CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Universidad Carlos III de Madrid e-Archivo

Fachhochschule Braunschweig/Wolfenbüttel
University of Applied Sciences

Department of Electrical Engineering

Final Project
SNOM Provisioning

Elisa Martín-Caro Zapardiel

Mentor: Prof. Diedrich Wermser
Supervisor: Michael Iedema

July 2009

This is to certify that except where specific referance is made, the work described in this project is result of the candidate. Neither this project,nor any part of it, has been presented or is currently submitted in candidature for any degree at another University

Candidate: Elisa Martín-Caro Zapardiel

Date: 07-09-2009

Table of Contents

Table of Figures

Abstract

The Internet was not designed in the beginning, to be used as a real-time transportation medium. Today, it is used to send any kind of service. My project is to add an application to AskoziaPBX that provisions SNOM phones. I use the protocol as a transport medium for creating, sending, and termination of communication with the SNOM phone is SIP protocol.

Asterisk is a free software application that provides the functionality of a PBX telephone exchange. Like any PBX, you can connect a number of phone calls to each other and even connect to VoIP providers.

The most interesting aspect of Asterisk is that it provides many VoIP protocols such as SIP, H.323, IAX, etc.

AskoziaPBX is a very comprehensive system based on Asterisk.

My job is to automate the provisioning of a SNOM 360 phone (telephones being used for this project) via SIP multicast. We can say that SIP is today one of the safer and better protocols for VoIP. Finally, I should add AskoziaPBX with my applications

The purpose of this application is to make it simpler and faster for users to configure their phones.

Acknowledgement

I give my sincere thanks to Professor Dr.Wermser and my tutor Michael Iedema for having conducted this project, the time they have spent with me and for the facilities they have given me to develop it.

Also thanks to the whole team in the laboratory which has helped me overcome all the disadvantages that have arisen, especially in Matthias Bormann, who has suffered with me in the beginning of the project.

I can not forget to acknowledge my cotutor Julio Villena and my partner and friend Adriana Arroyo who not only they have helped with the project, but also with those hard years of studies.

And of course my family and friends although they are not here in these moments, but they always supported me and endured

1. Introduction

First, a brief introduction to Asterisk[10]. Asterisk is a Private Branch Exchange (PBX). A PBX can be thought of as a private phone switchboard, connecting to one or more telephones on one side, and usually connecting to one more telephone lines on the other.

AskoziaPBX is more than another GUI for Asterisk. It is embedded PBX solution which eases system upgrades, backups and provisioning.

I have worked with the SNOM phone 360. This phone was designed for maximum efficiency in the everyday business environment. Dedicated keys provide you with direct access to the functions for audio and call control.

SNOM phone use the SIP protocol to exchange frames, the advantages are: easy implementability, scalability, expandability and flexibility. SIP can be used to manage any number of sessions with one or several participants. However, it is not limited to Voice over IP as sessions can be any number of multimedia streams or conferences.

1.1 Objectives

The project aims to add a new page to Askozia PBX which implements a button to search for XML configuration files which will then be sent to the phone to configure it.

My objective is to provision SNOM phones via the SIP protocol, using C as the programming language.

So, once I've studied the SIP protocol and its workings. My pc should recognize when the phone sends a broadcast (frame SUBSCRIBE) and responds to this with a confirmation frame and sends the XML file (frame OK and frame NOTIFY). Which, if everything happened correctly will receive a confirmation from the phone (frame OK).

My programm has to be running Continuously as daemon. It listens at all times for the broadcast packet and responds once one is received.

Finally, it will be added to AskoziaPBX, allowing the uploading of SNOM phone.
This will be the same as all pages in AskoziaPBX and implementing a Browse button to search for XML files, those containing the new phone setup, and a Load button to apply changes.

It is way faster and easier to configure the phone.

2. Information search and study about SIP

2.1 SIP Protocol

2.1.1 Introduction

There are many applications of the Internet that require the creation and management of a session, where a session is considered an exchange of data between an association of participants. The implementation of these applications is complicated by the practices of participants: users may move between endpoints, they may be addressable by multiple names, and they may communicate in several different media – sometimes simultaneously. Numerous protocols have been authored that carry various forms of real-time multimedia session data such as voice, video, or text messages. The Session Initiation Protocol (SIP) works in concert with these protocols by enabling Internet endpoints (called user agents) to discover one another and to agree on a characterization of a session they would like to share. For locating prospective session participants, and for other functions, SIP enables the creation of an infrastructure of network hosts (called proxy servers) to which user agents can send registrations, invitations to sessions, and other requests. SIP is an agile, general-purpose tool for creating, modifying, and terminating sessions that works independently of underlying transport protocols and without dependency on the type of session that is being established.[15]

2.1.2 Components

The SIP elements, that is, user agent clients and servers, stateless and stateful proxies and registrars, contain a core that distinguishes them from each other.

- **User agents:** A user agent is a SIP-enabled endevice. The user agent provides communication services to a user. A user agent client (UAC) is a functional module issuing a request and a user agent server (UAS) is a functional module responding to a request. In VoIP, a UAC is a calling party and a UAS is a called target party.

- **SIP server:** A SIP server supports communication establishment between a UAC and a UAS. Three types of SIP servers exist. A proxy server transfers a SIP request, which a UAC issues to a UAS and other servers.

2.1.3 Messages

SIP is a text-based protocol with syntax similar to that of HTTP. There are two different types of SIP messages: requests and responses. The first line of a request has a *method*, defining the nature of the request, and a Request-URI, indicating where the request should be sent. The first line of a response has a *response code*.

