummyUsers.begin(); iter != dummyUsers.end(); iter++) {
        string key = iter->first;
        string value = iter->second;
        log->debug("Publishing: " + key + " = " + value);
        countPublished = api->publish(key, value);
        if (countPublished > 0) {
            log->debug(" - OK");
        } else {
            log->debug(" - Error");
        }
    }
}

int Database::put(const string& key, const string& value) {
    return api->publish(key, value);
}

const string Database::get(const string& key) {
    // If DUMMY_FAST_SEARCH=TRUE in ini file, then first the dummy users will be
    // searched, without kademlia.
    extern bool DUMMY_FAST_SEARCH;
    if (DUMMY_FAST_SEARCH) {
        extern map<string, string> dummyUsers;
        log->debug("DEBUG Search for..." + key);
        map<string, string>::iterator iter = dummyUsers.find(key);
        // If found return it, if not, then normal search is done.
        if (iter != dummyUsers.end()) {
            return iter->second;
        }
    }

    // Update ini file before each search.
    log->debug("Searching for... " + key);
    api->updateIniFile();
    return api->search(key);
}

// Private constructors and destructor
Database::Database() {
    extern string kadcIniFile;
```
Appendix Page 41

8.1.10 Logger.cpp

```cpp
#include "Logger.h"
using namespace std;

// Init static member
extern string logFileName;
ofstream Logger::outFile(logFileName.c_str());

Logger::Logger(const string& senderName)
{
    name = senderName;
}

Logger::~Logger()
{
}

void Logger::debug(const string& msg) {
    cout << name << "": " " << msg << endl;
    Logger::outFile << name << "": " " << msg << endl;
}

void Logger::debug(const char * msg) {
    cout << name << "": " " << msg << endl;
    Logger::outFile << name << "": " " << msg << endl;
}

void Logger::debug(int i) {
    cout << name << "": " " << i << endl;
    Logger::outFile << name << "": " " << i << endl;
}

void Logger::debug(const string& msg, int number) {
    cout << name << "": " " << msg << number << endl;
    Logger::outFile << name << "": " " << msg << number << endl;
}

void Logger::debug(const char * msg, int number) {
    cout << name << "": " " << msg << number << endl;
    Logger::outFile << name << "": " " << msg << number << endl;
}
```
8.1.11 SipServer.cpp

```cpp
#include "SipServer.h"
#include "SipProcessor.h"
#include "Database.h"
#include <sstream>
using namespace std;

// Define static values
Logger* SipServer::log = new Logger("SipServer");

SipServer::SipServer() {
    log->debug("Entering constructor SipServer() ..");
    serverPort = 5600;  // default port
}

SipServer::SipServer(int port) {
    log->debug("Entering constructor SipServer(int port) ..");
    serverPort = port;
}

SipServer::~SipServer() {
    log->debug("Entering destructor...");
    delete log;
}

void SipServer::start() {
    extern string myIP;
    log->debug("Starting the server...");
    /** Data from socketcc */
    UDPServerSocket serverSocket(serverPort);
    IPAddress cIPSource;
    int iBytesTransferred, iPortSource;
    char pcBuffer[BUFFER_SIZE];

    //cIPSource.SetAddress(myIP.c_str(), false);

    pcBuffer[0] = 0;

    ostringstream os;
    os << "SIP Server started at " << serverSocket.LocalIPAddress() << ":" << serverSocket.LocalPortNumber();
    log->debug(os.str()); os.clear();

    for (; ;)
    {
        // Wait for a request
        iBytesTransferred = serverSocket.Receive Datagram(pcBuffer, BUFFER_SIZE, cIPSource, iPortSource);

