Building Telephony Systems with OpenSER

A step-by-step guide to building a high-performance telephony system

Flavio E. Goncalves

PACKT PUBLISHING

Building Telephony Systems with OpenSER

A step-by-step guide to building a high-performance telephony system

Flavio E. Goncalves

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

Building Telephony Systems with OpenSER

First published: April 2008

Production Reference: 1140408

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847193-73-5

www.packtpub.com

Cover Image by Raghuram Ashok (raghuram@iiitb.ac.in)

Credits

Author

Flavio E. Goncalves

Reviewers

Bogdan-Andrei Iancu

Daniel-Constantin Mierla

Development Editor

Swapna V. Verlekar

Technical Editor

Bhupali Khule

Editorial Team Leader

Mithil Kulkarni

Project Manager

Abhijeet Deobhakta

Indexer

Hemangini Bari

Proofreader

Chris Smith

Production Coordinator

Shantanu Zagade

Cover Work

Shantanu Zagade

About the Author

Flavio E. Goncalves was born in 1966 in Minas Gerais, Brazil. Having always had a strong interest in computers, he got his first personal computer in 1983 and since then it has been almost an addiction. He received his degree in Engineering in 1989 with focus in computer aided design and computer aided manufacturing.

He is also CEO of V.Office Networks in Brazil, a consulting company dedicated to the areas of Networks, Security, Telecom, and Operating Systems and a training center since its foundation in 1996. Since 1993, he has participated in a series of certification programs having being certificated as Novell MCNE/MCNI, Microsoft MCSE/MCT, Cisco CCSP/CCNP/CCDP, Asterisk dCAP, and some others.

He started writing about open-source software because he thinks the way certification programs were organized in the past was very good to help learners. Some books today are written by strictly technical people, who, sometimes, do not have a clear idea on how people learn. He tried to use his 15-year experience as instructor to help people learn open-source telephony software. His experience with networks, protocol analyzers, and IP telephony, combined with his teaching skills, gave him an edge to write this book. This is the second book he has written; the first one was *The Configuration Guide for Asterisk PBX*.

As the CEO of V.Office, Flavio E. Goncalves balances his time between family, work, and fun. He is the father of two children and lives in Florianopolis, Brazil, in his opinion one of the most beautiful places in the world. He dedicates his free time to water sports such as surfing and sailing.

You can contact him at `flavio@asteriskguide.com`, or visit his website `www.asteriskguide.com`.

Writing this book has been a process that involved many people. I would like to thank the staff at Packt Publishing who worked in all the process of reviewing and editing the book. I would like to thank Guilherme Goes, who wrote a good part of Chapter 6 and developed SerMyAdmin for this book. I would also like to thank several students, who took courses in the first versions of this book (in the Portuguese language) for their feedback. Finally, I would like to thank my family, for all the support they gave me during all these years.

About the Reviewers

Bogdan-Andrei Iancu is a part of the new generation of IT people with dual nature—both technical and business. He is a co-founder of SER and OpenSER projects and also founder and CEO of **Voice System SRL**, a "know-how" VoIP/ OpenSER company.

Born in 1978 in Romania, he received in 2001 the Master Degree in **Computer Science** at **University "Politehnica" Bucharest**. For the next 4 years, his research work at **Fraunhofer Fokus Research Institute** for Open Communication, Berlin is sustained by hands-on experience in VoIP/SIP area as co-founder (in 2002) and main developer of the Open Source project "SIP Express Router".

In 2004, Bogdan-Andrei Iancu starts his own enterprise—Voice System—dedicated to designing, implementing, and deploying VoIP platforms. Focusing on advanced service and dynamic routing together with scalability and security, in 2005 he (along with other members of Voice System's team) co-founds the OpenSER public project as the next step in VoIP enhancement.

For the last 4 years, Bogdan Iancu concentrated the Voice System energy in a dual head direction: continue effort and contribution to the Open Source "OpenSER" project as code, advertising, management, and sponsorship; developing industry proofed VoIP platforms and solutions from ITSPs/ISPs to large carriers and telcos.

Voice System team grouped over the years more 7 core and main developers for the "OpenSER" project, accumulating a large and comprehensive knowledge on it. The works goes hand in hand with research and standardization especially in new SIP related domains like presence, where Anca-Maria Vamanu provided a full presence implementation for the project.

Voice System, as major OpenSER sustainer, is looking in how to share valuable knowledge about OpenSER with the rest of community via several ways: training courses, documentation, and starting from now, with helping the emergency of OpenSER related books.

> But all this wouldn't have been possible without the sustained effort and help of the entire OpenSER community—developers and users and I would like to thanks to all of them for putting trust in Open Source and OpenSER

Daniel-Constantin Mierla is co-founder of OpenSER SIP Server project and CEO of ASIPTO, a company focused on VoIP and OpenSER-based services. His experience with SIP and VoIP started in the beginning of 2002, since then authoring many online tutorials about OpenSER, including "OpenSER Devel Guide", "OPENSER Core Cookbook", "OpenSER Pseudo-variables and transformations". He participates periodically to VoIP events, speaking about OpenSER and VoIP. Since 2005, when OpenSER started, he is member of the management board of the project.

Table of Contents

Preface

We are starting a new era in the collaboration area. Voice and Video over IP are starting to dominate the world of telecommunications in a disruptive movement capable of changing the whole industry. The SIP (Session Initiation protocol) technology is at the center of this revolution. I believe, at present, SIP is the most used protocol for Voice and Video over IP.

In the future, when people learn how to use the technology, SIP will be for voice communications what email is today for text communications. We are starting with islands of SIP communications inside VoIP providers, enterprises, and even governments. In the near future, the barriers between the islands will be broken and you will be able to communicate with anyone in anyplace without paying high fees. The only fees you will pay in the future will be the access to the data network, because with the pervasiveness of VoIP and Video over IP, everything will be simply data. I remember the first days of Internets email in the early 90s. It took some years until everyone had an email address. The same thing could be happening in the SIP world now. Unfortunately, the SIP providers still behave as islands not, usually, allowing free inter-domain routing.

With the introduction of 3G, 4G, and WiMAX, fast data communications are becoming widespread in the mobile industry. Newer phones from mainstream manufacturers are starting to support WiFI, WiMAX, and obviously 3G. SIP clients can run in these platforms changing the whole mobile communication industry in the near future. Sure, the telephone companies will try, legitimately, to protect their revenue sources, but they cannot hide for ever the SIP communication infrastructure already in place. Slowly, users will start to use SIP clients in their mobile phones hugely cutting the communication costs. The movement, even without a huge sponsor, will spread by word of mouth until it becomes pervasive.

The infrastructure required for SIP communication has as its main component a "SIP Proxy" server. OpenSER is one of the best SIP Proxies in the market. It is robust, scalable, and licensed according to GNU GPL. OpenSER is now in the stage of early adopters. It is still hard to learn and to use. The idea of this book is to teach you how to implement the architecture of the SIP protocol using OpenSER. I hope this book helps you, if you are starting to learn SIP, or implement a SIP infrastructure in your company, school, or government. I wish you success in your implementations and I sincerely hope that this material helps you.

What This Book Covers

Chapter 1 provides an overview of the SIP protocol, its architecture, and its main components. SIP flows are explained and will be essential for the future comprehension on this book. Some important concepts such as codecs, session description protocol, and real-time protocol are presented at the end of the chapter.

Chapter 2 will give you an overview of the OpenSER software. We also cover how SIP requests are processed and the basic concepts of transactions and dialogs after explaining what SIP is.

Chapter 3 is where you will learn how to install Linux prepared for OpenSER and OpenSER itself. After the installation, you will learn how to start and stop the daemon and how to initialize OpenSER at boot time.

Chapter 4 introduces you to the basic scripts and analyzes the default configuration. At the end it shows you some important concepts about routing transactions and dialogs. Be sure to understand the routing basics before going ahead.

Chapter 5 shows how to connect OpenSER to a MySQL database to authenticate all the initial requests. Later in this chapter you will see how to add some security mechanisms to improve your system.

Chapter 6 introduces some important concepts about the user portal. You will learn how to install and do the basic operations with SerMyAdmin. SerMyAdmin is a graphical user interface for OpenSER that aims to make your life easier in the administration of the server.

Chapter 7 teaches you how to connect to the PSTN (Public Switched Telephone Network) using a gateway. Details on how to connect to an Asterisk Server or a Cisco gateway are provided.

Chapter 8 introduces you to the concepts of call forwarding to a voicemail server. You can use Asterisk as a voicemail server connected to OpenSER. Concepts such as failure_route and AVPs are presented in this chapter.

Chapter 9 covers SIP NAT traversal. It introduces the problems and techniques to traverse NAT devices for SIP communications.

Chapter 10 is about billing. It teaches you how to send call detail records to a RADIUS Server and how to rate the calls using a GPL tool known as CDRTool.

Chapter 11 introduces some tools to help you in stress test your platform, detect voice quality problems, and trace SIP calls.

In *After Words* you can see last minute information covering newer versions of OpenSER, not available when the book was written.

What You Need for This Book

To use this book, I recommend that you have a formatted PC with at least 1GHz of CPU, 20G bytes of disk and 512 Mbytes of RAM. You can also use the free and downloadable VMWare Server *(*`http://www.vmware.com`*)*, to install Linux and OpenSER safely in a VM (virtual machine) running inside your own machine. You will have to download the Debian distribution of Linux (`www.debian.org`). I used the version 4.0R3 (etch) to test the labs. All the software used in this book is freely available on the Internet, so in the proper chapters you will find the instructions on how to download and execute. You will need at least two SIP devices to test most of the labs. I use two PCs with a free SIP softphone and OpenSER in a virtual machine. For the PSTN and Voicemail chapters, you will need to use an external gateway, usually an Asterisk Server. In my testing environment I used a second virtual machine using VMWare server. If you are going to use VMWare, be sure to disable desktop firewall, a good source of headaches for labs.

Who This Book Is For

This book is intended for Linux and networking professionals, who want to understand SIP and OpenSER from a practical perspective, or are interested in IP telephony and call routing. It is suitable for VoIP provider personnel, because it covers most of the existing components. Some experience with Linux and Networks is required to be successful in the labs. Even inexperienced Linux users can complete the labs, but knowledge about computer networks is essential. For inexperienced Linux users I suggest using WinSCP and Putty to edit the configuration files and send commands; both are free downloads.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are three styles for code. Code words in text are shown as follows: "Notice that we have added only the keywords `contrib` and `non-free` after our repository definitions".

A block of code will be set as follows:

```
# /etc/apt/souces.list
deb http://ftp.br.debian.org/debian/ etch main contrib non-free
deb-src http://ftp.br.debian.org/debian/ etch main contrib non-free
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/serMyAdmin">
  <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver"
maxActive="20" maxIdle="10" maxWait="-1" name="jdbc/openser_MySQL"
type="javax.sql.DataSource" url="jdbc:mysql://localhost:3306/openser"
username="sermyadmin" password="secret"/>
</Context>
```