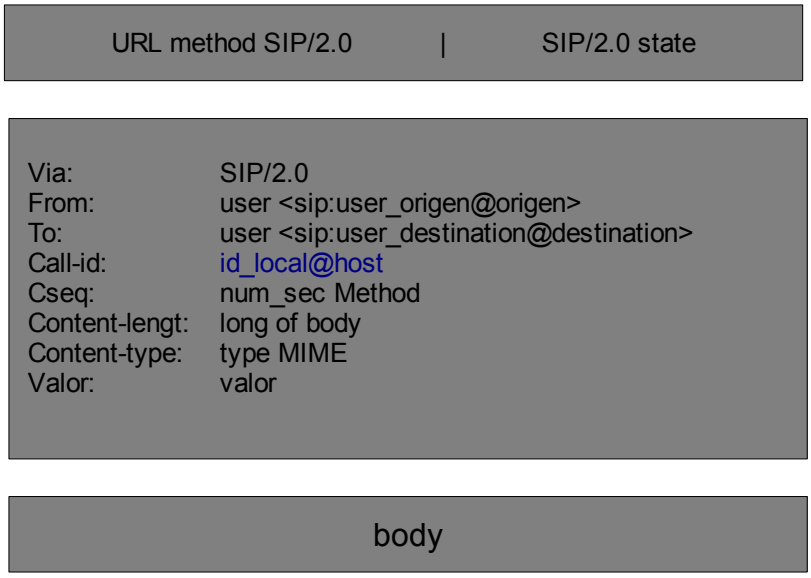

Figure 1: SIP header

The first line of the header indicates the type of request or response. The five headers Via, From, To, Call-ID and CSeq are present in all messages and identify each message in a unique way within each call. Its meaning is as follows:

- Via: Indicates the transport used for shipping and identifies the route of the request.
- From: Indicates the direction of the origin of the request.
- To: Indicates the address of the recipient of the request.
- Call-ID: Unique identifier for each call and contains the host address.
- Cseq: It begins with a random number and identifies each request in sequence.

For SIP requests,RFC 3261defines the following methods:

- REGISTER: Used by a UA to notify its current IP address the URLs for which it would like to receive calls.
- INVITE: Used to establish a media session between user agents.
- ACK: Confirms reliable message exchanges.
- CANCEL: Terminates a pendind request
- BYE: Terminates a session between two users in a conference.
- OPTIONS: Requests information about the capabilities of a caller, without setting up a call.

The SIP response types defined in RFC 3261 fall in one of the following categories:

- Provisional (1xx): Request received and being processed.
- Success (2xx): The action was successfully received, understood, and accepted.
- Redirection (3xx): Further action needs to be taken (typically by sender) to complete the request.
- Client Error (4xx): The request contains bad syntax or cannot be fulfilled at the server.
- Server Error (5xx): The server failed to fulfil an apparently valid request.
- Global Failure (6xx): The request cannot be fulfilled at any server.

2.1.4 Specific event notification

The ability to request asynchronous notification of events proves useful in many types of SIP services for which cooperation between end-nodes is required. Examples of such services include automatic callback services (based on terminal state events), buddy lists (based on user presence events), message waiting indications (based on mailbox state change events), and PSTN and Internet Internetworking (PINT) status (based on call state events).

The general concept is that entities in the network can **subscribe** to resource or call state for various resources or calls in the network, and those entities (or entities acting on their behalf) can send notifications when those states change.

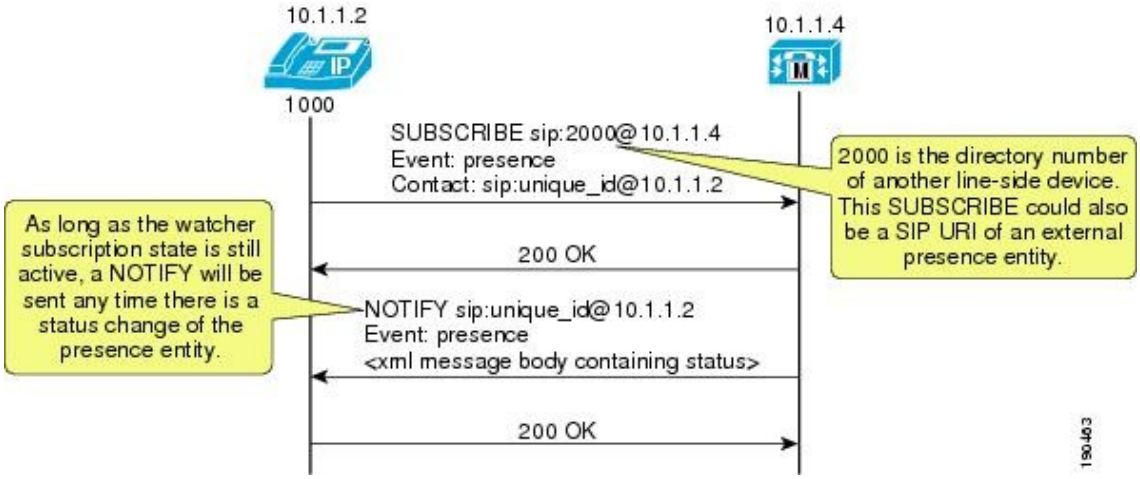

Figure 2: SIP Communication (SUBSCRIBE). [17] www.cisco.com

Identification of events is provided by three pieces of information: Request URI, Event Type, and (optionally) message body. The Request URI of a SUBSCRIBE request, most importantly, contains enough information to route the request to the appropriate entity per the request routing procedures outlined in SIP. It also contains enough information to identify the resource for which event notification is desired, but not necessarily enough information to uniquely identify the nature of the event (e.g., "sip:unique_id@10.1.1.2" would be an appropriate URI to subscribe to for my presence state; it would also be an appropriate URI to subscribe to the state of my voice mailbox).

Subscribers MUST include exactly one "**Event**" header in SUBSCRIBE requests, indicating to which event or class of events they are subscribing. The "Event" header will contain a token which indicates the type of state for which a subscription is being requested. This token will be registered with the IANA and will correspond to an event package which further describes the semantics of the event or event class. The "Event" header MAY also contain an "id" parameter. This "id" parameter, if present, contains an opaque token which identifies the specific subscription within a dialog. An "id" parameter is only valid within the scope of a single dialog.

The subscriber can expect to receive a NOTIFY message from each node which has processed a successful subscription or subscription refresh. Until the first NOTIFY mssage arrives, the subscriber should consider the state of the subscribed resource to be in a neutral state. Documents which define new event packages MUST define this "neutral state" in such a way that makes sense for their application.

Due to the potential for both out-of-order messages and forking, the subscriber MUST be prepared to receive NOTIFY messages before the SUBSCRIBE transaction has completed.