        // Request received
        string request(pcBuffer);
    }
}
```
Appendix Page 43

8.1.12 SipProcessor.cpp

#include "SipProcessor.h"
#include "Database.h"
#include "time.h"
#include <sstream>
using namespace std;

// Init static members
const string SipProcessor::INVITE = "INVITE";
const string SipProcessor::REGISTER = "REGISTER";
const string SipProcessor::ACK = "ACK";
const string SipProcessor::SIP_HEADER_OK = "SIP/2.0 200 OK";
const string SipProcessor::SIP_HEADER_TRYING = "SIP/2.0 100 Trying";
const string SipProcessor::SIP_HEADER_MOVED_TEMPORARILY = "SIP/2.0 302 Moved Temporarily";
//const string SipProcessor::SIP_HEADER_MOVED_TEMPORARILY = "SIP/2.0 305 Use Proxy";

// Constructor
SipProcessor::SipProcessor(UDPServerSocket* sSocket, IPAddress* callerIP, int cPort) {
    log = new Logger("SipProcessor");
    serverSocket = sSocket;
    callerIPSource = callerIP;
    callerPort = cPort;
}

// Destructor
SipProcessor::~SipProcessor() {
}

// Processes a SIP request.
void SipProcessor::processRequest(const string& request) {
    originalRequest = request;
    log->debug("Request received.");

    istringstream is;
    is.str(request);
    string line;
    getline(is, line);
    if (line.find(REGISTER) == 0) {
        processRegister();
    } else if (line.find(INVITE) == 0) {
        processInvite();
    } else {
        log->debug("Received unknown request: 
" + request);
    }
}

8.1.12 SipProcessor.cpp

#include "SipProcessor.h"
#include "Database.h"
#include "time.h"
#include <sstream>
using namespace std;

// Init static members
const string SipProcessor::INVITE = "INVITE";
const string SipProcessor::REGISTER = "REGISTER";
const string SipProcessor::ACK = "ACK";
const string SipProcessor::SIP_HEADER_OK = "SIP/2.0 200 OK";
const string SipProcessor::SIP_HEADER_TRYING = "SIP/2.0 100 Trying";
const string SipProcessor::SIP_HEADER_MOVED_TEMPORARILY = "SIP/2.0 302 Moved Temporarily";
//const string SipProcessor::SIP_HEADER_MOVED_TEMPORARILY = "SIP/2.0 305 Use Proxy";

// Constructor
SipProcessor::SipProcessor(UDPServerSocket* sSocket, IPAddress* callerIP, int cPort) {
    log = new Logger("SipProcessor");
    serverSocket = sSocket;
    callerIPSource = callerIP;
    callerPort = cPort;
}

// Destructor
SipProcessor::~SipProcessor() {
}

// Processes a SIP request.
void SipProcessor::processRequest(const string& request) {
    originalRequest = request;
    log->debug("Request received.");

    istringstream is;
    is.str(request);
    string line;
    getline(is, line);
    if (line.find(REGISTER) == 0) {
        processRegister();
    } else if (line.find(INVITE) == 0) {
        processInvite();
    } else {
        log->debug("Received unknown request: 
" + request);
    }
}
} else if (line.find(ACK) == 0) {
    processAck();
} else {
    sendRequestNotSupported();
}
}

// Private methods

void SipProcessor::processRegister() {
    log->debug("Entering processRegister...");

    istringstream is(originalRequest);
    ostringstream responseStream;
    string line;

    responseStream << SIP_HEADER_OK << endl;
    string contactLine;

    // Copy needed lines from request to response
    for (getline(is, line); line != "");

    if (line.find("Via:")) {
        responseStream << line << endl;
    } else if (line.find("To:")) {
        responseStream << line << endl;
    } else if (line.find("From:")) {
        responseStream << line << endl;
    } else if (line.find("Call-ID:")) {
        responseStream << line << endl;
    } else if (line.find("CSeq:")) {
        responseStream << line << endl;

        responseStream << "Date: " << getDateAndTime() << endl;
    } else if (line.find("Contact:")) {
        responseStream << line << endl;
        contactLine = line;
    } else if (line.find("Content-Length:")) {
        responseStream << line << endl << endl;
    } else if (line.find("Contact:")) {
        responseStream << line << endl;
    } else if (line.find("Content-Length:")) {
        responseStream << line << endl << endl;
    }

    // Prepare contact data, will be needed by SipServer
    if (!splitContactData(contactLine)) {
        log->debug("Error reading contact data from: " + contactLine);
        sendServerError();
        return;
    }

    // Everything ok -> send response to requestor
    string response = responseStream.str();
    log->debug("Sending REGISTER ok to requestor: + " + response);
    int bytesTransferred = serverSocket->SendDatagram(response.c_str(), response.size(), "callerIPSource, callerPort");