Any command-line input and output is written as follows:

```
openser:/usr/src# cp mysql-connector-java-5.1.5/mysql-connector-java-
5.1.5-bin.jar /usr/local/tomcat6/lib
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "To register a new user, in the login screen simply click on the **Register** Button".

Important notes appear in a box like this.

Tips and tricks appear like this.

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to `feedback@packtpub.com`, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or email `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the Example Code for the Book

Visit `http://www.packtpub.com/files/code/3735_Code.zip` to directly downlad the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **Submit Errata** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

1
Introduction to SIP

The Session Initiation Protocol (SIP) was standardized by the Internet Engineering Task Force (IETF) and is described in several documents known as RFCs (Request for Comments). RFC3261 is one of the most recent and is called SIP version 2. SIP is an application-layer protocol used to establish, modify, and terminate sessions or multimedia calls. These sessions can be conferences, e-learning, telephony over the Internet, and similar applications. It is based on a text protocol similar to Hypertext Transfer Protocol (HTTP) and it is designed to start, keep, and close interactive communication sessions between users. These days SIP is one of the most used protocols for VoIP and is present on almost every IP phone in the market.

By the end of this chapter you will be able to:

- Describe what SIP is
- Describe what SIP is for
- Describe SIP architecture
- Explain the meaning of its main components
- Understand and compare the main SIP messages
- Describe the header fields processing for INVITE and REGISTER requests

The SIP protocol supports five features for establishing and closing multimedia sessions.

- **User location:** Determines the endpoint address used for communication.
- **User parameters negotiation:** Determines the media and parameters to be used.
- **User availability:** Determines if the user is available or not to establish a session.
- **Call establishment:** Establishes the parameters for caller and callee, and informs on call progress (ringing, ringback, congestion) to both parties.
- **Call management:** Session transfer and closing.

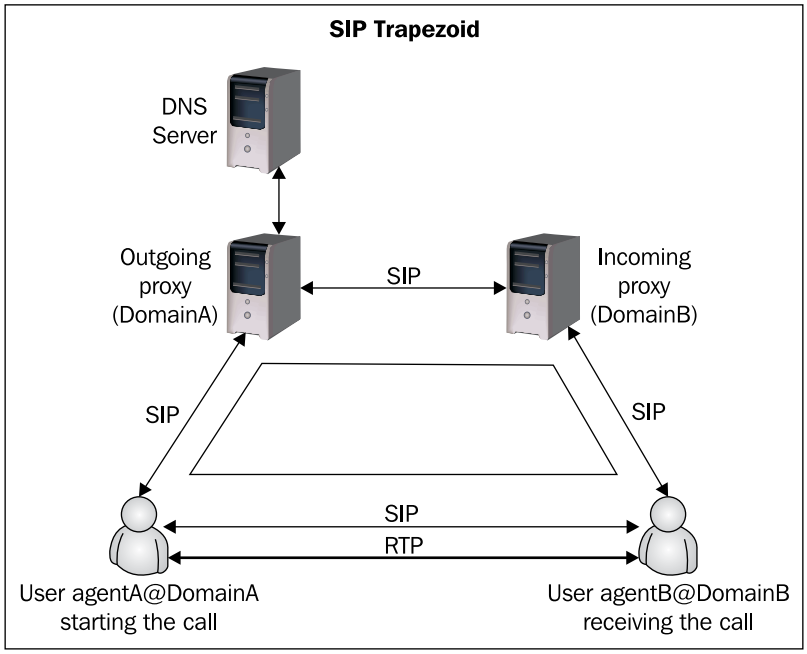
The SIP protocol was designed as part of a multimedia architecture containing other protocols such as RVSP, RTP, RTSP, and SDP. However it does not depend on them to work.

SIP Basics

SIP is very similar to HTTP in the way it works. The SIP address is just like an e-mail address. An interesting feature used in SIP proxies is **alias**, so you can have multiple SIP addresses such as:

- johndoe@sipA.com
- +554845678901@sipA.com
- 45678901@sipA.com

In the SIP architecture, we have user agents and servers. SIP uses a peer-to-peer distributed model with a signaling server. The server handles just the signaling, while the user agent clients and the user agent servers handles signaling and media. This is depicted in the figure below:



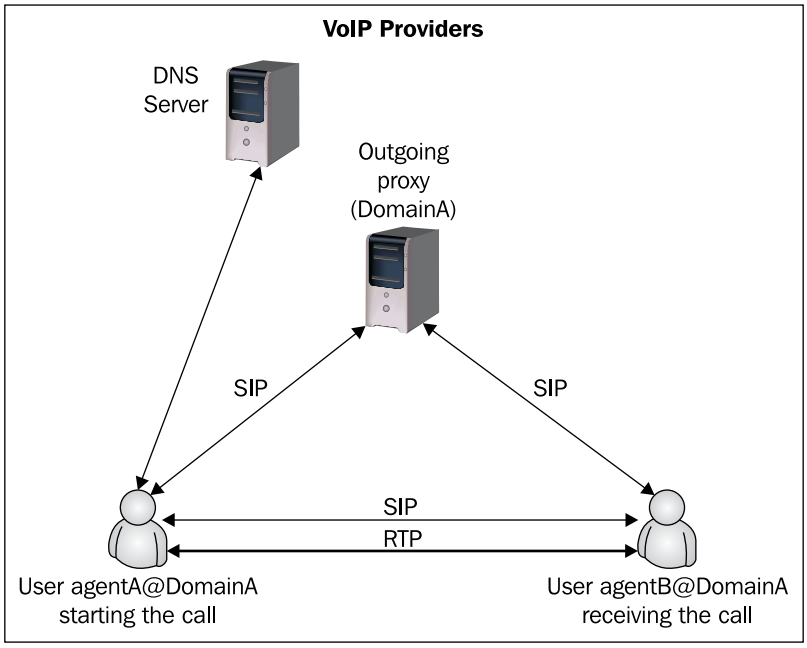
In the SIP model, a user agent, usually a SIP phone, will start communicating with its SIP proxy, seen here as the outgoing proxy, to send the call using a message known as INVITE.

The outgoing proxy will see that the call is directed to an outside domain. It will seek the DNS server for the address of the target domain and resolve the IP address. Then, the outgoing proxy will forward the call to the SIP proxy responsible for DomainB.

The incoming proxy will query its location table for the IP address of agentB. If this address was inserted in the location table by a previous registration process, so the incoming proxy can locate the address. Now with this address, it can forward the call to agentB.

After receiving the SIP message, agentB will have all the information required to establish an RTP session (usually audio) with agentA. Using a message such as BYE will terminate the session.

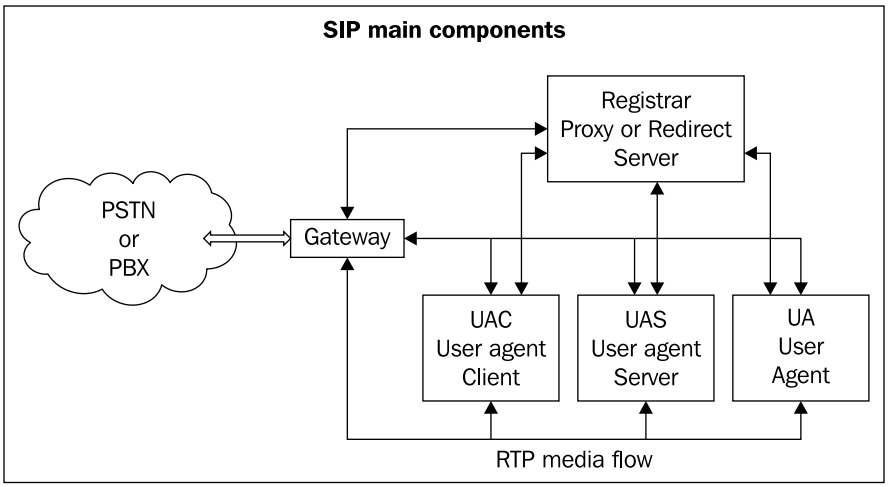
SIP Proxy in the Context of a VOIP Provider



Usually VoIP providers don't implement a pure SIP trapezoid, they don't allow you to send calls to outside domains, because this affects the revenue stream. They implement a topology closer to a SIP triangle.

SIP Operation Theory

Below, you can see the main components of the SIP architecture. The entire SIP signaling flows through the SIP proxy server. On the other hand, the media signaling, transported by the RTP protocol, flows directly from one endpoint to another. Some of the components will be briefly explained in the list below.



- **UAC (user agent client)**—Client or terminal that starts the SIP signaling.
- **UAS (user agent server)**—Server that responds to the SIP signaling coming from a UAC.
- **UA (user agent)**—SIP terminal (IP phone, ATA, softphone).
- **Proxy Server**—It receives requests from a UA and transfers them to another SIP proxy if this specific terminal requested is not under its domain.
- **Redirect Server**—This receives requests and sends back to the caller including data about the destination, instead of sending directly to the callee.
- **Location Server**—This provides the callee's contact addresses to Proxy and Redirect Servers.

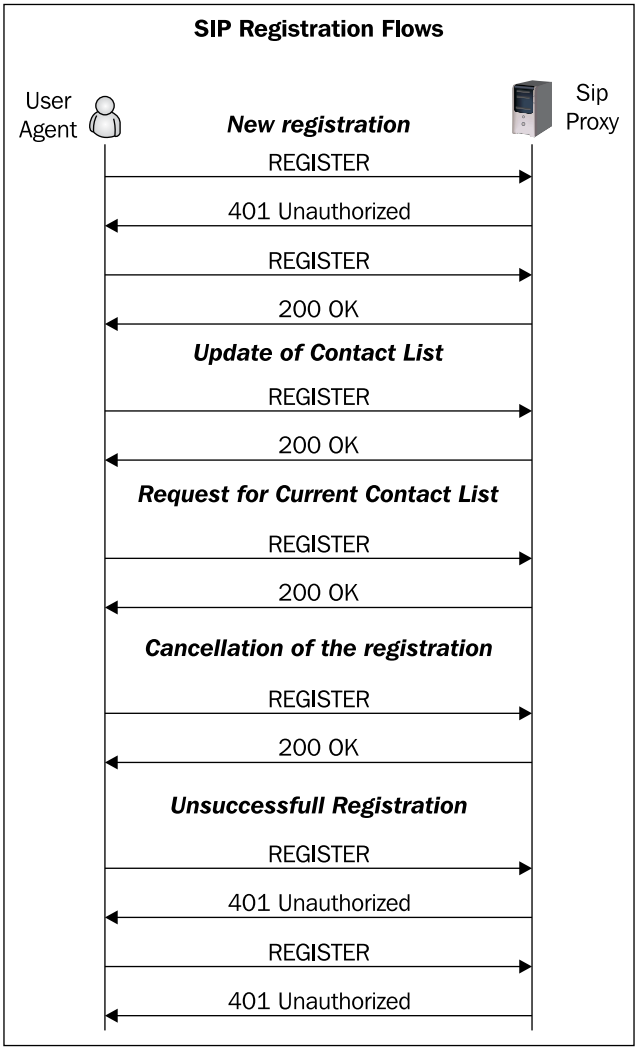
The Proxy, Redirect, and Location servers are usually available physically in the same computer and software.

SIP Registration Process

The SIP protocol employs a component called a **registrar**. It is a server that accepts REGISTER requests and saves the information received in these packets on the Location server for their managed domains. The SIP protocol has a discovery capacity; in other words, if a user starts a session with another user, the SIP protocol has to discover an existing host where the user can be reached. The discovery process is done by a Location server that receives the request and finds where to send it. This is based in a Location database maintained by the Location server per domain. The Registrar server may accept other types of information, not only the client's IP addresses. It can receive other information such as CPL (Call Processing Language) scripts on the server.

Before a telephone can receive a call, it needs to be registered within the location database. In this database we will have all phones associated with their respective IP addresses. In our example, you will see the SIP user `8590@voffice.com.br` registered at the IP address 200.180.1.1.

RFC3665 defines best practices to implement a minimum set of functionality for a SIP IP communications network. Below are the flows defined according to RFC3665 for the register transactions:



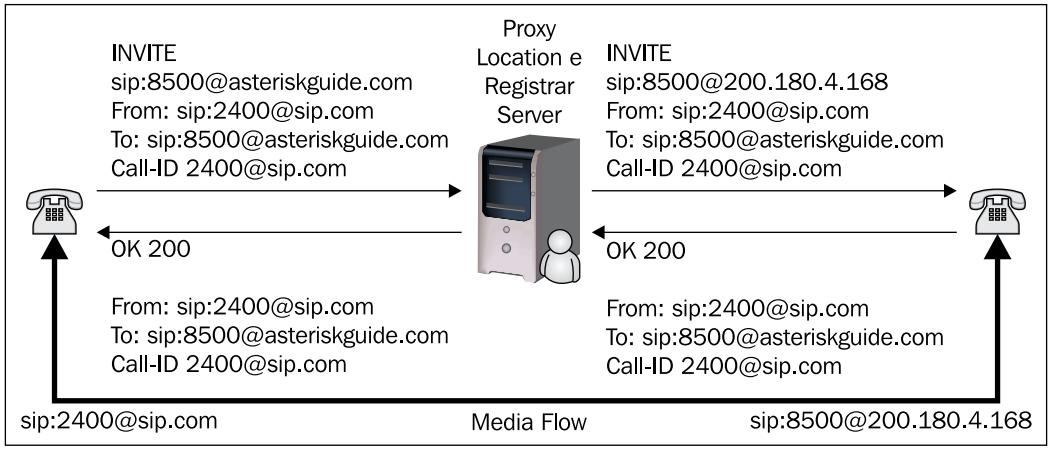
According to RFC3665, there are five basic flows associated with the process of registering a user agent, which are as follows:

1. A successful new registration—after sending the Register request, the user agent will be challenged against its credentials. We will see this in detail in the chapter dedicated to authentication.

2. An update of the contact list—Since it is not a new registration, the message already contains the digest and a "401" message won't be sent. To change the contact list, the user agent just needs to send a new register message with the new contact in the CONTACT header field.
3. Request for current contact list—In this case, the user agent will send the CONTACT header field empty, indicating the user wishes to query the server for the current contact list. In the **200 OK** message, the SIP server will send the current contact list in the CONTACT header field.
4. Cancellation of a registration—The user agent now sends the message with an EXPIRES header field of **0** and a CONTACT HEADER field configured as '*' to apply to all existing contacts.