2.1 SNOM telephone

2.1.1 Auto provisioning

Auto provisioning is a feature implemented proprietarely in the standard firmware of all SNOM 3xx VoIP phones. Auto provisioning allows remote administration (configuration and maintenance) of unlimited number of distinct SNOM phone types. Can be used to provide general and specific configuration parameters ("Settings") to the phones and to manage firmware actualization. Depending on the phone installation environment five scenarios can be distinguished how to provide the setting (provisioning) URL to the phones:

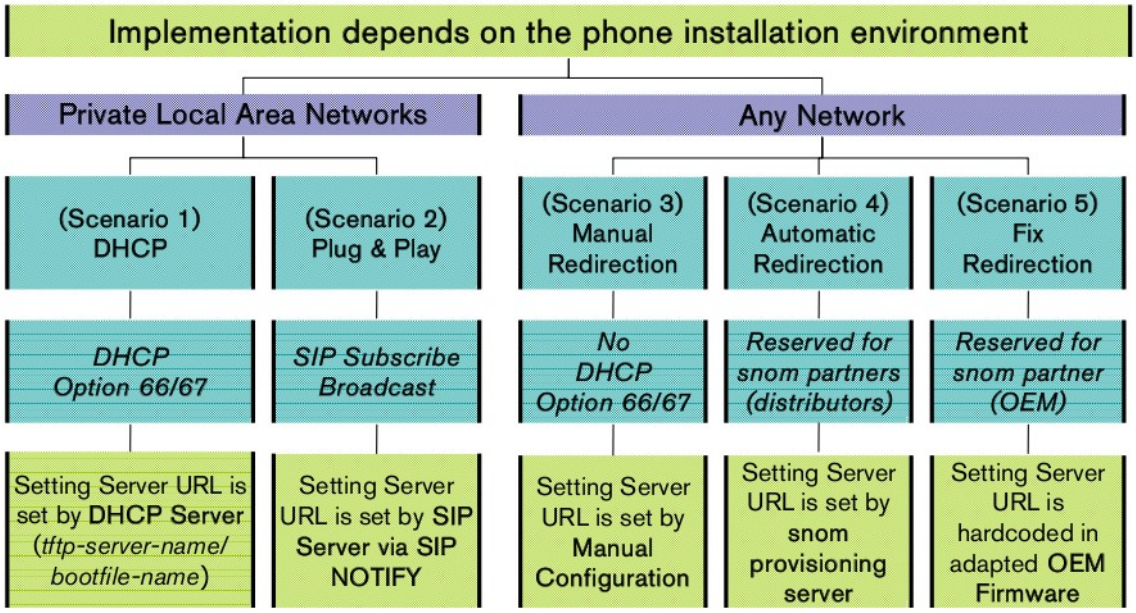

Figure 3: Phone installation types. [14]: Snom VoIP phones. Page 5

I will focus on Scenario 2, **Plug and Play**. Which is the case with this project.

2.2.2 Settings

There are currently more than about 300 configuration parameters available. Retrieve a complete list from the Web User Interface:

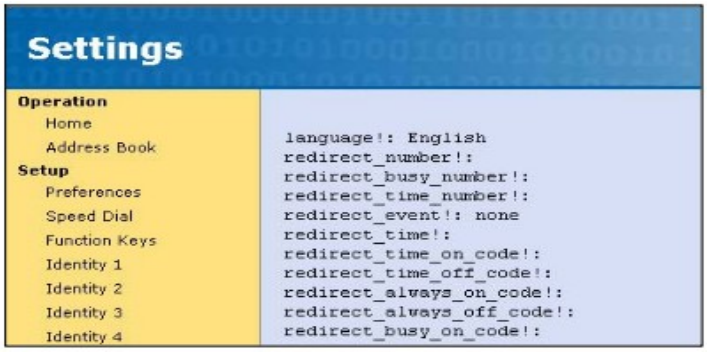

Figure 4: Settings page. [6]

From firmware version 7 onwards the complete list of configuration parameters can be saved either in text or XML format by clicking the appropriate link on the top of the "Settings" page.

Below shows an XML schema with the required parameters for setting:

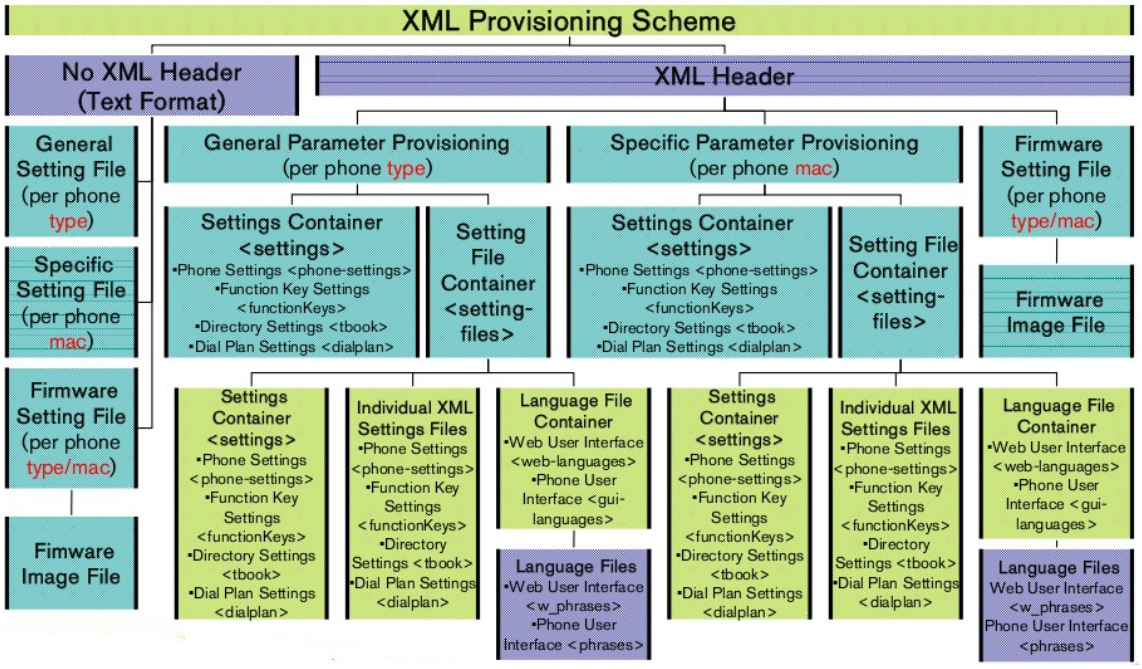

Figure 5: XML Provisioning Scheme. [14]: Snom VoIP phones. Page 30

One general settings file container <settings-files> per phone type 360providiong a list of setting file URLs linked to:

1. **One "settings container (<settings>)" per phone type** containing **general** configuration parameters grouped in XML tags (<phone-settings>, <functionKeys>, <tbook>, <dialplan>) or/and **individual XML Settings Files per phone type** containing **general** configuration parameters:(Phone settings setting file , Function key setting file , Directory setting file , Dial plan setting file).