    log->debug(" - sent bytes: ", bytesTransferred);
// Publish the values in the kademlia network
log->debug("Publishing the user data in kademlia...");
Database* db = Database::getInstance();
string name = responseContact[0];
string data = name + "@" + responseContact[1];
if (responseContact[2] != "") {
    data += ":" + responseContact[2]; // if port is not default
} 
int countPublished = db->put(name, data);
if (countPublished < 1) {
    log->debug("ERROR: Could not publish: " + data);
}
log->debug("Processing REGISTER ok :)");
}

void SipProcessor::processInvite() {
    log->debug("Entering processInvite...");
    int bytesTransferred = 0;

    // First send trying
    string tryingMessage = createTryingMessage();
    log->debug("Sending TRYING to caller...");
    log->debug("Trying message:
" + tryingMessage);
    bytesTransferred = serverSocket->SendDatagram(tryi
    ngMessage.c_str(), tryingMessage.size(), *callerIPSource, callerPort);
    log->debug(" - sent bytes: ", bytesTransferred);

    // Prepare invite request with data from kademlia
    string inviteRequest = prepareInvite();
    if (inviteRequest == "") {
        // "" indicates error
        // Error processing is done in prepareInvite() so we just return.
        return;
    }

    // Get the destination where to forward the message
    string name = responseContact[0];
    string ip = responseContact[1];
    string port = responseContact[2];
    if (port == "") {
        port = "5060"; // Def SIP port
    }
    int remotePort = atoi(port.c_str());

    // Create the socket to callee
    calleeSocket = new UDPConnectedSocket(false);
    calleeIPSourc
    e = ip.c_str();
    log->debug("Creating socke

    // Forward INVITE to callee
    string tempIP(calleeSocket->RemoteIPAddress().GetAd
    dressString());
log->debug("Forwarding INVITE Message to: IP = "+ tempIP + ", port = ", calleeSocket->RemotePortNumber());
bytesTransferred = calleeSocket->SendDatagram(inviteRequest.c_str(), inviteRequest.size());
log->debug("- sent bytes: ", bytesTransferred);
char response[BUFFER_SIZE];
response[0] = 0;

// Wait for TRYING
log->debug("Waiting for TRYING message");
bytesTransferred = calleeSocket->ReceiveDatagram(response, BUFFER_SIZE);
log->debug("- received bytes: ", bytesTransferred);
string returnResponse(response);
log->debug("Response:\n" + returnResponse);

// Forward TRYING to original requestor
bytesTransferred = serverSocket->SendDatagram(returnResponse.c_str(), returnResponse.size(), *callerIPSource, callerPort);
log->debug("TRYING forwarded...");
log->debug("- sent bytes: ", bytesTransferred);

// Wait for RINGING
log->debug("Waiting for RINGING message...");
bytesTransferred = calleeSocket->ReceiveDatagram(response, BUFFER_SIZE);
log->debug("- received bytes: ", bytesTransferred);
returnResponse = response;
log->debug("Response:\n" + returnResponse);

// Forward the RINGING to original requestor
bytesTransferred = serverSocket->SendDatagram(returnResponse.c_str(), returnResponse.size(), *callerIPSource, callerPort);
log->debug("RINGING forwarded...");
log->debug("- sent bytes: ", bytesTransferred);

// Wait for OK
log->debug("Waiting for OK message...");
bytesTransferred = calleeSocket->ReceiveDatagram(response, BUFFER_SIZE);
log->debug("- received bytes: ", bytesTransferred);
returnResponse = response;
log->debug("Response:\n" + returnResponse);

// Forward the OK to original requestor
bytesTransferred = serverSocket->SendDatagram(returnResponse.c_str(), returnResponse.size(), *callerIPSource, callerPort);
log->debug("OK forwarded...");
log->debug("- sent bytes: ", bytesTransferred);

/*
 // Wait for ACK from original requestor
 log->debug("Waiting for ACK message...");
 bytesTransferred = serverSocket->ReceiveDatagram(response, BUFFER_SIZE, *callerIPSource, callerPort);
 log->debug("- received bytes: ", bytesTransferred);
 returnResponse = response;*/
void SipProcessor::processAck() {
    log->debug("Entering processAck...");
    /*
     * Copy needed lines from request to response
    */
    int bytesTransferred;
    istringstream is(originalRequest);
    string line;
    string contactLine;

    // Forward the ACK to the found address
    string name = responseContact[0];
    string ip = responseContact[1];
    string port = responseContact[2];
    if (port == "") {
        port = "5060"; // Def SIP port
    }
    int remotePort = atoi(port.c_str());

    // Create the socket to callee