5. Unsuccessful Registration—The UAC sends a Register Request and receives a "401 Unauthorized" message, in exactly the same way as the successful registration. In the sequence, it produces a hash and tries to authenticate. The server, detecting an invalid password, again sends a "401 Unauthorized" message. The process will be repeated for the number of retries configured in the UAC.

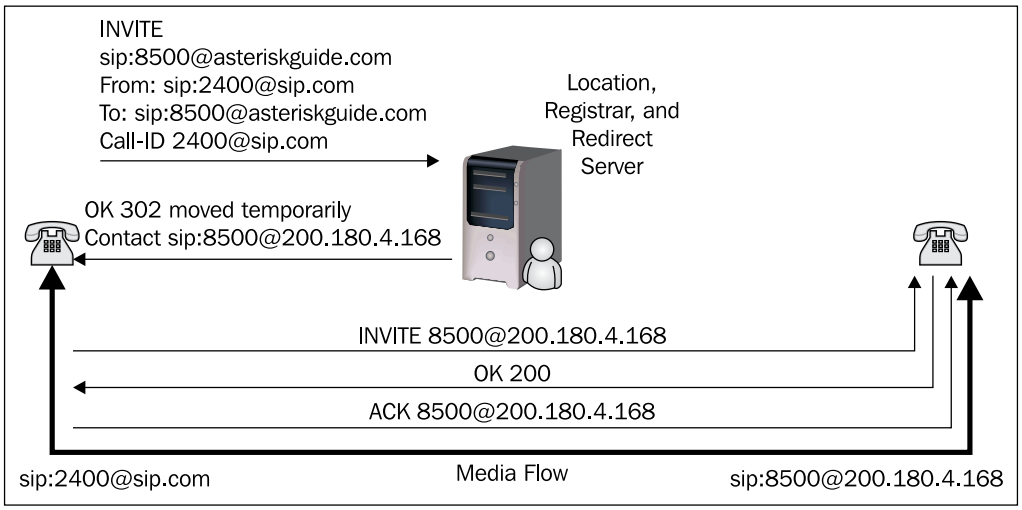
Server Operating as a SIP Proxy

In the SIP proxy mode, the entire SIP signaling goes through the SIP proxy. This behavior will help in processes such as billing and it is, by far, the most common choice. The drawback is the overhead caused by the server in the middle of all SIP communications during the session establishment. Remember, RTP packets will always go directly from one endpoint to another, even if the server is working as a SIP proxy.



Server Operating as a SIP Redirect

The SIP proxy can operate in the SIP redirect mode. In this mode the SIP server is very scalable, because it doesn't keep the state of transactions. Just after the initial INVITE, it replies to the UAC with a "302 Moved Temporarily" and gets off the SIP dialog. In this mode a SIP proxy, even with very few resources, can forward millions of calls per hour. It is normally used when you need high scalability, but don't need to bill the calls.



Basic Messages

The basic messages sent in a SIP environment are:

SIP basic messages

Method	Description	RFC
ACK	Acknowledge an INVITE	RFC3261
BYE	Terminate an existing session	RFC3261
CANCEL	Cancel a pending request	RFC3261
INFO	Mid-call signaling information	RFC2976
INVITE	Session establishment	RFC3261
MESSAGE	Instant message transport	RFC3428
NOTIFY	Send information after subscribe	RFC3265
OPTIONS	Query the capabilities of the UA	RFC3261
PRACK	Acknowledge a provisional response	RFC3262
PUBLISH	Upload status information to the server	RFC3903
REGISTER	Register the user an update the location table	RFC3261
REFER	Ask another UA to act upon URI	RFC3515
SUBSCRIBE	Establish a session to receive future updates	RFC3265
UPDATE	Update session state information	RFC3311

Most of the time, you will use REGISTER, INVITE, BYE, and CANCEL. Some messages are used for other features. As an example, INFO is used for DTMF relay and mid-call signaling information. PUBLISH, NOTIFY, and SUBSCRIBE give support to presence systems. REFER is used for call transfer and MESSAGE for chat applications. Newer messages can appear depending on the protocol standardization process.

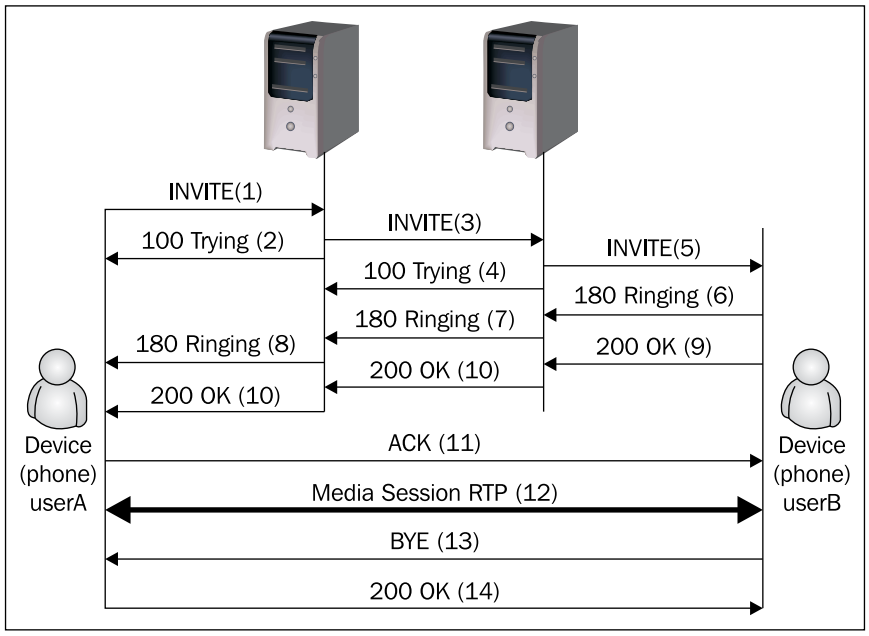
Responses to these messages are in text format as in the HTTP protocol. Some of the most important are shown below:

Response Codes

Description	Code	Examples	
Informational or provisional response	1XX	100 Trying 180 Ringing 181 Call Is Being Forwarded	182 Queued
Success	2XX	200 OK 202 Accepted	
Redirect	3XX	300 Multiple Choices 301 Moved Permanently 302 Moved Temporarily	303 See Other 305 Use Proxy 380 Alternative Service
Client Errors	4XX	400 Bad Request 401 Unauthorized 402 Payment Required 403 Forbidden 404 Not Found 405 Method Not Allowed 406 Not Acceptable 407 Proxy Authentication Required 408 Request Timeout 409 Conflict 410 Gone	411 Length Required 413 Request Entity Too Large 414 Request-URI Too Large 415 Unsupported Media Type 420 Bad Extension 480 Temporarily not available 481 Call Leg/Transaction Does Not Exist 482 Loop Detected 483 Too Many Hops 484 Address Incomplete 485 Ambiguous 486 Busy Here
Server Errors	5XX	500 Internal Server Error 501 Not Implemented 502 Bad Gateway	503 Service Unavailable 504 Gateway Time-out 505 SIP Version not supported
Global Errors	6XX	600 Busy Everywhere 603 Decline	604 Does not exist anywhere 606 Not Acceptable

SIP Dialog Flow

This section introduces some basic SIP operations using a simple example. Let's examine this message sequence between two user agents shown below. You can see several other flows associated with the session establishment in RFC3665.



The messages are labeled in sequence. In this example userA uses an IP phone to call another IP phone over the network. To complete the call, two SIP proxies are used.

The userA calls userB using its SIP identity, called SIP URI. The URI is similar to an email address, such as `sip:userA@sip.com`. A secure SIP URI can be used too, such as `sips:userA@sip.com`. A call made using SIPS will use a secure transport (TLS-Transport Layer Security) between the caller and the callee.

The transaction starts with userA sending an INVITE request addressed to userB. The INVITE request contains a certain number of header fields. Header fields are named attributes that provide additional information about the message; they include a unique identifier, the destination, and information about the session.

The first line of the message contains the method name. The following lines contain a list of header fields. This example contains the minimum set required. We will briefly describe these header fields below:

- **VIA**: This contains the address at which **userA** will be waiting to receive responses to this request. It also contains a parameter called branch that identifies this transaction. The VIA header defines the last SIP hop as IP, transport, and transaction-specific parameters. VIA is used exclusively for routing back the replies. Each proxy adds an additional VIA header. It is a lot easier for replies to find the route back using the VIA header, than to go again to the location server or DNS.
- **TO**: This contains the name (display name) and the SIP URI (that is, `sip:userB@sip.com`) to the destination originally selected. The TO header field is not used to route the packets.
- **FROM**: This contains the name and SIP URI (that is, `sip:userA@sip.com`) that indicate the caller ID. This header field has a tag parameter containing a random string that was added to the URI by the IP phone. It is used for purposes of identification. The tag parameter is used in the TO and FROM fields. It serves as a general mechanism to identify the dialog, which is the combination of the Call-ID along with the two tags, one from each participant in the dialog. Tags can be useful in parallel forking.

- **CALL-ID**: This contains a globally unique identifier for this call generated by the combination of a random string and the host name or IP address from the IP phone. A combination of the tags TO, FROM, and CALL-ID fully defines an end-to-end SIP relation known as a SIP dialog.
- **CSEQ**: The CSEQ or command sequence contains an integer and a method name. The CSEQ number is incremented for each new request inside a SIP dialog and is a traditional sequence number.
- **CONTACT**: This contains a SIP URI, which represents a direct route to contact **userA**, usually composed of a user name and a FQDN (fully qualified domain name). Sometimes the domains are not registered, thus, IP address are permitted too. While the VIA header field tells the other elements where to send a response, the CONTACT tells the other elements where to send future requests.
- **MAX-FORWARDS**: This is used to limit the number of allowed hops a request can make in the path to its final destination. It consists of an integer decremented by one on each hop.
- **CONTENT-TYPE**: This contains a body message description.
- **CONTENT-LENGTH**: This contains a byte count of the body message.

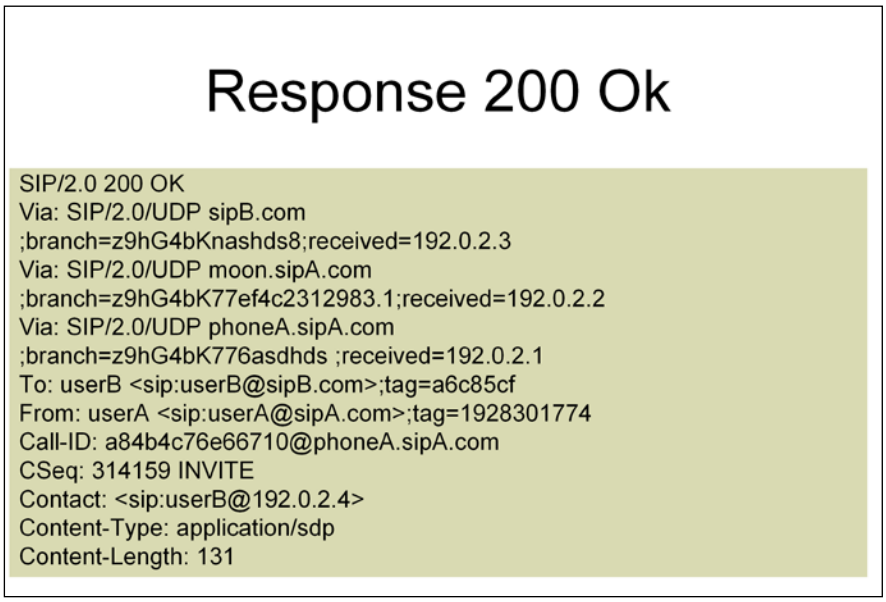
Session details, like media type and codec are not described using SIP. Instead it uses a session description protocol called SDP (RFC2327). This SDP message is carried by the SIP message, similar to an email attachment.

The sequence is as follows:

The phone does not know the location of **userB** or the server responsible for domainB. Thus, it sends the INVITE request to the server responsible for the domain sipA. This address is configured in the phone of userA or can be discovered by DHCP. The server sipA.com is also known as the SIP proxy for the domain sipA.com.

1. In this example, the proxy receives the INVITE request and sends a message "100 trying" back to userA, signaling that the proxy received the INVITE and is working to forward the request. The SIP responses use a three digit code followed by a descriptive phrase. This response contains the same TO, FROM, CALL-ID, and CSEQ header fields and a parameter "branch" in the header field VIA as the INVITE request. This allows **userA**'s phone to correlate the INVITE request sent.
2. ProxyA locates proxyB consulting a DNS server (SRV records) to find what server is responsible for the SIP domain sipB and forwards the INVITE request. Before sending the request to proxyA, it adds a VIA header field that contains its own address. This allows userA's phone to correlate the response to the INVITE request sent. .
3. ProxyB receives the INVITE request and responds with a "100 Trying" message back to proxyA indicating that it is processing the request.
4. ProxyB consults its own location database for userB's address and then it adds another VIA header field with its own address to the INVITE request and sends to **userB**'s IP address.
5. UserB's phone receives the INVITE request and start ringing. The phone indicates back this condition, sending a message "180 Ringing".
6. This message is routed back through both proxies in the reverse direction. Each proxy uses the VIA header fields to determine where to send the response and removes its own VIA header from the top. As a result, the message "180 Ringing" can return back to the user without any lookups to DNS or Location Service Responses and without the need for stateful processing. Thus, each proxy sees all messages resulting from the INVITE request.
7. When userA's phone receives the "180 Ringing" Responses, it starts to ring back, to signal to the user that the call is ringing on the other side. Some phones show this in the display.

8. In this example, userB decides to answer the call. When userB responds, the phone sends a response "200 Ok" to indicate that the call was taken. The "200 Ok" message contains in its body a session description specifying codecs, ports, and everything pertaining to the session. It uses the SDP protocol for this duty. As a result, there is an exchange in two phases of messages from A to B (INVITE) and B to A (200 OK) negotiating the resources and capabilities used on the call in a simple "offer/response" model. If userB does not want to receive the call or is busy, the "200 OK" won't be sent and a message signaling the condition (that is, "486 Busy Here") will be sent instead.



The first line contains the response code and a description (OK). The following lines contain the header fields. The fields VIA, TO, FROM, CALL-ID, and CSEQ are copied from the INVITE request. There are three VIA fields, one added by **userA**, another by proxyA and finally that added by proxy B. The SIP phone of userB added a parameter TAG on both end points inside the dialog, which will be included on all future requests and responses for this call.

The CONTACT header field contains the URI with which **userB** can be contacted directly on their own IP phone.

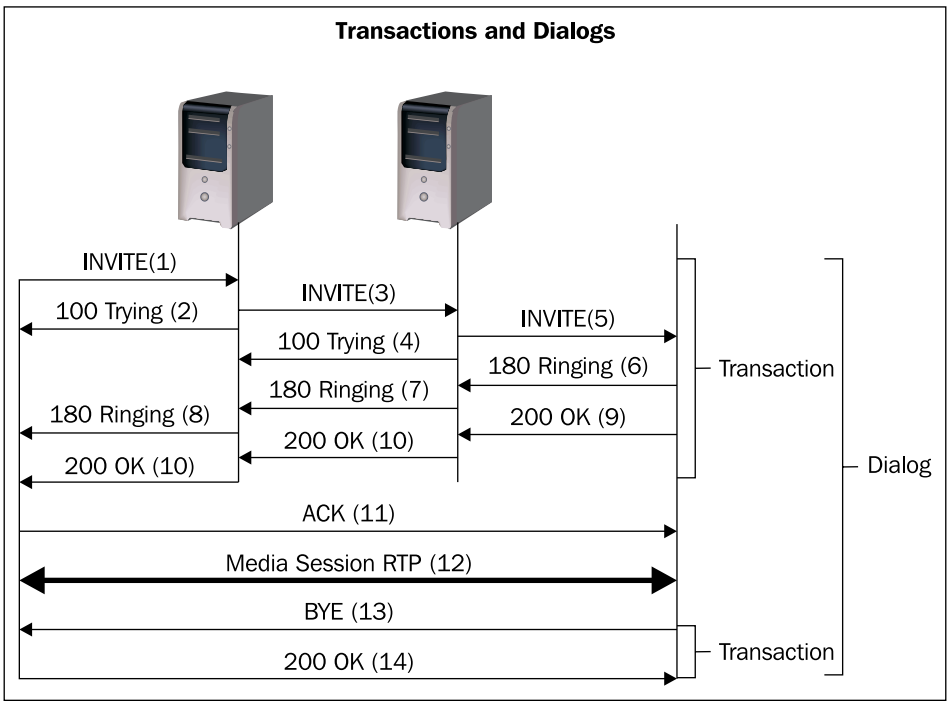
The CONTENT-TYPE and CONTENT-LENGTH header-fields give some information about the the SDP header ahead. The SDP header contains media-related parameters used to establish the RTP session.

1. In this case, the message "200 Ok" is sent back through both proxies and is received by userA and then the phone stops ringing back indicating that the call was accepted.
2. Finally userA sends an ACK message to userB's phone confirming the reception of the "200 OK" message. In this example the ACK is sent directly from phoneA to phoneB avoiding both proxies. ACK is the only SIP method that has no reply. The endpoints learned each other's addresses from the CONTACT header fields during the INVITE process. This ends the cycle INVITE/200 OK/ACK also known as SIP three way handshake.
3. At this moment the session between both users starts and they send media packets to each other using a mutually agreed format established by the SDP protocol. Usually these packets are end-to-end. During the session, the parties can change the session characteristics issuing a new INVITE request. This is called a re-invite. If the re-invite is not acceptable, a message "488 Not Acceptable Here" will be sent, but the session will not fail.
4. At the session end, userB disconnects the phone and generates a BYE message. This message is routed directly to userA's softphone bypassing both proxies.
5. UserA confirms the reception of the BYE message with a "200 OK" message ending the session. No ACK is sent. An ACK is sent only for INVITE requests.

In some cases it can be important for proxies to stay in the middle of the signaling to see all messages between endpoints during the whole session. If the proxy wants to stay in the path after the initial INVITE request it has to add the RECORD-ROUTE header field to the request. This information will be received by userB's phone and it will send back the message through the proxies with the RECORD-ROUTE header field included too. Record routing is used in most scenarios.

The REGISTER request is the way that proxyB uses to learn the location of userB. When the phone initializes or at regular time intervals, softphone B sends a REGISTER request to a server on domain sipB known as "SIP REGISTRAR". The REGISTER messages associate a URI (`userB@sipB.com`) to an IP address. This binding is stored in a database in the Location server. Usually the Registrar, Location, and Proxy server are in the same computer and use the same software. OpenSER is capable of playing the three roles. A URI can only be registered by a single device at a certain time.

SIP Transactions and Dialogs



It is important to understand now the difference between a transaction and a dialog. A transaction occurs between a user agent client and a user agent server and comprises all messages from the first request to the final response. The responses can be provisional starting with 1 followed by two digits (e.g. 180 Ringing) or final starting with 2 followed by two digits (e.g. 200 OK). The scope of a transaction is defined by the stack of VIA headers of the SIP messages. So, the user agents, after the initial invite, don't need to rely on DNS or location tables to route the messages.

A dialog usually starts with an INVITE transaction and ends with a BYE transaction. A dialog is identified by the CALL-ID header field. A combination of the TO tag, the FROM tag, and the Call-ID completely defines the dialog.

According to RFC 3665 there are 11 basic session establishment flows. The list is not meant to be complete, but covers the best practices. The first two were already covered in this chapter, "Successful Session Establishment" and 'Session Establishment Through Two Proxies". Some of them will be seen in the chapter dedicated to call forwarding such as "Unsuccessfull with no Answer" and "Unsuccessful Busy".

The RTP Protocol

The Real Time Protocol (RTP) is responsible for the real-time transport of data such as audio and video. It was standardized in RFC3550. It uses UDP as the transport protocol. To be transported, the audio or video has to be packetized by a codec. Basically, the protocol allows the specification of timing and content requirements of the media transmission for the incoming and outgoing packets using:

- Sequence number
- Timestamps
- Packet forward without retransmission
- Source identification
- Content identification
- Synchronism

The RTP has a companion protocol called RTCP (Real Time Control Protocol) used to monitor the RTP packets. It can measure the delay and jitter.