2. **One "Phone user interface language file container" per phone type** with a list of phone user interface language file URLs

3. **One "Web user interface language file container" per phone type** with a list of web user user interface language file URLs

3. Implementation

3.1 Introduction

Once my research was completed, I could start with application development. I've one server *141.41.40.77/~elisa* where all my files are saved. My server is separated in to two folders. One folder *(pagweb)* has web page files and my executable which makes the application run. Second folder (*sipphone*) has the XML file with the phone configuration.

Therefore, the absolute path used to load my website is:
http:///141.41.40.77/~elisa/pagweb/index.php.

And the Provisioning URI that used in the SIP exchange is:
http:///141.41.40.77/~elisa/sipphone/settings.xml?mac={mac}

3.2 SIP Communication

SIP itself is not a transport protocol. It relies on other protocols to carry it from element to element. SIP was designed to allow almost any transport protocol to be used. Currently, the specifications define how to carry SIP using the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP) .

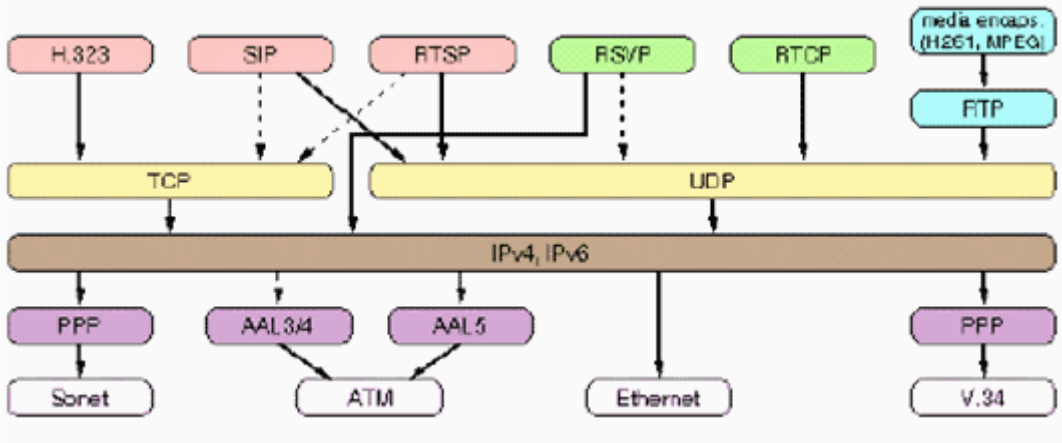

Figure 6: Signaling protocol architecture. [16]

I've chosen to work with the UDP protocol, because UDP is simple, straightforward and is not connection oriented. It is more fault tolerant and usually uses less bandwidth. Also it produces fewer delays and buffer usage in devices is lower. It is difficult to implement security measures on UDP and is therefore more vulnerable to attacks. It also presents more problems to get through routers that use NAT. At first, all the software and hardware implemented SIP over UDP.

So, I create UDP sockets with port 5060 corresponding to SIP protocol. Before, I make a daemon who is always alive to receive broadcast requests.
Then, we can see a picture of the exchange of frames between the phone (141.41.40.131) and my PC (141.41.40.184).

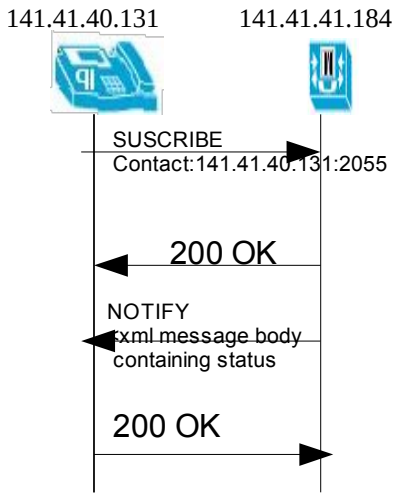

Figure 7: SIP Communication (SUBSCRIBE)

Here my results exchanging frames between the SNOM telephones and my pc can be seen:

Message 1. SNOM phone send on boot-up a SIP ***SUBSCRIBE*** message to a **multicast address**:

```
Sent to udp:224.0.1.75:5060   Broadcast:224.0.1.75

SUBSCRIBE sip:MAC%3a00041329882C@voip.ikt-bs.de SIP/2.0
Via: SIP/2.0/UDP 141.41.40.131:2055;rport
From: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=1023721533
To: <sip:MAC%3a00041329882C@voip.ikt-bs.de>
Call-ID: 1936281451@141.41.40.131
CSeq: 1 SUBSCRIBE
Event: ua-profile;profile-
type="device";vendor="snom";model="snom360";version="7.3.14"
Expires: 0
Accept: application/url
Contact: <sip:141.41.40.131:2055>
Content-Length: 0
```

Figure 8: SUBSCRIBE frame

Message 2. The notifier processes the subscription request and creates a new subscription. A ***200 OK*** response is sent to confirm the subscription.

```
Received from udp:141.41.40.131:5060

SIP/2.0 200 OK
Via: SIP/2.0/UDP 141.41.40.131:2055;rport
Contact: <sip:141.41.40.184:5060>
To: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=12344321
From: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=1023721533
Call-ID: 1936281451@141.41.40.131
CSeq: 1 SUBSCRIBE
Expires: 0
Content-Length: 0
```

Figure 9: Response 200 OK

Message 3. In order to complete the process, the notifier sends the subscriber a ***NOTIFY*** with the current state of the resource. SIP NOTIFY message containing the Auto Provisioning URL

```
Received from udp:141.41.40.184:5060

NOTIFY sip:141.41.40.131:2055 SIP/2.0
Via: SIP/2.0/UDP 141.41.40.184:5060
Max-Forwards: 70
Contact: <sip:141.41.40.184:2055>
From: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=12344321
To: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=1023721533
Call-ID: 1936281451@141.41.40.131
CSeq: 2 NOTIFY
Content-Type: application/url
Subscription-State: terminated;reason=timeout
Event: ua-profile;profile-
type="device";vendor="snom";model="snom360";version="7.4.19"
Content-Length:58

http://141.41.40.77/~elisa/sipphone/settings.xml?mac={mac}
```

Figure 10: NOTIFY

In the body of the frame is the provisioning URI
http://141.41.40.77/~elisa/sipphone/settings.xml?mac={mac}

The file with the new configuration is settings.xml and it is on my server, *elisa@141.41.40.77* .