    calleeSocket = new UDPConnectedSocket(false);
    calleeIPSource = ip.c_str();
    log->debug("Creating socket to callee, IP=", calleeIPSource, ", port=", remotePort);
    calleeSocket->Connect(calleeIPSource, remotePort);

    // Forward INVITE to callee
    string tempIP(calleeSocket->RemoteIPAddress().GetString());
log->debug("Forwarding ACK Message to: IP = " + tempIP + ", port = ", calleeSocket->RemotePortNumber());
    bytesTransferred = calleeSocket->SendDatagram(originalRequest.c_str(), originalRequest.size());
    log->debug(" - sent bytes: ", bytesTransferred);
}

void SipProcessor::sendServerError() {
    string response = "SERVER ERROR"
    serverSocket->SendDatagram(response.c_str(), response.size(), *callerIPSource, callerPort);
    log->debug("SERVER ERROR sent to caller.");
}

void SipProcessor::sendRequestNotSupported() {
    string response = "REQUEST NOT SUPPORTED"
    serverSocket->SendDatagram(response.c_str(), response.size(), *callerIPSource, callerPort);
    log->debug("REQUEST NOT SUPPORTED sent to caller.");
}

void SipProcessor::sendInvalidRequest() {
    string response = "INVALID REQUEST"
    serverSocket->SendDatagram(response.c_str(), response.size(), *callerIPSource, callerPort);
    log->debug("INVALID REQUEST sent to caller.");
}

void SipProcessor::sendNotFound() {
    string response = "NOT FOUND"
    serverSocket->SendDatagram(response.c_str(), response.size(), *callerIPSource, callerPort);
    log->debug("NOT FOUND sent to caller.");
}

const string SipProcessor::prepareInvite() {
    istringstream is(originalRequest);
    string line;

    // First iterate over the request, to find the <To: XXX> line
    // When found, search in Kademlia network for real data
    // Then iterate over the request again, replace the old
    // To data with the one found in kademlia network

    for (getline(is, line); line != ""); getline(is, line) {
        if (line.find("To:")) == 0) {
            if (!splitContactData(line)) {
                log->debug("Error reading contact data from line: " + line);
                sendInvalidRequest();
                return "";
            }
        }
    }

    return "";}
// Now search for data in kademlia
Database* api = Database::getInstance();
string foundData = api->get(responseContact[0]);
if (foundData == "") {
    log->debug("Name "+ responseContact[0] + ": not found in kademlia!");
    sendNotFound();
    return "";
}

int index = foundData.find("@");
responseContact[0] = foundData.substr(0, index); // name
responseContact[1] = foundData.substr(index + 1); // ip + port

index = responseContact[1].find(":");

if (index == -1) {
    responseContact[2] = "5060"; // def port
} else {
    responseContact[2] = responseContact[1].substr(index + 1); // port
    responseContact[1] = responseContact[1].substr(0, index); // IP adr
}

// The data should be prepared. Now generate the response
ostringstream response;
is.str(originalRequest);

// Copy needed lines from request to response
int emptyLines = 0;
for (getline(is, line); emptyLines < 2; getline(is, line)) {
    if (line == "") {
        // Count empty lines, after two of them we are done
        response << line << endl;
        emptyLines++;
    } else if (line.find(INVITE) == 0) {
        // Replace values in header
        response << INVITE << " sip:" << responseContact[0] << " SIP/2.0" << endl;
    } else if (line.find("To:")) {
        // Replace values in To: field
        response << "To: <sip:" << responseContact[0] << ":" << responseContact[2] << ">" << endl;
    } else {
        // Else just copy the line
        response << line << endl;
    }
}

response << endl;
return response.str();

const string SipProcessor::createTryingMessage() {
istringstream is(originalRequest);
ostringstream responseStream;
string line;

responseStream << SIP_HEADER_TRYING << endl;
// Copy needed lines from request to response
for (getline(is, line); line != ""; getline(is, line)) {
    if (line.find("Via:" ) == 0) {
        responseStream << line << endl;
    } else if (line.find("To:" ) == 0) {
        responseStream << line << endl;
    } else if (line.find("Contact:" ) == 0) {
        responseStream << line << endl;
    } else if (line.find("From:" ) == 0) {
        responseStream << line << endl;
    } else if (line.find("Call-ID:" ) == 0) {
        responseStream << line << endl;
    } else if (line.find("Server:" ) == 0) {
        responseStream << line << endl;