Codecs

The content described in the RTP protocol is usually encoded by a codec. Each codec has a specific use. Some have compression while others don't. The G.711 codec, which does not use compression, is very common. With 64Kbps of bandwidth for a single channel it needs a high speed network, commonly found in Local Area Networks (LANs). However, in Wide Area Networks (WAN) 64Kbps can be too expensive to buy for a single voice channel. Codecs such as G.729 and GSM can compress the voice packets to as low as 8Kbps saving a lot of bandwidth. Some codecs such as the iLBC from Global IP sound can conceal packet loss. The iLBC can sustain a good voice quality even with 7% packet loss. So you have to choose the codecs you will support in your VoIP provider wisely.

DTMF-Relay

In some cases the RTP protocol is used to carry signaling information such as DTMF. RFC2833 describes a method to transmit DTMF as named events in the RTP protocol. It is very important that you use the same method between user agent servers and user agent clients.

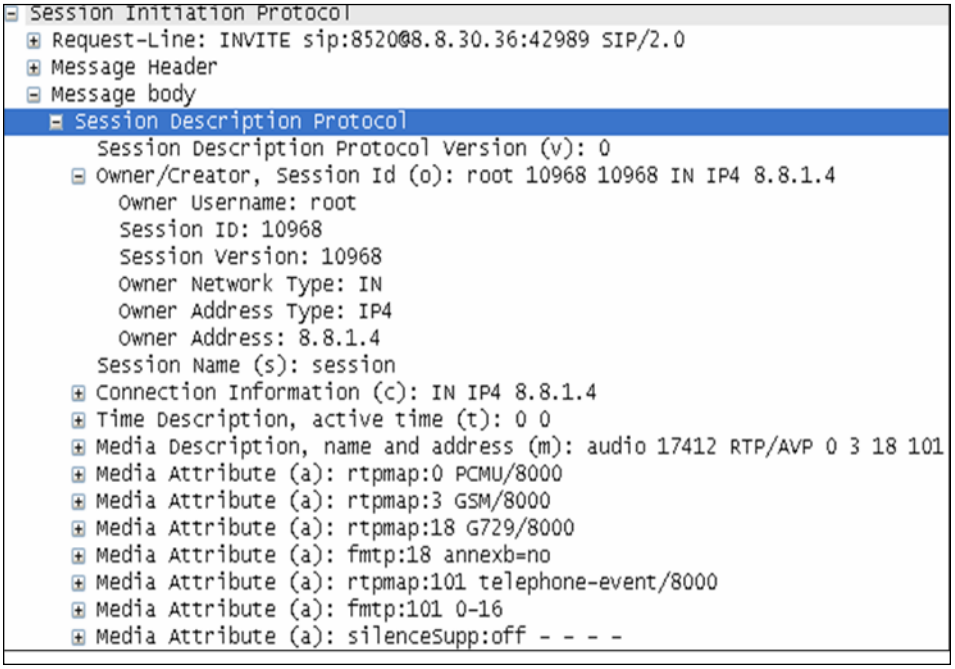
Real Time Control Protocol (RTCP)

RTCP can provide feedback on the quality of reception. It provides out-of-band control information for an RTP media flow. Statistics such as jitter, round trip time (RTT), latency, and packet loss can be gathered using RTCP. RTCP is usually used for voice quality reporting.

Session Description Protocol (SDP)

The SDP protocol is described in RFC4566. It is used to negotiate session parameters between the user agents. Media details, transport addresses, and other media-related information are exchanged between the user agents using the SDP protocol. Normally the INVITE message contains the SDP offer message, while the "200 OK" contains the answer message. Below these messages are shown. You can observe that the GSM codec is offered, but the other phone does not support it. Then it answers with the supported codecs, in this case G.711 ulaw (PCMU) and G.729. The session rtpmap:101 is the DTMF-relay described in the RFC2833.

INVITE (SDP Offer).

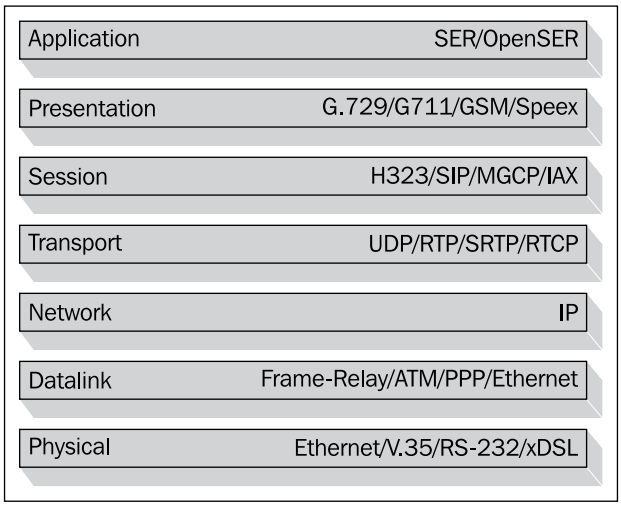


200 OK (SDP Answer).

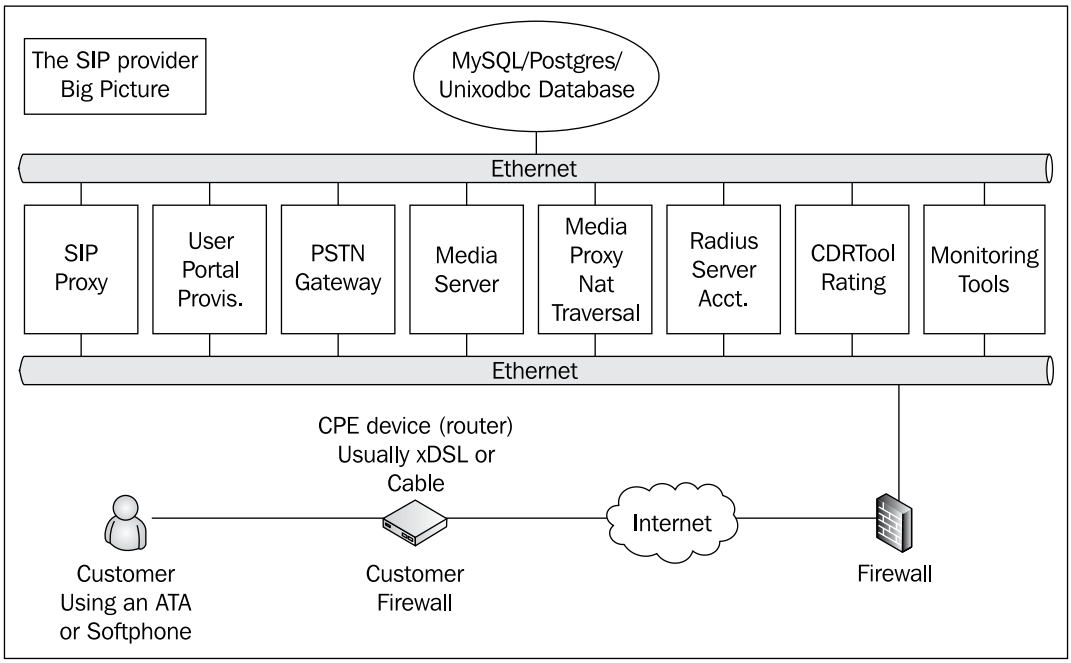
```
Session Initiation Protocol
  Status-Line: SIP/2.0 200 OK
  Message Header
  Message body
    Session Description Protocol
      Session Description Protocol Version (v): 0
    Owner/Creator, Session Id (o): root 11218 11218 IN IP4 8.8.1.4
        Owner Username: root
        Session ID: 11218
        Session Version: 11218
        Owner Network Type: IN
        Owner Address Type: IP4
        Owner Address: 8.8.1.4
      Session Name (s): session
    Connection Information (c): IN IP4 8.8.1.4
    Time Description, active time (t): 0 0
    Media Description, name and address (m): audio 17428 RTP/AVP 0 18 101
    Media Attribute (a): rtpmap:0 PCMU/8000
    Media Attribute (a): rtpmap:18 G729/8000
    Media Attribute (a): fmtp:18 annexb=no
    Media Attribute (a): rtpmap:101 telephone-event/8000
    Media Attribute (a): fmtp:101 0-16
    Media Attribute (a): silenceSupp:off - - - -
```

The SIP Protocol and the OSI Model

It is always important to understand the voice protocols against the OSI model to situate where each protocol fits.



The VoIP Provider "Big Picture"



Before we start to dig in the SIP proxy it is important to understand all the components for a VoIP provider solution. A VoIP provider usually consists of several servers and services. The services described here could be installed in a single server or multiple servers depending on the dimensioning.

In this book we will cover each one of these components, from left to right, in the chapters ahead. We are going to use this picture in all chapters to help you to know where you are.

SIP Proxy

The SIP proxy is the central component of our solution. It is responsible for registering the users and for keeping the location database (which maps IP to SIP addresses). The entire SIP routing and signaling is handled by the SIP proxy and it is responsible too for end user services such as call forwarding, white/blacklist, speed dialing, and others. This component never handles the media (RTP packets); all media-related packets are routed directly from the user agent clients, servers, and PSTN gateways.

User, Administration, and Provisioning Portal

One important component is the user administration and provisioning portal. In the portal, the user may subscribe to the service and should be able to buy credits, change passwords, and verify his or her account. On the other hand, administrators should be able to remove users, change user credits, grant, and remove privileges. Provisioning is the process used to make it easier, for administrators, to provide automatic installation of user agents such as IP phones, analog telephony adapters, and softphones.

PSTN Gateway

To communicate to the public switched telephone network, a PSTN gateway is required. Usually this gateway will interface to the PSTN using E1 or T1 trunks. The most common products in this arena are gateways from Cisco, AudioCodes, and Quintum. Asterisk is gaining market in this area, because of its price per port cost, sometimes 75% less than the competitors. To evaluate a good gateway, check the support of SIP extensions such as RFC3515 (REFER), RFC3891 (Replaces), and RFC3892 (Referred by). These protocols will allow unattended transfers behind the SIP proxy; without them in the gateway it might be impossible to transfer calls.

Media Server

The SIP proxy never handles the media. Services such as IVRs, voicemail, conference, or anything related to media should be implemented in a media server. SEMS SIP Express media server, developed by iptel, has some nice features such as conference, voicemail, and announcements. Once again, Asterisk can be used as a wildcard to provide these services.

Media Proxy or RTP Proxy for Nat Traversal

Any SIP provider will have to handle NAT traversal for its customers. The media proxy is an RTP bridge that helps the users behind symmetric firewalls to access the SIP provider. Without proxies it won't be possible to serve as much as 35% of the users. You can implement a universal NAT traversal technique using these components. The media proxy can help you too in the accounting correction for unfinished SIP dialogs, which, for some reason, didn't receive the BYE message.

RADIUS Accounting

A server with RADIUS installed will be fundamental for accounting the calls. A SIP provider should take maximum care of accounting records. OpenSER can be configured to send the accounting to a RADIUS server such as Radiator or FreeRADIUS. SIP calls can be accounted to a database as well. However, accounting to a database generates two records that need to be correlated manually.

CDRTool Rating

The RADIUS server has information about call duration, but does not have information about the rates and prices for the call. Applying prices to calls can be very tricky. We will use for our provider a GPL tool called CDRTool developed by AG projects (cdrtool.agprojects.com). It will be responsible for applying rates to calls.

Monitoring Tools

Finally we will need monitoring, troubleshooting, and testing tools to help debug any problems occurring in the SIP server. The first tool is the protocol analyzer and we will see how to use ngrep, ethereal, and tethereal. OpenSER has a module called SIP trace, which we will use too.

Where You Can Find More Information

The best reference for the SIP protocol is RFC3261. To read the RFCs is a little bit boring and sleepy (it is very good when you have insomnia). You can find the RFC at: `http://www.ietf.org/rfc/rfc3261.txt`.

A good SIP tutorial can be found at Columbia University: `http://www.cs.columbia.edu/~coms6181/slides/11/sip_long.pdf`. Together with this you can find a lot of information about SIP at `http://www.cs.columbia.edu/sip/`.