Message 4. Finally, the subscriber confirms receipt of the NOTIFY request with ***200 OK***

```
Sent to udp:141.41.40.184:5060

SIP/2.0 200 Ok
Via: SIP/2.0/UDP 141.41.40.184:5060
From: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=12344321
To: <sip:MAC%3a00041329882C@voip.ikt-bs.de>;tag=1023721533
Call-ID: 1936281451@141.41.40.131
CSeq: 2 NOTIFY
Content-Length: 0
```

Figure 11: 200 OK. The communication was successful

This is a Wireshark capture which shows it working correctly:

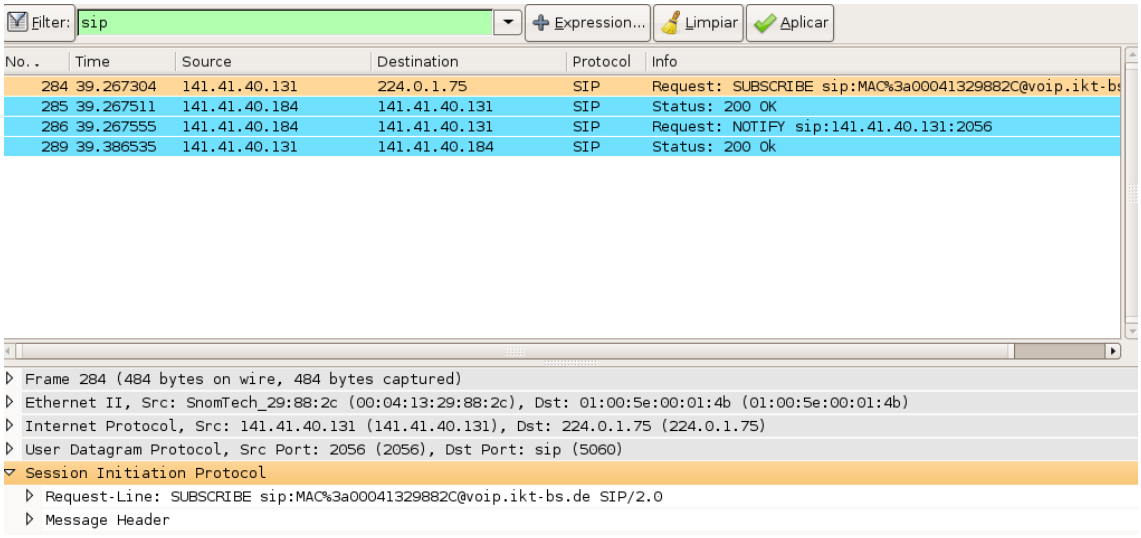

Figure 12: Capture with Wireshark to show the SIP broadcast comunication

3.3 Web Application

The second part of my job is to create a new page to be added in AskoziaPBX which allows the uploading of SNOM phone configuration files. The new page has the same layout of AskociaPBX.

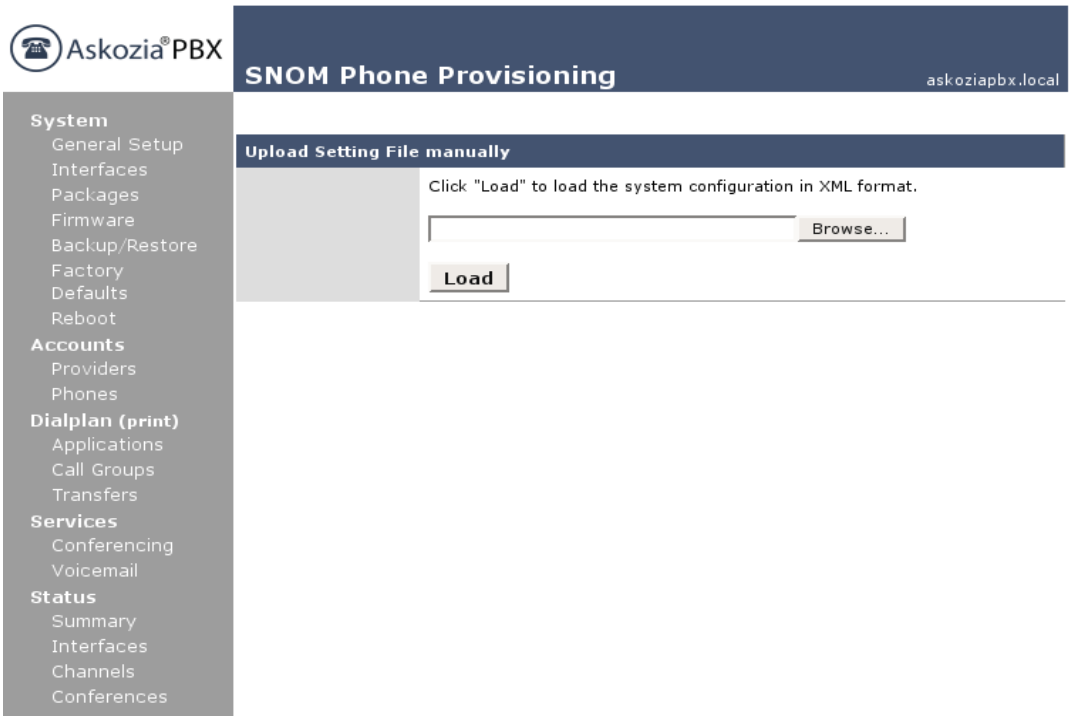

Figure 13: Presentation of the page.