    } else if (line.find("Content-Length:" ) == 0) {
        responseStream << "Content-Length: 0" << endl << endl;
}
// Everything ok -> return created message;
return responseStream.str();
}

const string SipProcessor::createRedirectionMessage() {
    istringstream is(originalRequest);
    string line;

    string foundData = "";
    // First search the for the TO field, and get data
    for (getline(is, line); line != ""; getline(is, line)) {
        if (line.find("To:" ) == 0) {
            if (!splitContactData(line)) {
                log->debug("Error reading contact data from line: " + line);
                sendInvalidRequest();
                return "";
            }
            // Now search for data in kademlia
            Database* api = Database::getInstance();
            log->debug("Searching for " + responseContact[0]);
            foundData = api->get(responseContact[0]);
            log->debug("Found " + foundData);
            if (foundData == "") {
                log->debug("Name " + responseContact[0] + " not found in kademlia!");
                sendNotFound();
                return "";
            }
        }
    }
}
log->debug("Data found, prepare redirection");
istringstream is2(originalRequest);
ostringstream responseStream;
responseStream << SIP_HEADER_MOVED_TEMPORARILY << endl;
for (getline(is2, line); line != ""; getline(is2, line)) {
    if (line.find("Via:")) == 0) {
        string tempLine = "Via: " + line;
        responseStream << tempLine << endl;
        responseStream << line << endl;
    } else if (line.find("To:")) == 0) {
        string tempLine = "To: " + line;
        responseStream << tempLine << endl;
        responseStream << line << endl;
    } else if (line.find("From:")) == 0) {
        responseStream << line << endl;
    } else if (line.find("Contact:")) == 0) {
        string tempLine = "Contact:" + line;
        responseStream << tempLine << endl;
        responseStream << line << endl;
    } else if (line.find("Call-ID:")) == 0) {
        responseStream << line << endl;
    } else if (line.find("CSeq:")) == 0) {
        responseStream << line << endl;
    } else if (line.find("Content-Length:")) == 0) {
        string tempLine = "Content-Length: 0";
        responseStream << tempLine << endl;
    }
}

// Everything ok -> return created message;
log->debug("Exiting...");
return responseStream.str();

/**
 * Prepare contact data
 */
bool SipProcessor::splitContactData(const string& cl) {
    string contactLine = cl;
    int index = contactLine.find("<sip:");
    int index2 = contactLine.find">
    if (index == -1 || index2 == -1) {
        return false;
    }
    contactLine = contactLine.substr(index + 5, index2 - index - 5);
    index = contactLine.find("@");
    if (index == -1) {
        return false;
    }

    string name = contactLine.substr(0, index);
    string ipAddress = contactLine.substr(index + 1);
    string port = "";
    index = ipAddress.find(:);
if (index != -1) {
    port = ipAddress.substr(index+1);
    ipAddress = ipAddress.substr(0, index);
}

responseContact[0] = name;
responseContact[1] = ipAddress;
responseContact[2] = port;
log->debug("PORT = " + port);
return true;
}

/**
 * Gets current date and time in format prepared for SIP.
 */

const string SipProcessor::getDateAndTime() {
    time_t rawtime;
    struct tm * timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    char* now = asctime(timeinfo);
    string nfdt(now);

    string fdt;

    fdt += nfdt.substr(0, 3) + " " ; // day in week
    fdt += nfdt.substr(8, 2) + " " ; // date
    fdt += nfdt.substr(4, 3) + " " ; // Month
    fdt += nfdt.substr(20, 4) + " " ; // year
    fdt += nfdt.substr(11, 8) + " GMT";

    return fdt;
}

8.1.13 sip2p.ini

# INI FILE USED FOR CREATING SERVER FOR X-LITE

S2P_SERVER_IP=<ip address>
S2P_SERVER_PORT=5060;
KADC_INI_FILE=kadc.ini
DEBUG=TRUE
DUMMY_PUBLISH=TRUE
DUMMY_FAST_SEARCH=TRUE

[DUMMY_P2P_USERS]
test=test@<ip address>
#localx=konj@<ip address>:5062
8.1.14 Makefile

.SUFFIXES:
.SUFFIXES: .cpp .o

CC = g++
LD = g++
OUTPUT = dist/sip2p
OBJECTS = Logger.o KadCppApi.o Database.o SipProcessor.o SipServer.o sip2p.o
CFLAGS = -c
VPATH = src
HEANDERS_DIR = include
ARCHIVE_DIR = lib

#LIBS= -lpthread -lz -lKadC -lsocketcc
LIBS= -lpthread -lz lib/KadC.a /usr/local/lib/libsocketcc.so

all: $(OBJECTS)
  $(LD) $(OBJECTS) -L$(ARCHIVE_DIR) $(LIBS) -o $(OUTPUT)

%.o : %.cpp %.h
  $(CC) -I$(HEADERS_DIR) $(CFLAGS) $<

clean:
  rm -rf dist
  rm -f *.o