A very good tutorial can be found at the iptel website: `http://www.iptel.org/files/sip_tutorial.pdf`.

There is a mailing list where you can post questions about SIP called SIP implementors: `https://lists.cs.columbia.edu/mailman/listinfo/sip-implementors`.

Summary

In this chapter you have learned what the protocol SIP is and its functionality. You had the opportunity to get to know the SIP components such as the SIP proxy, SIP Registrar, User Agent Client, User Agent Server, and Gateway PSTN. You saw SIP architecture, its main messages and processes. Some places to find further information were listed too.

2
The SIP Express Router

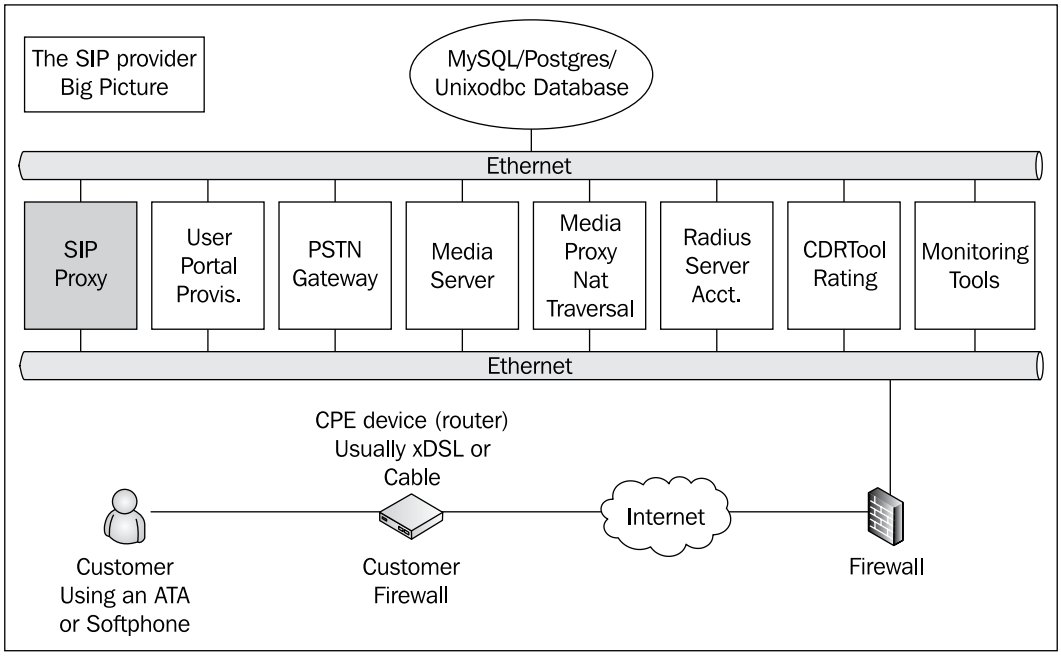
We discussed in the last chapter the big picture of a VoIP provider. Usually a VoIP provider is composed of several components. These components can reside in the same machine or be spread over several machines depending on your dimensioning. One of these components is the SIP proxy server, in our case the server running the OpenSER software. As the name implies, what best describes the SER is a SIP Router. It is able to manipulate the SIP headers and route packets at extremely high speeds. Third-party modules give SER extreme flexibility to play roles for which it was not originally intended, such as NAT traversal, IMS, Load Balancing, and other functionalities. In this chapter we will show you the possibilities and the architecture of the SIP Express Router.

By the end of this chapter you will be able to:

- Explain what the SIP Express Router (SER) is
- Choose between two available open-source projects, SER and OpenSER
- Describe their usage scenarios
- Distinguish between the different sections of the `openser.cfg` file
- Describe the processing of SIP messages
- Distinguish between loose and strict routing
- Distinguish between SIP and SDP

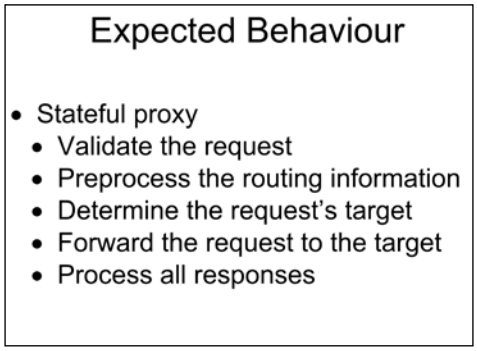
Where Are We?

The VoIP provider solution has many components. To avoid loosing the perspective, we will show this picture in every chapter. In this chapter, we are working with the SIP proxy component.



What is the SIP Express Router?

The SIP Express Router is an open-source SIP proxy server compliant to the IETF RFC3261 SIP protocol. It is targeted at large volume applications.



With its small footprint the SER is extremely fast to forward requests and can handle thousands of users in a single server. It is being used by large voice-over-IP providers and for embedded IP PBXes with very low processing power. Their interoperability with several sorts of equipment makes them a *de facto* standard.

What Software to Use, SER or OpenSER?

SER was originally developed by the FhG Fokus research institute in Berlin, Germany, and released under the GPL license. The core developers of SER were Andrei Pelinescu-Onciul, Bogdan-Andrei Iancu, Daniel Constantin Mierla, Jan Janak, and Jiri Kuthan. Some contributors joined the project later, namely Juha Heinamen (RADIUS, ENUM, DOMAIN, URI), Greg Fausak (POSTGRES), Maxim Sobolev (NATHELPER), Adrian Georgescu (MEDIAPROXY), Elena Ramona Modroiu (XLOG, DIAMETER, AVPOPS, SPEEDDIAL), Miklos Tirpak (Permissions), and others.

OpenSER is a fork of the original SER project. In 2004 FhG Fokus started a spinoff of the SER project creating the `iptel.org`. In 2005 the commercial variant of IPtel was sold to TEKELEC. The core development team was split in two. Three of them went to `iptel.org` (Andrei Pelinescu-Onciul, Jan Janak, and Jiri Kuthan). The other two (Bogdan Andrei Iancu and Daniel Constantin Mierla) left the FhG to start a company called *Voice-System* the main maintainer of the OpenSER project started in 2005.

This book started in late 2005 based on the SER project. At that time, I was interested in a NAT traversal solution that was only available using SER. The scalability of *Asterisk* was not good enough to host a SIP provider, and so I started playing with SER. The documentation was really hard to understand and I started writing my own to train the administrators of the SIP providers.

After the eBook was ready, I found that the SER project was almost halted. Most of the code dated to 2003. After a little research I found the OpenSER project. It seemed to be more active, with newer modules and more frequent releases. I was able to change everything to OpenSER in very little time.

I don't want to get into the politics of SER versus OpenSER. The concepts presented here are valid for both. The fact is that is written for OpenSER.

OpenSER has a flexible plug-in model for third-party applications. Applications can be easily created and plugged in to the server. This plug-in model, allowed the development of several new modules, such as RADIUS, DIAMETER, ENUM, PRESENCE and SMS to name a few. Newer modules are being added every month. You can check available modules for OpenSER 1.2.x at `http://www.openser.org/docs/modules/1.2.x/`.

The performance and robustness of OpenSER allows it to be used to serve millions of users. On a recent performance report dated 14th March 2007, OpenSER 1.2.x was able to handle register requests to an equivalent of 4 million users. The TM (Transaction Module) was able to handle 28 million calls per hour. The complete report can be seen at:

`http://www.openser.org/docs/openser-performance-tests/`

OpenSER is not used just by service providers. It can be used to construct SIP appliances. There are SIP firewalls, Session Border Controllers, and load balancers that are using code borrowed from the OpenSER project today. OpenSER was chosen by LINKSYS for the One PBX platform, probably because of the small footprint and high performance available.

OpenSER is flexible, portable, and extendable. Having being developed in ANSI C it can be easily ported to any platform. It is very easy to extend by creating new modules using C language. Recently new layers of programming were added. It is possible to use Call Processing Language to simplify the routing scripts and Perl to process requests in real time. WeSIP is an application program interface that allows you to use Java and servlets to extend the OpenSER server creating a SIP application server. Check WeSip at `www.wesip.com`.

Usage Scenarios

OpenSER is primarily used as a SIP proxy and Registrar. However, it can be used in some other applications such as Proxy dispatches, Jabber Gateway, NAT Traversal together with MediaProxy and RTPproxy. It supports IP versions 4 and 6 and is able to serve multiple domains. OpenSER can be executed in Linux, Solaris, and FreeBSD platforms.

OpenSER was created to be a SIP proxy. However, with the addition of new modules, now OpenSER can be used in a several scenarios such as:

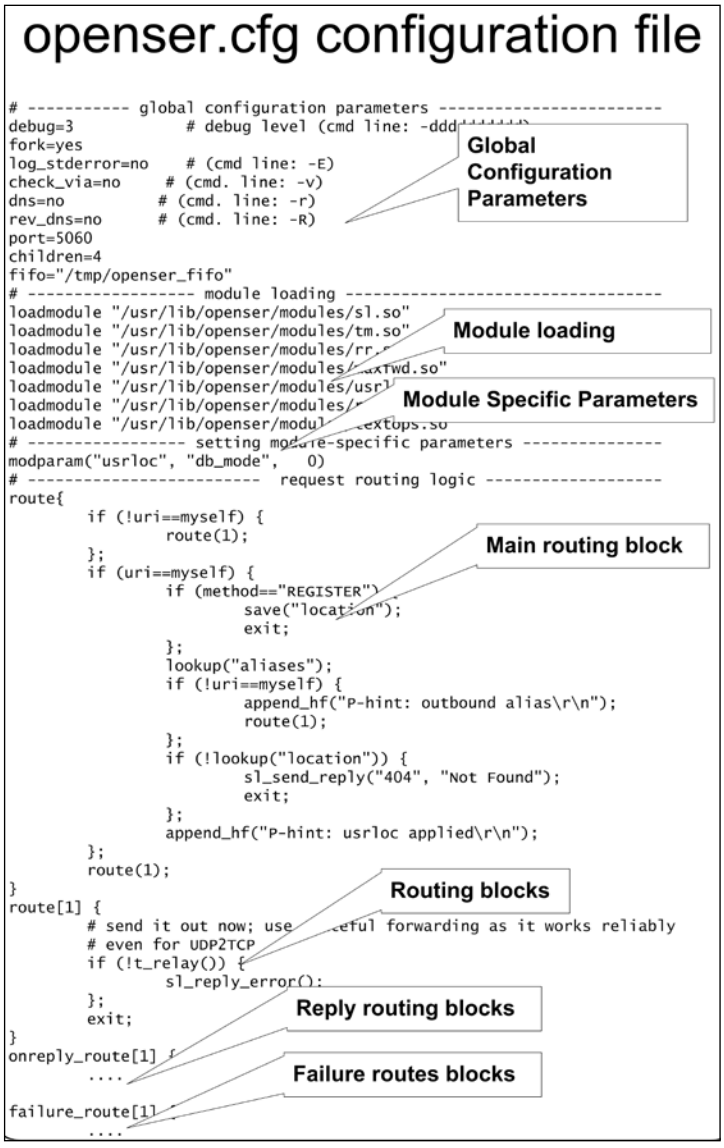
Modules	Functionality
DISPATCHER, PATH	Load balancing
MEDIAPROXY, RTPPROXY, NATHELPER	Nat Traversal
PRESENCE	Presence Server
IMC, XMPP	Instant Messaging

Let's see the most common usage scenarios for OpenSER. In all these scenarios OpenSER works like glue that binds all the SIP components together.

- VoIP providers
- Instant Messaging providers

- SIP Load Balancing
- Embedded IP PBX
- NAT Traversal
- SIP.EDU

OpenSER Architecture



Core and Modules

OpenSER is built on top of a core that is responsible for the basic functionality and handling of SIP messages. The modules are responsible for the majority of OpenSER functions. OpenSER modules expose their functionality inside OpenSER with new commands and parameters used inside scripts. OpenSER is configured in a file called `openser.cfg`. This configuration file controls which modules are loaded and their respective parameters. All the SIP flow is controlled too in several routing blocks defined in the file. The file `openser.cfg` is the OpenSER main configuration file.

Sections of the File openser.cfg

The `openser.cfg` file has seven sections:

- **Global definitions**: This portion of the file contains several working parameters for OpenSER including the listening `ip:port` pair for the SIP service and debug level.
- **Modules**: Contains a list of external libraries required to expose the functionalities not available in the core. Modules are loaded with `loadmodule`.
- **Modules configuration**: Modules have parameters that needs to be set appropriately. These parameters are configured using `modparam(modulename, parametername, parametervalue)`.
- **Main routing block**: The main routing block is where the SIP message processing starts. It controls the processing of each message received.
- **Secondary routing blocks**: The administrator can define new routing blocks using the command `route()`. These routing blocks works like subroutines in the OpenSER script.
- **Reply routing blocks**: Reply routing blocks are used to process reply messages, usually `200 OK`.
- **Failure routing blocks**: Failure routing blocks are used to process failure conditions such as busy or timeout.

This file will be covered in detail in the Chapters 4, 5, 6, 7, 8, and 9.

Sessions, Dialogs, and Transactions

It is important to understand some SIP concepts used in OpenSER processing:

- **SIP transaction**: A SIP message including any resends and their direct responses (that is, REGISTER and 200 OK).
- **SIP dialog**: A relation that exists for some time between two SIP entities (that is, a dialog established between two UACs from the INVITE until the BYE message).
- **SIP Session**: A media flow (`audio/video/text`) between two SIP entities.

openser.cfg Message Processing

The `openser.cfg` is a script executed for each SIP message received. For example: If the userA wants to talk to userB it sends an INVITE message. This message is processed in the main routing block. The processing continues until it finds a `t_relay()` (forward) or an `sl_send_reply` (send an error message) or eventually discards the message at the end of the block using the `exit()` command.

SIP Proxy—Expected Behavior

It is important to understand the basic processing of a SIP proxy according to RFC3261. Without this understanding it will be very difficult to configure a Proxy server.

Each proxy will take routing decisions, modifying the request before sending it to the next element. The responses will be routed over the same set of proxies traversed by the original request in the reverse order.

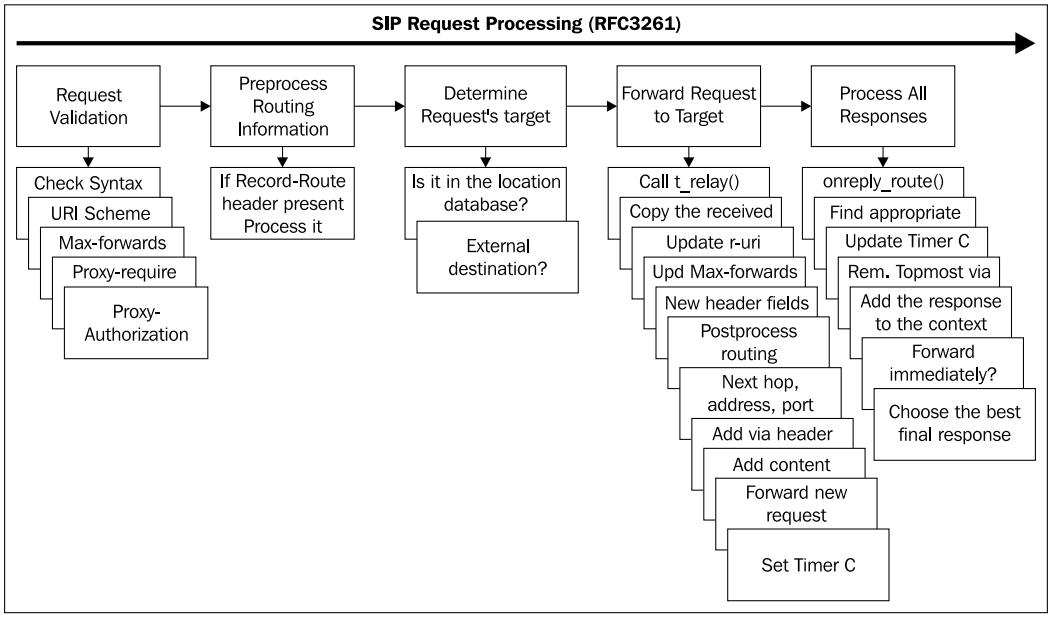
A SIP proxy can operate in stateless or stateful mode. When a SIP proxy works as a simple SIP packet forwarder, it forwards the packets to a single element determined by the request. A proxy working in stateless mode discards any information about the message after the message has been forwarded. This characteristic limits the failure treatment and billing.

When OpenSER knows that the message 200 OK corresponds to a specific INVITE we say that it is working in stateful mode. This means simply that you can now manage the response in an `onreply_route()` block. With stateless processing each message is handled without a context. Stateless processing is used in applications like load balancing; it uses the command `forward()` in the script.

When you need more sophisticated resources like billing, call forward, and voicemail, you will need to use stateful processing. Each transaction will be maintained in memory and failures, responses, and retransmissions will be associated with this specific transaction. Stateful transactions are handled by the TM (transaction) module and usually use the `t_relay()` command.

An often misunderstood concept is that the processing is stateful by transaction and not by dialog. Thus, it is the stateful processing of an INVITE request until the 200 OK response (transaction) and not from the INVITE to the BYE request (dialog).

Stateful Operation



This is a simplified description of the stateful operation. You will find a complete and more detailed description in the RFC3261 text. There is a close resemblance between the `openser.cfg` sections and the figure above. However some processes are manual, such as to check the Max-forwards header, while others are encapsulated in a single command. To illustrate, when you call `t_relay()` all the forward request processing as described is done automatically.

When operating in stateful mode, a proxy is simply a SIP transaction processor and all these processing steps are required:

- Validate the request
- Pre-process the routing information
- Determine the request's target
- Forward the request to the target
- Process all responses

A stateful proxy creates a new server transaction for each new request received. Any retransmissions of the request will then be handled by that server transaction.

Example: For each request traversing our SIP proxy we will:

Step 1: Request validation

- Check the message size to avoid buffer overflows.
- Check the Max-forwards header to detect loops.

Step 2: Routing information pre-processing

- If there is a record-route header, process it.

Step 3: Determine the request target

- Is it in the location database (registered users)?
- Is there a route to the destination (gateway destinations)?
- Is it directed to an external domain (external addresses)?

Step 4: Request forwarding

- Call the function `t_relay()` and OpenSER will do all the job for you statefully.

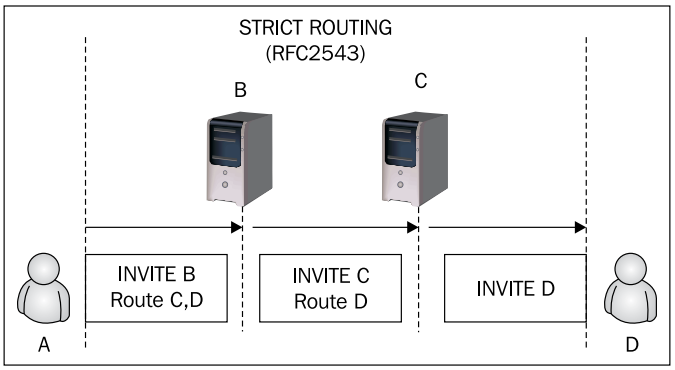
Step 5: Response processing

- Usually this is done automatically by OpenSER. Sometimes you can use the `onreply_route[]` section to handle some response. Example: in a "call forward on busy" scenario, we could use the response 486 (Busy) to direct the call to a voicemail server.

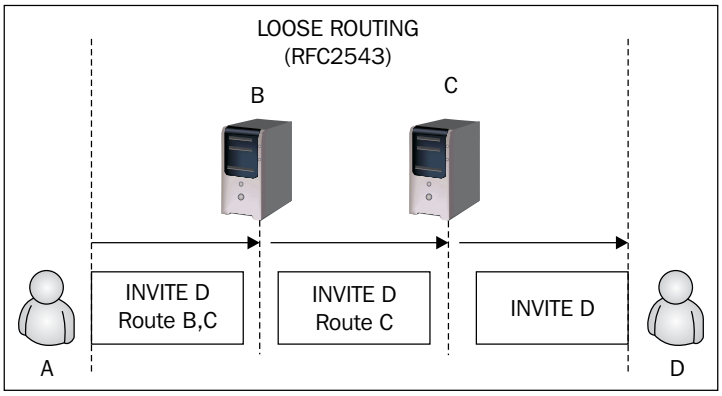
Differences between Strict Routing and Loose Routing

Loose and strict are different methods of routing SIP messages. Loose routing is new in SIP version 2. When you use loose routing, the R-URI is never changed and backwards compatibility is maintained with the older method (strict routing RFC2543).

The problem with strict routing is in the process of specifying the entire proxy set in the initial request before starting the SIP dialog. The processing throws away the information contained in the received R-URI. The behavior of UAs with outbound-proxy was problematic. The whole system would fail if there was a failure in one of the elements.



The solution, is that loose routing is the correct method. It keeps the target separated from the route. It allows each destination to route the packet and has a mechanism to keep backward compatibility with strict routing. The support of loose routing is indicated by the parameter `;lr`.



When the SIP server receives a message, it can decide if it wants to stay in the middle or not (record-route). If the SIP server does not want to stay in the middle, it can pass the information to the user agents' UAs to connect each other. After this process the messages follow between the user agents.

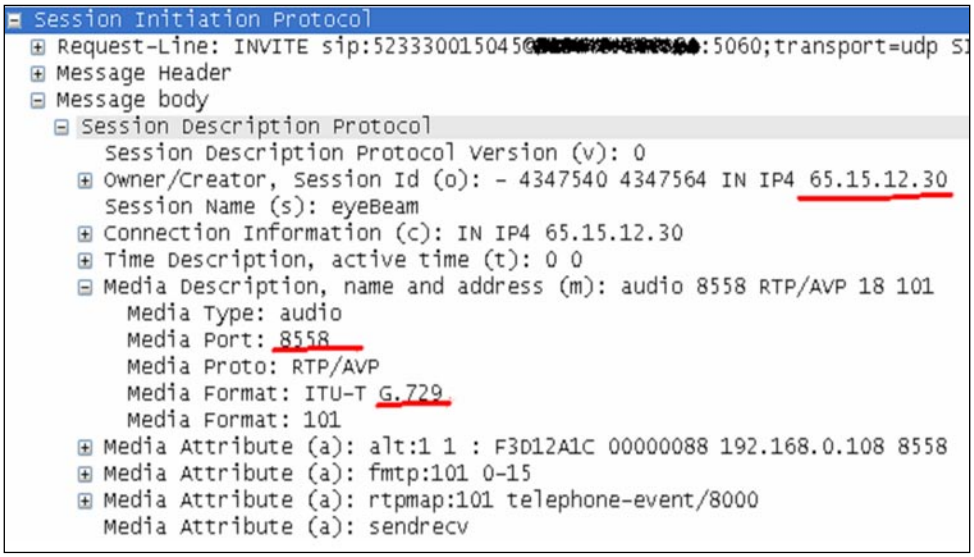
If OpenSER wants to stay in the middle of the conversation (that is, for billing purposes) it should insert a ROUTE header field using the function `record_route()`.

Understanding SIP and RTP

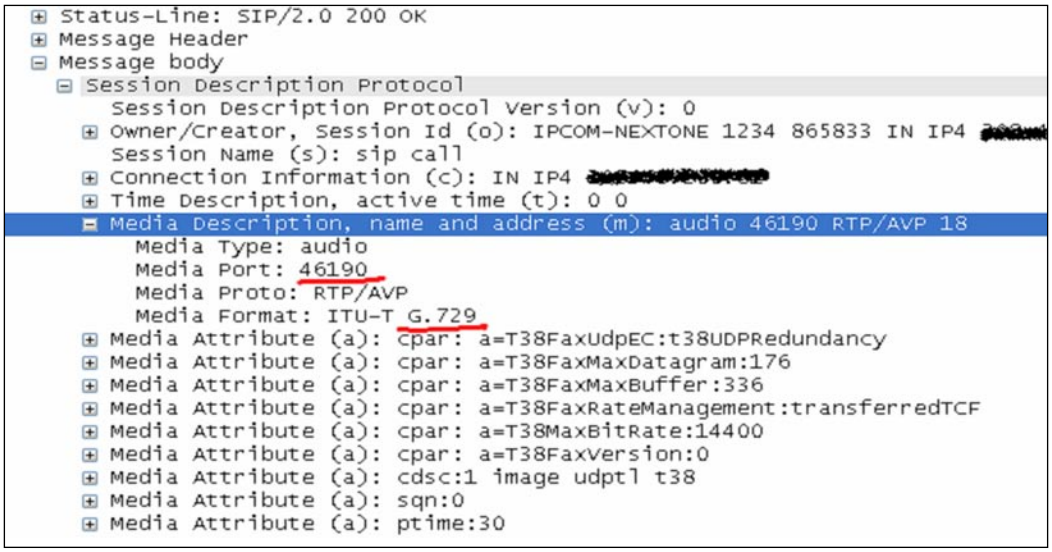
To understand the following subsections, you should understand some things about SIP and RTP. First, SIP is a signaling protocol that controls the call with methods such as INVITE, BYE, and CANCEL. The SIP protocol includes in the INVITE request information about the session (audio/video/text) using a protocol called SDP (Session Description Protocol). The information contained in the SDP describes one or more media flows configured between two user agents.