Note: The work is done on a demo, then be integrated into the current page AskoziaPBX

When you click on *browse...* you look for the SNOM phone configuration file. The file should be *XML*. Then, you click load, and the file is uploaded and saved on my server with name the *settings.xml*, so I always can use the following Auto Provisioning URL:

```
http://141.41.40.77/~elisa/sipphone/settings.xml?mac={mac}
```

Finally, while the configuration is processing, a progress bar (red line) appears.

Figure 14: The configuration is processing

When the configuration is finished, this image is shown, with the button “comeback”. If you want to do a new configuration, you can.

Figure 15: Come back home

4. Results

Result of the project is the SNOM 360 phone configuration. Here is an example, the SNOM phone previously was configured with the English language:

```
<language perm="">English</language>
```

And I'm going to change it to Spanish

```
<language perm="">Spanisch</language>
```

This first figure shows how the phone we are working with is setup.

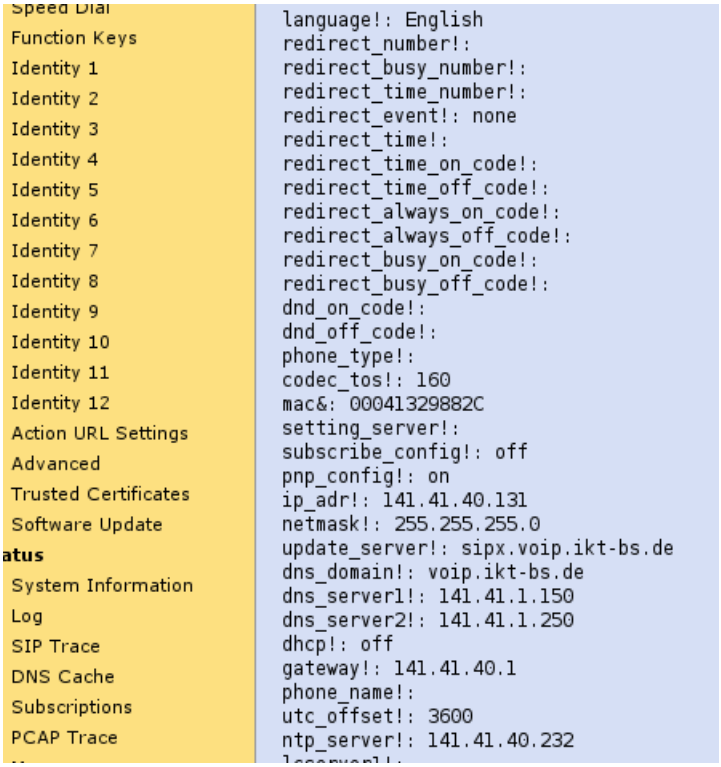

Figure 16: Before settings configuration. [6]

SNOM Phone 360, settings:

```
@ip : 141.41.41.131
mask: 255.255.255.0
server: 141.41.40.232
... ... ...
```

We can see that the default language is English.

First parameter:

```
language:! English
```

Now, to run my program. It is listening to a broadcast request from my phone, then reboots my phone to save time. The steps to reboot the phone are: from page 141.41.40.131 (server phone) click on advanced, update, reboot, and confirm:

Figure 17: Reboot phone. [6]

Then, I selected the XML file that I want to send, for example, the same configuration file from the phone, just change the language. This is a part of the XML file that I send, changing the language parameter to Spanish:

```
<?xml version="1.0" encoding="utf-8"?>
<settings>
<phone-settings e="2">
<language perm="">Spanish</language>
<redirect_number perm=""></redirect_number>
<redirect_busy_number perm=""></redirect_busy_number>
<redirect_time_number perm=""></redirect_time_number>
<redirect_event perm="">none</redirect_event>
 ... ... ...
<pnp_config perm="">on</pnp_config>
<ip_adr perm="RW">141.41.40.131</ip_adr>
<netmask perm="RW">255.255.255.0</netmask>
<update_server perm="RW">sipx.voip.ikt-bs.de</update_server>
<dns_domain perm="RW">voip.ikt-bs.de</dns_domain>
<dns_server1 perm="RW">141.41.1.150</dns_server1>
<dns_server2 perm="RW">141.41.1.250</dns_server2>
<dhcp perm="">off</dhcp>
<gateway perm="RW">141.41.40.1</gateway>
<phone_name perm=""></phone_name>
<utc_offset perm="">3600</utc_offset>
<ntp_server perm="RW">141.41.40.232</ntp_server>
... .... ....
</settings>
```

When I receive the broadcast request, my program establishes SIP communication with the phone and sends the file with the new configuration. And, again we see the server's phone page and note that the language has been modified

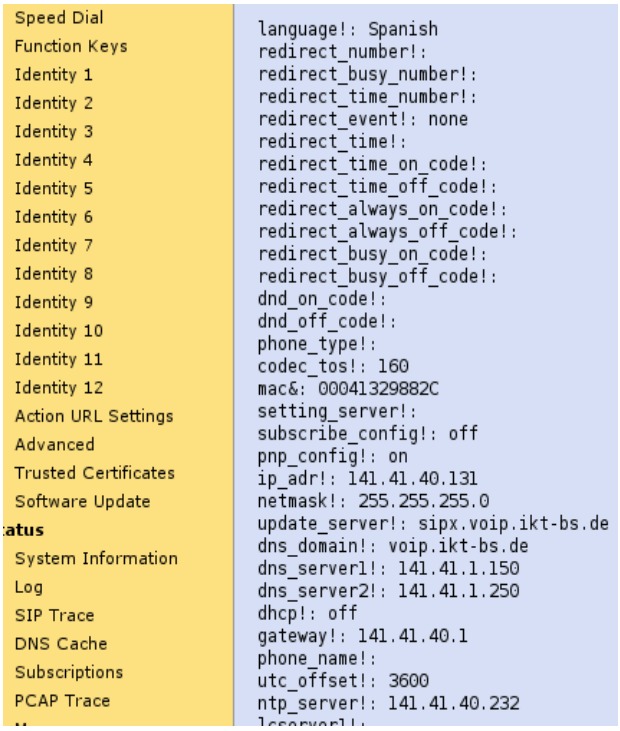

Figure 18: After setting configuration. [6]

Here, can see the modification, *language!: Spanish*

The phone, of course, is the same: SNOM Phone 360, settings:

@ip : 141.41.41.131
mask: 255.255.255.0
server: 141.41.40.232
... .

Note: The project will be integrated into the base system AskoziaPBX version 2.0 : https://wush.net/trac/askozia/ticket/20

5. Conclusions and Future Work

As I said at the beginning of this document, the main objective of this project is to expand AskoziaPBX with a page that implements SNOM 360 phone provisioning.

For the preparation of this project a comprehensive study was done to find out what is the best way to achieve the objectives. And so, through SIP over UDP communication.

Personal experience, I've been very happy with this project. I greatly expanded my knowledge and the theme of the project seems interesting, because he telephone and Internet is a topical and important issue today with many improvements.

In the future as an extension of work: it is possible to configure several phones at once? That is, in my case it is a only one phone. Respecting always the IP addresse of each.

This way you can save time configuring 100 phones at once, and no to one to one.

List of abbreviatures

- SIP: **S**ession **I**nitiation **P**rotocol
- IP: **I**nitiation **P**rotocol
- PBX: **P**rivate **b**ranch **e**xchange
- IAX: **I**nter-**A**sterisk e**X**change
- UDP: **U**ser **D**atagram **P**rotocol
- TCP: **T**ransmission **C**ontrol **P**rotocol
- HTTP: **H**yper**t**ext **T**ransfer **P**rotocol
- UAS: **U**ser **A**gent **S**erver
- UAC: **U**ser **A**gent **C**lient
- URL: **U**niform **R**esource **L**ocator
- XML: E**x**tensible **M**arkup **L**anguage
- VoIP: **V**oice **o**ver **I**nternet **P**rotocol
- @ip: ip addrese

References

[1] http://systhread.net/texts/200508cdaemon2.php

[2] http://ntrg.cs.tcd.ie/undergrad/4ba2/multicast/antony/example.html

[3] http://wiki.snom.com/Category:Setting

[4] http://wiki.snom.com/Features/Mass_Deployment

[5] http://wiki.snom.com/wiki/index.php?title=Features/Mass_Deployment/PnP

[6] http://141.41.40.131/settings.htm

[7] http://tools.ietf.org/html/rfc3261

[8] http://www.ietf.org/rfc/rfc3265.txt

[9] http://www.voip-info.org/wiki/view/SIP

[10] D.Gomillon, B.Dempster. Building Telephony Systems with Asterisk. Packt Publishing Ltd. ISBN 1904811159. September 2005

[11] Syed A. Ahson and Mohammad Ilyas. SIP Handbook: Services, Technologies and Security of Session Initiation Protocol. Publisher: Taylor & Francis, Inc. Pub. ISBN-13: 9781420066036. November 2008. Chapter 5: SIP Event Notification and Presence Information.

[12] http://www.voipforo.com/SIP/SIPcomponentes.php

[13] http://en.wikipedia.org/wiki/Session_Initiation_Protocol#SIP_network_elements

[14] www.snom.com. SNOM VoIP phones. Mass Deployment, SNOM advanced training.

[15] SIP Communication Software Protocol Session Initiation Protocol . VOCAL Technologies, Ltd.2004. Custom Product Design Division 200 John James Audubon Parkway . Pages 1-2.

[16] http://www.monografias.com/trabajos33/telecomunicaciones/telecomunicaciones3.shtml#bibl

[17] http://www.cisco.com/en/US/i/100001-200000/190001-200000/190001-191000/190463.jpg

Annex 1: Configure Plug&Play

To configure via Plug and Play, the following steps be must performed:

PnP is enabled by default:

Figure 19: Configuration Plug and Play. [14]: Snom VoIP phones. Page 79

The phone must send a SIP SUBSCRIBE message to a multicast address (224.0.1.75).
SIP servers which have membership to the group can respond to the SUBSCRIBE and send NOTIFY messages with the setting server HTTP URL in the body. The phone retrieves its settings from the URL specified, shaped:
"http://192.168.100.10/sipphone/sipphoneconfig.xml?mac={mac}"

Annex 2: My code

Let me explain the more relevant parts of my code. All code is written in a single file, listenerSIP.c, which listens to telephone calls and send the respective answers.

Initially there is a connection socket with UDP port 5060 which listens for SIP frames the broadcast address 224.0.1.75,

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
/*************cdaemon**************/
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <syslog.h>
#include <unistd.h>

#define DEFAULT_INTERVAL 3
#define DEFAULT_LOGFLAG 0
/*********************************/
#define HELLO_PORT 5060                 //SIP PORT 5060

#define HELLO_GROUP "224.0.1.75"   //broadcast SIP: 224.0.1.75
#define MSGBUFSIZE 448
    struct sockaddr_in addr;
    struct ip_mreq mreq;
    struct info_phone phone;
    int fd, nbytes,addrlen;
    char msgbuf[MSGBUFSIZE];

//UDP CONNECTION
struct sockaddr_in createSocket(void){
    u_int yes=1;
    /* create what looks like an ordinary UDP socket */
    if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
         perror("socket");
         exit(1);
    }
   /* allow multiple sockets to use the same PORT number */
   if (setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(yes)) < 0) {
      perror("Reusing ADDR failed");
      exit(1);
    }
  /* set up destination address */
    memset(&addr,0,sizeof(addr));
    addr.sin_family=AF_INET;
    addr.sin_addr.s_addr=htonl(INADDR_ANY); /* N.B.: differs from sender */
    addr.sin_port=htons(HELLO_PORT);
    /* bind to receive address */
    if (bind(fd,(struct sockaddr *) &addr,sizeof(addr)) < 0) {
       perror("bind");
       exit(1);
    }
```

```
    /* use setsockopt() to request that the kernel join a multicast group */
    mreq.imr_multiaddr.s_addr=inet_addr(HELLO_GROUP);
    mreq.imr_interface.s_addr=htonl(INADDR_ANY);
    if (setsockopt(fd,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq)) < 0)
    {
      perror("setsockopt");
      exit(1);
    }
return addr;
}
```

Then, I made the structure that I'm going to use with headers of the frame

```
//INFO PHONE
struct info_phone{
    char mac_addr[16];
    char ip_addr[16];
    char port[5];
    char via[64];
    char from[64];
    char to[64];
    char call_id[64];
    char cseq[64];
    char tagFrom[64];
    char fromSinTag[64];
    char event[108];
};
```

This flow diagram represents the infoPhone method, which is the method that receives data from the SUSCRIBE frame to save in the headers.

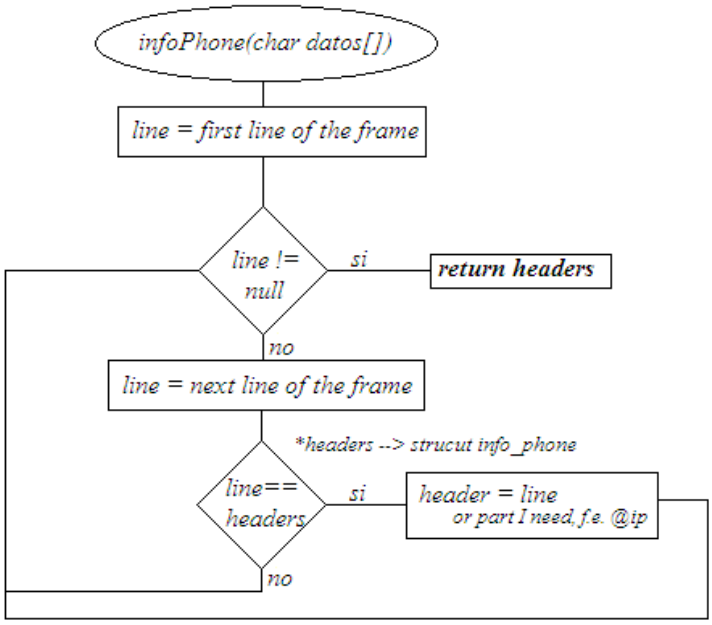

Figure 20: Flow diagram infoPhone method

The method sender fills the frame with the header and sends them

```
void sender( struct info_phone phone){
    char my_ip_addr [255];
    gethostname(my_ip_addr, sizeof(my_ip_addr)); //Return my ip address
    char ok_response[480]="SIP/2.0 200 OK\r\n";
    // If a phone has been recognized first send 200 OK
    if((struct info_phone *) &phone!=NULL){
        strncat(ok_response, phone.via, strlen(phone.via));
        strncat(ok_response, "\r\n", 2);
        strncat(ok_response, "Contact: <sip:", 14);
        strncat(ok_response, my_ip_addr , strlen(my_ip_addr));
        strncat(ok_response, ":5060>\r\n", 8);
```

I'm going to fill all the headers of the response 200 Ok: Via, Contact, to ...

```
..... .... ..... .....
    /* set up destination address */
    memset(&addr,0,sizeof(addr));
    addr.sin_family=AF_INET;
    addr.sin_addr.s_addr=inet_addr(phone.ip_addr);
    addr.sin_port=htons(atoi(phone.port));

    if ((fd=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)) < 0) {
        perror("socket sever");
        exit(1);
    }
    //send 200 OK
    if (sendto(fd,ok_response,sizeof(ok_response),0, (struct sockaddr *) &addr, sizeof(addr)) < 0)
    {
        perror("sendto");
        exit(1);
      }
```

Then, repeat the same but for NOTIFY frame

The listener method the diagram is very easy. First, it waits to listen to something. If I have received something is ok. In the infoPhone method check that the data received is corrects.

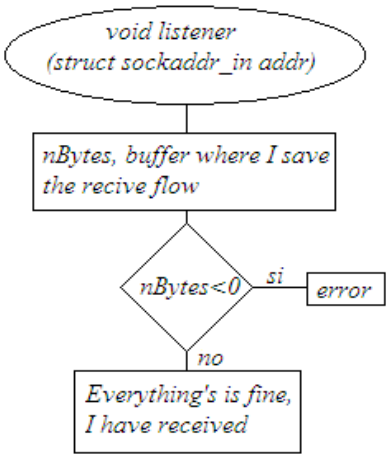

Figure 21: Flow diagram. Listener

The *main*: implements the daemon

```
int main(int argc, char **argv) {

    static int ch, interval, logflag;
    pid_t   pid, sid;

    interval = DEFAULT_INTERVAL;
    logflag  = DEFAULT_LOGFLAG;

    while ((ch = getopt(argc, argv, "lp:")) != -1) {
        switch (ch) {
        case 'l':
            logflag = 1;
            break;
        case 'p':
            interval = atoi(optarg);
            break;
        }
    }
    pid = fork();
/*First the fork() system call will be used to create a copy of our process(child), then let parent
exit. Orphaned child will become a child of init process (this is the initial system process, in
other words the parent of all .....processes). As a result our process will be completely detached
from its parent and start operating in background.*/
    if (pid < 0) {
        exit(EXIT_FAILURE);
    } else if (pid > 0) {
        exit(EXIT_SUCCESS);
    }
    umask(0);
/*Most servers runs as super-user, for security reasons they should protect files that they create.
Setting user mask will pre vent unsecure file priviliges that may occur on file creation. */
    sid = setsid();// obtain a new process group; setsid detaches and puts the daemon into a new
session:
/*A process receives signals from the terminal that it is connected to, and each process inherits
its parent's controlling tty. A server should not receive signals from the process that started it, so
it must detach itself from its controlling tty.
In Unix systems, processes operates within a process group, so that all processes within a group
is treated as a single entity. Process group or session is also inherited. A server should operate
independently from other processes.
This call will place the server in a new process group and session and detach its controlling
terminal. (setpgrp() is an alternative for this)*/
    if (sid < 0) {
        exit(EXIT_FAILURE);
    }
    if ((chdir("/")) < 0) {
        exit(EXIT_FAILURE);
    }
/*A server should run in a known directory. There are many advantages, in fact the opposite has
many disadvantages: suppose that our server is started in a user's home directory, it will not be
able to find some input and output files.
chdir("/servers/"); The root "/" directory may not be appropriate for every server, it should be
choosen carefully depending on the type of the server.*/
    if (logflag == 1)
        syslog (LOG_NOTICE, " started by User %d", getuid ());
//Main code for the exec:
    sleep(interval);
    struct sockaddr_in addr;
    addr = createSocket();
    listener(addr);
```

```
//Is it a SUBSCRIBE?
      if((memcmp(msgbuf, "SUBSCRIBE", 9))==0)
      {
          struct info_phone info;
          info = infoPhone(msgbuf);
          sender(info);
      }
       exit(EXIT_SUCCESS);
}
```