A SIP proxy never participates in the media flow, thus it is media agnostic. In other words it supports calls with whichever media are specified by UA and gateways. However, sometimes a B2BUA (back to back user agent) such as mediaproxy can be installed at the same server to treat RTP audio (that is, NAT traversal mechanism). The SDP protocol works in an Offer/Answer model. The SDP offer is embedded in the INVITE request and the answer in the 200 OK response.

Example: Excerpt from Ethereal:



The packet described in the preceding figure is an INVITE request. The INVITE request embeds the SDP packet that describes the session information. We can see there the INVITE generated on an eyeBeam phone. It is offering to use the Codec G.729 at the UDP port 8558 for Audio (I conceal the IP address for security reasons). The attribute rtpmap:101 telephone-event/8000 describes the DTMF forward method being used (RFC2833). The other device, in this case a gateway, answers the offer in the 200 OK reply.



Summary

In this chapter we have learned what OpenSER is and its main characteristics. Now you can identify the `openser.cfg` configuration file and its configuration blocks such as global definitions, load modules, module's parameters, main routing block, routing blocks, reply routing block, and failure routing block. Each request accepted by the proxy is processed according to the `openser.cfg` script. The script is organized almost in the same sequence as the SIP stateful proxy processing. Usually OpenSER operates as a loose router (SIP version 2). At the end we presented concepts about SIP and SDP.

3
OpenSER Installation

The installation is the beginning of the work. It is very important to install OpenSER correctly from the source code. You can install much faster from the Debian packages or using the `apt-get` utility. However, the installation from the source code is much more flexible allowing you to select which modules will be compiled. You cannot install RADIUS accounting support from the Debian Packages. That's why we won't use any shortcut to the installation. I strongly advise you to install using Debian.

If you choose to install on another platform, you will have to deal with init scripts and fixing the installation of the other packages.

By the end of this chapter you will be able to:

- Install Linux prepared for OpenSER
- Download OpenSER source and its dependencies
- Compile and install OpenSER with MySQL and RADIUS support
- Start and stop OpenSER
- Configure the Linux system to start OpenSER at boot time.

Hardware Requirements

There are no minimum hardware requirements for OpenSER. It will run in an ordinary PC. The best bets we have are from performance tests realized on the 1.2 version. A PC with the following specifications was capable of 28 million complete calls per hour. The testing server was an ordinary desktop, Intel Core2 CPU 6300 @ 1.86GHz, 1GB of memory, 100Mbs Ethernet card. Unfortunately, there are currently no formulas for OpenSER dimensioning. The correct hardware (CPU and memory) must be obtained empirically.

Software Requirements

The OpenSER software runs in a variety of Linux, BSD, and Solaris platforms. Some generic packages are available for some varieties of Linux and Solaris. These packages can be downloaded from `www.openser.org`. The following packages are required to compile OpenSER.

- gcc
- bison or yacc (Berkley yacc)
- flex
- GNU make
- GNU tar
- GNU install

Some modules such as MYSQL, POSTGRES, RADIUS, and others will require additional packages to compile. They will be presented in their respective chapters.

Lab—Installing Linux for OpenSER

All of these labs were prepared using a VMware virtual machine with Debian Etch 4.0 installed. We have used as the Linux distro the Debian Etch, which can be downloaded from:

`http://cdimage.debian.org/debian-cd/4.0_r0/i386/`

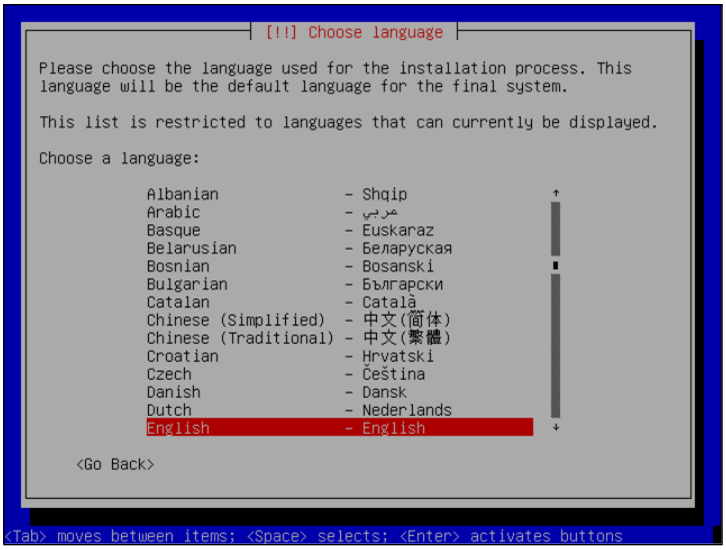
Warning

The instruction for this lab formats the computer. Back up all the data before proceeding or run in some virtual machine such as VMware or XEN.

Step 1: Insert the CD and boot the computer using the Debian Etch 4.0 CD. Press **ENTER** to start the installation.

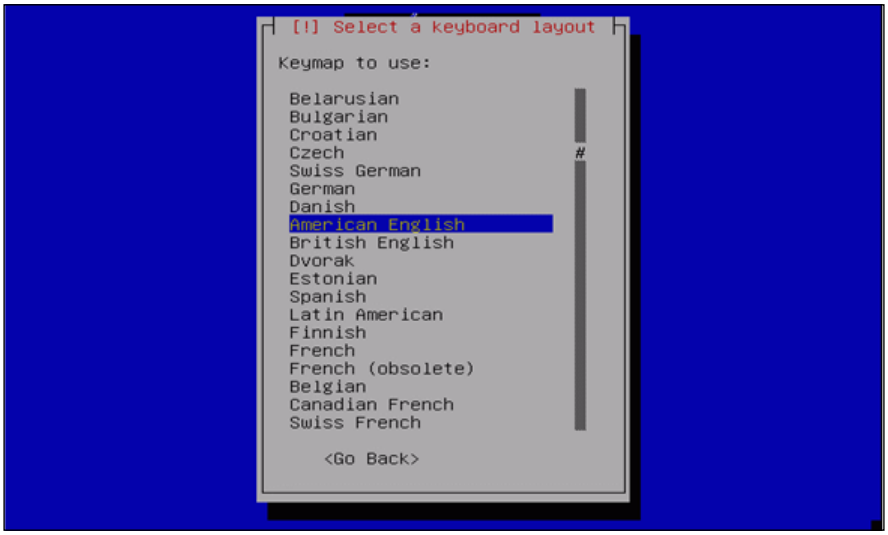
In this screen, you can also select boot and installation options. Sometimes you will need to choose some hardware-specific parameters for your installation. Press *F1* for help if needed.

Step 2: Choose a language.



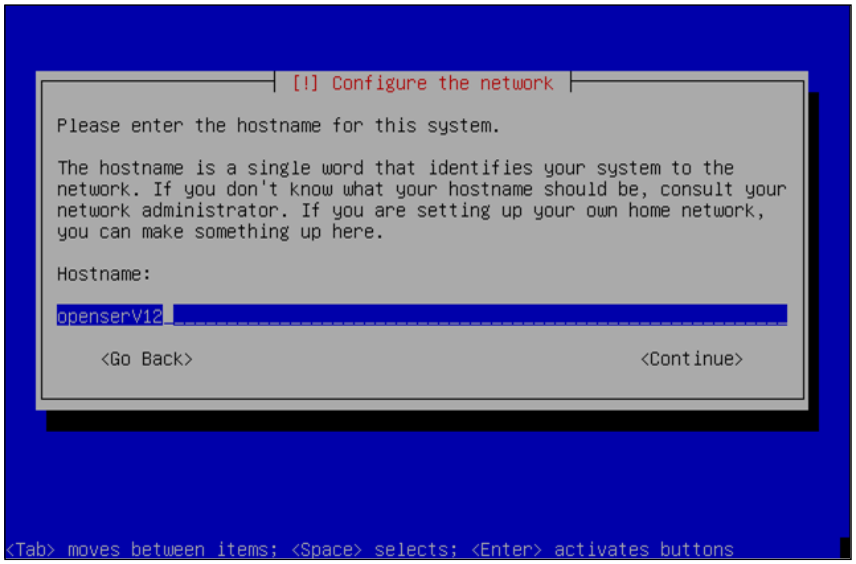
Choose the language of your preference for the use in the installation process.

Step 3: Choose the **keyboard layout**.



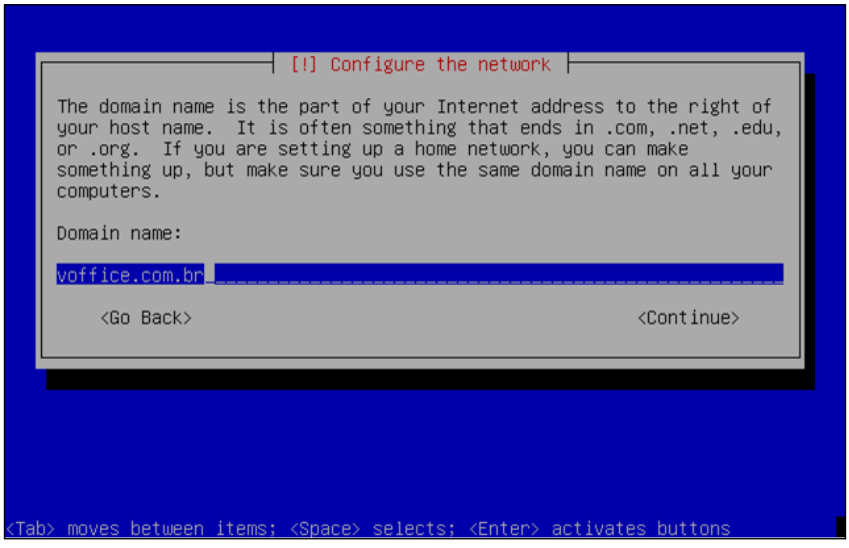
It is very common to have to choose a keyboard layout, mainly in European and Asian countries.

Step 4: Choose the **Hostname**.



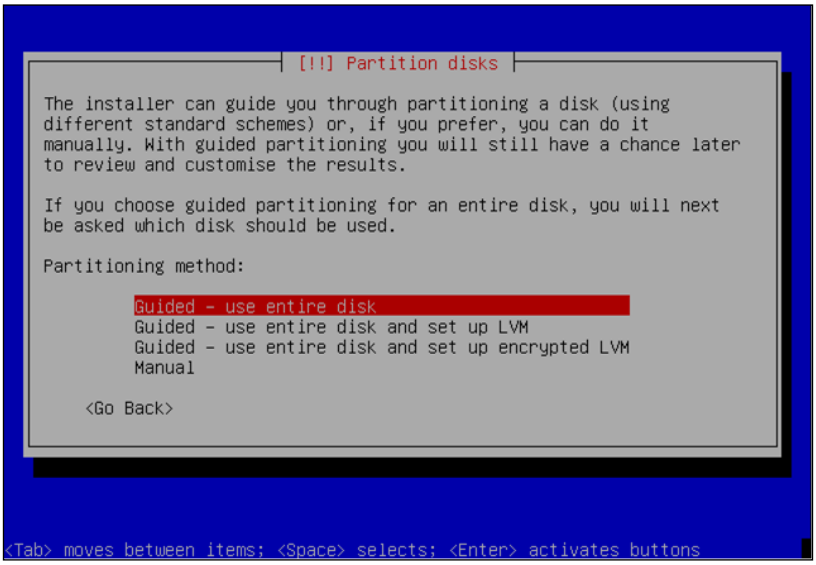
Choose the name of the server. It is important because later you will use this name to access the server.

Step 5: Choose your **Domain name**.



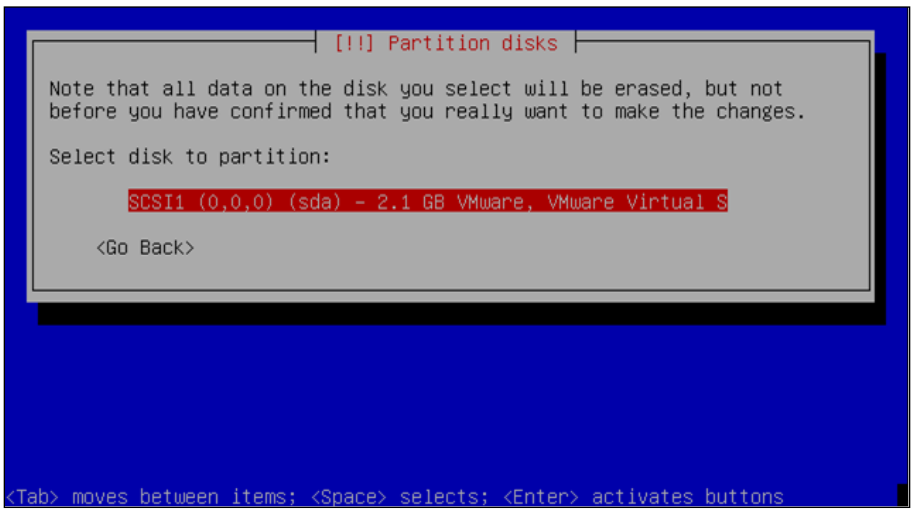
The domain name is obvious, but important, because OpenSER use domains to distinguish users, so be sure to anwer correctly this screen.

Step 6: Choose a **Partitioning method**.



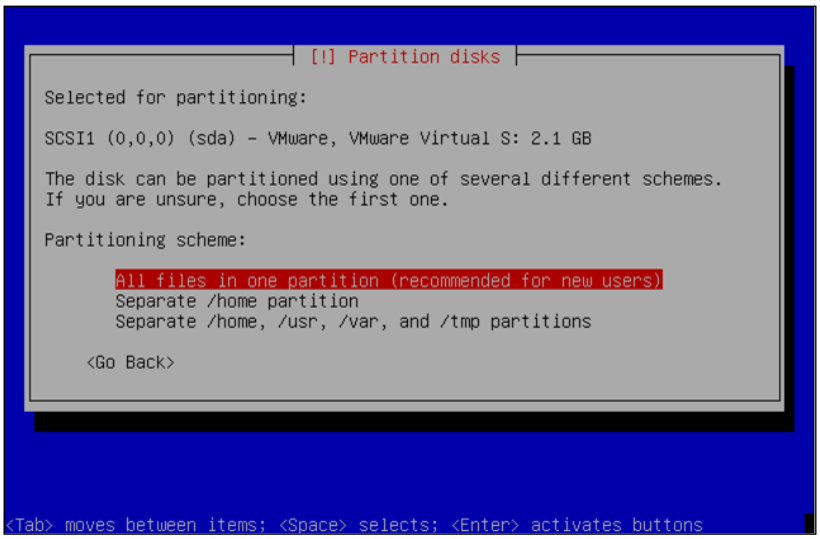
We could write a whole chapter about partitioning. Linux geeks, certainly, will use the manual option. For the purposes of learning, you can simply use entire disk. Consult a Linux specialist for the best partioning scheme for your server.

Step 7: Select disk to partition.



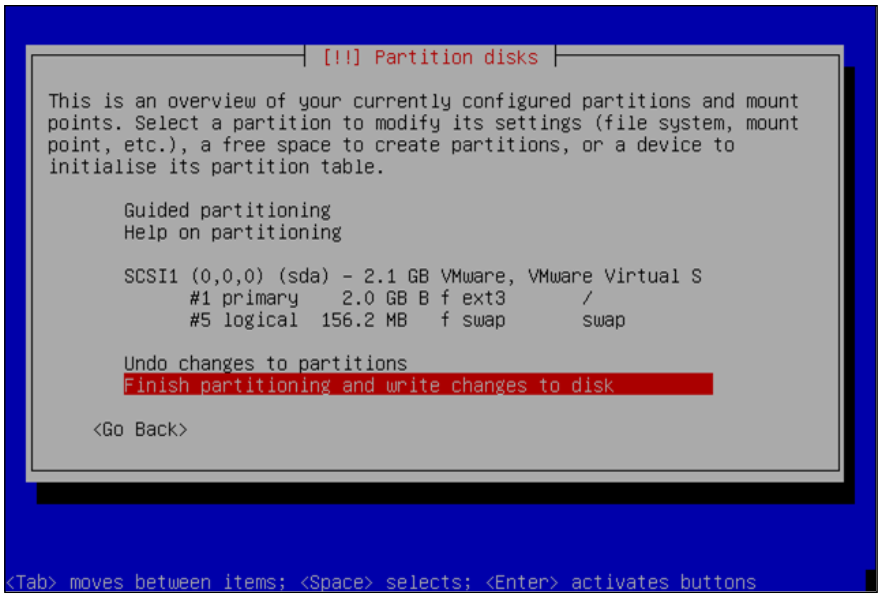
Now, just select the disk being used to install Linux.

Step 8: Select all files in one partition.



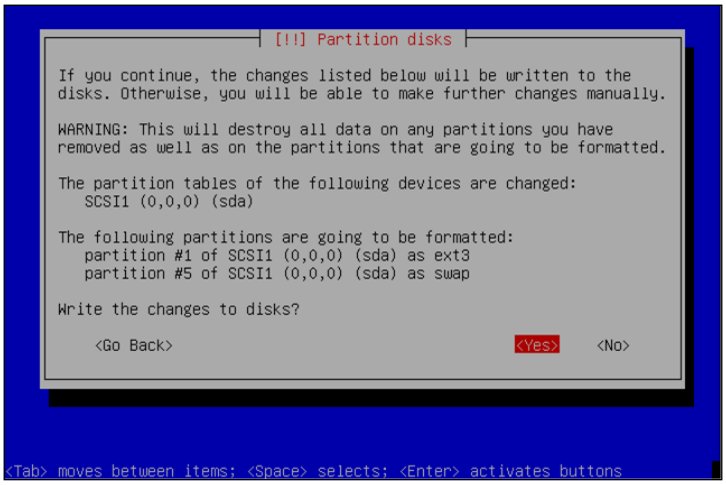
Again, you can choose how to partition the system. Let's stick with the default installation again. Some advanced users may want to change it a bit.

Step 9: Finish the partition changes to disk.



Now, just finish the partitioning step and write changes to the disk. Never do it if you want to preserve your disk. After the partitioning, all the pre-existing content of the disk will be erased. So do it wisely. I used VMWare to test OpenSER; it is free and creates a virtual machine, where I can work safely.

Step 10: Write changes to the disk.

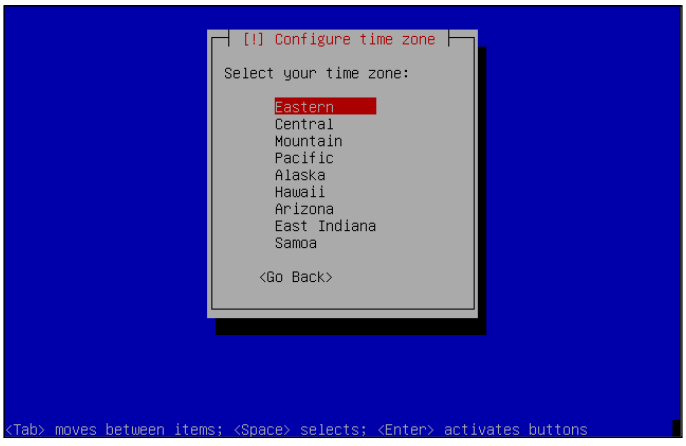


Now, it comes to the scary part. Confirm that you want to erase all the content of the disk. Well, think twice, or even three times before saying "Yes".

Warning

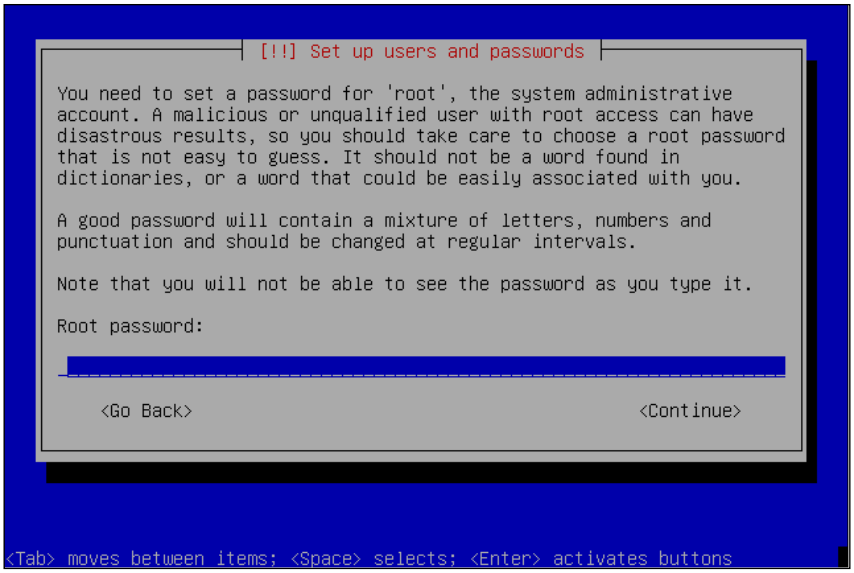
All data on the disk will be destroyed!

Step 11: Configure the time zone.



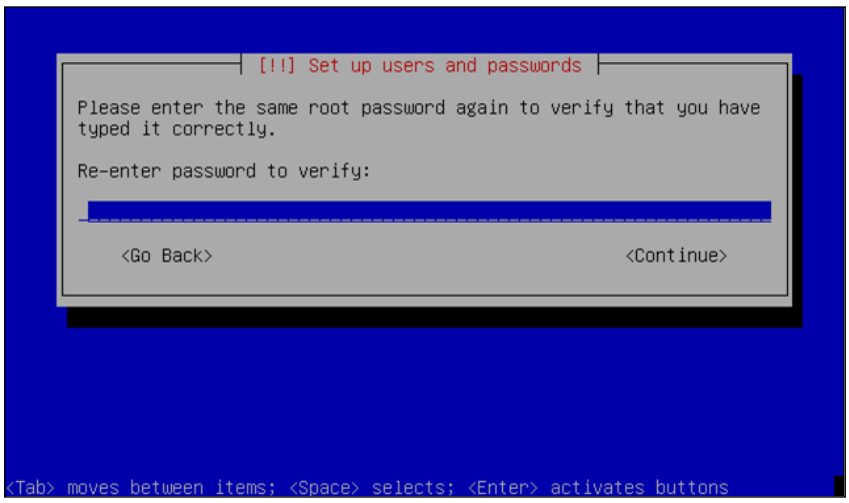
Select the time zone. It is important to have the correct time zone, mainly for reports. If you don't do it correctly, you will end up with voicemail messages with the wrong time.

Step 12: Set the **Root password** to "openser".



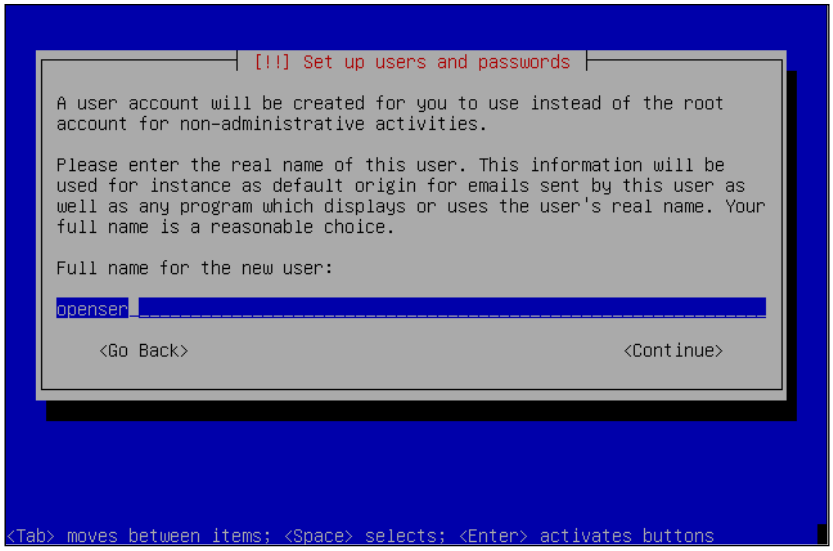
Choose a password for your root user. This is the most important password on the system.

Step 13: Re-enter password to verify.



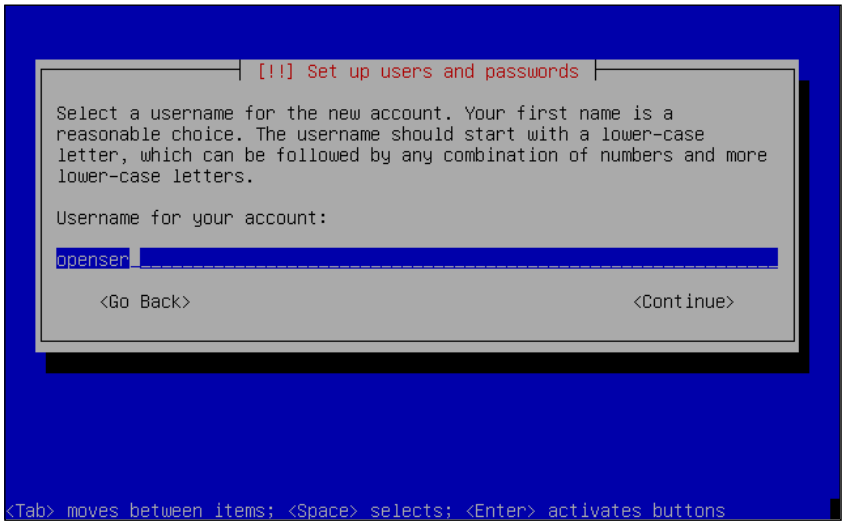
Please, re-enter the password for confirmation purposes. Try to use a password hard to crack (8 characters minimum, letters, numbers, and some kind of special character such as "*"or "#").

Step 14: Enter the full name for the user account as "openser".



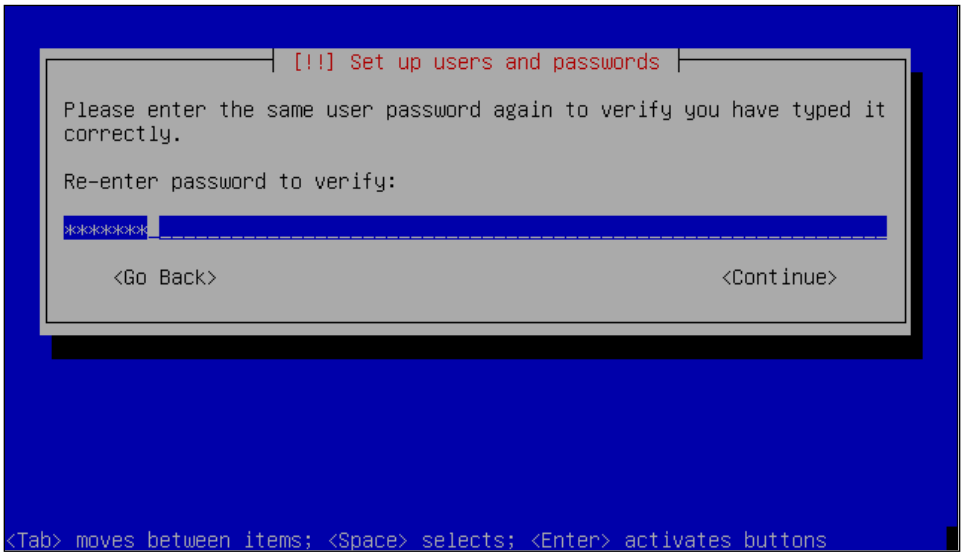
Some systems require you to create at least one user. Let's do it, starting with the **full user name**.

Step 15: Enter the user name for user account as "openser".



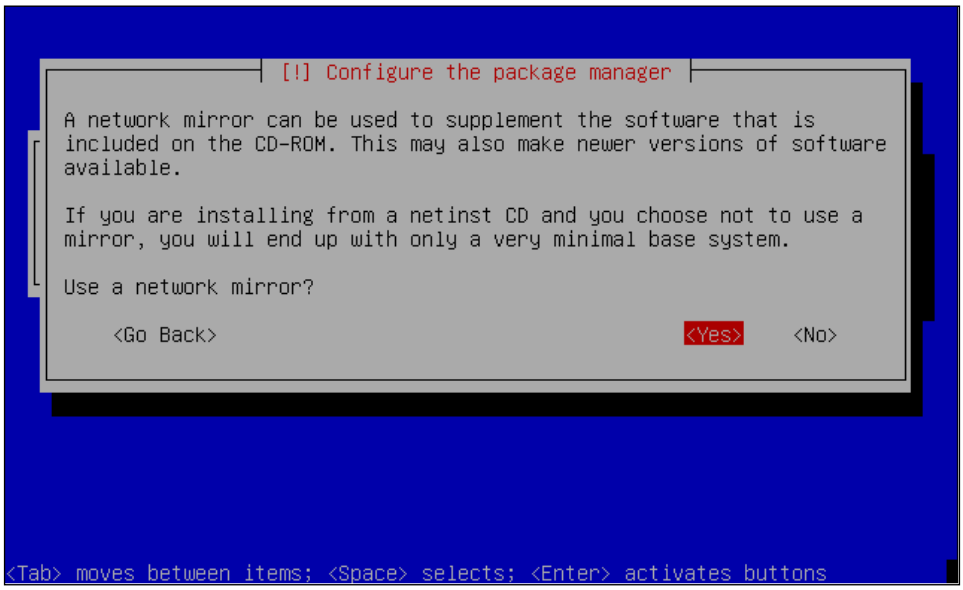
Now the name used to log on the user on the system.

Step 16: Enter the password for the user account "openser" and re-enter to confirm.



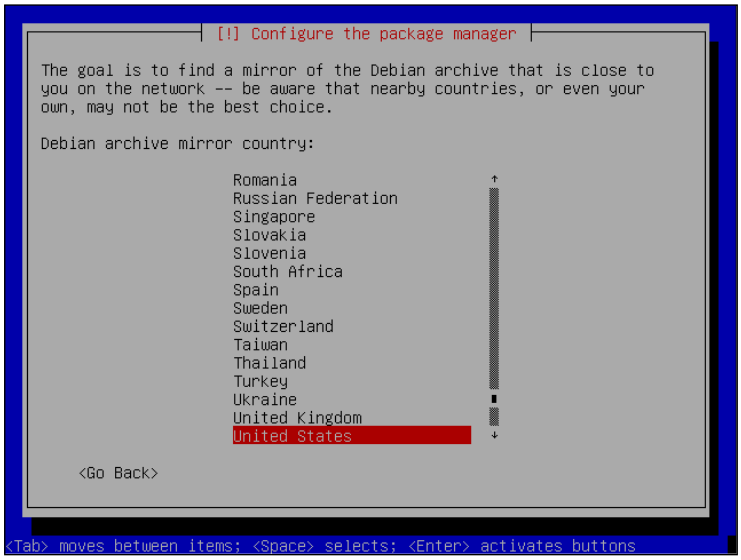
Enter the password and confirm it. Again, try to use a password hard to crack.

Step 17: Configure the package manager. Select **Yes** to use a mirror.



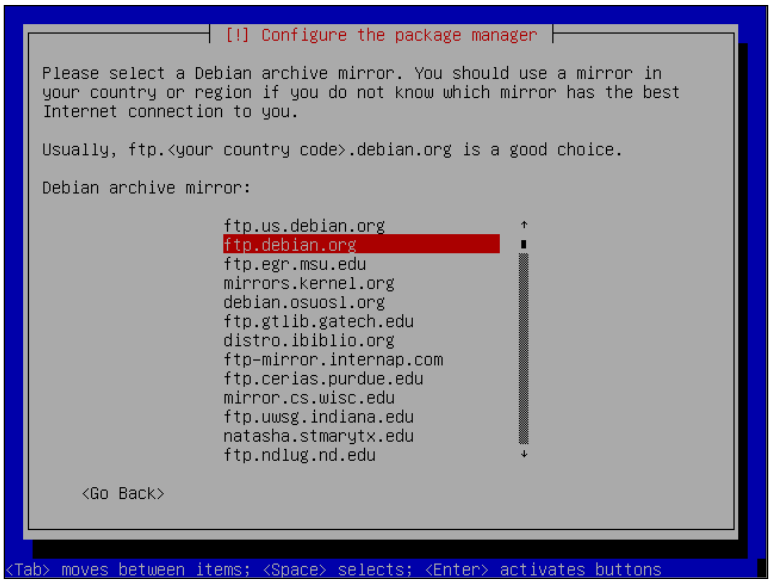
During the process of installation, we will use several packages distributed by Debian.

Step 18: Select a **mirror country**.



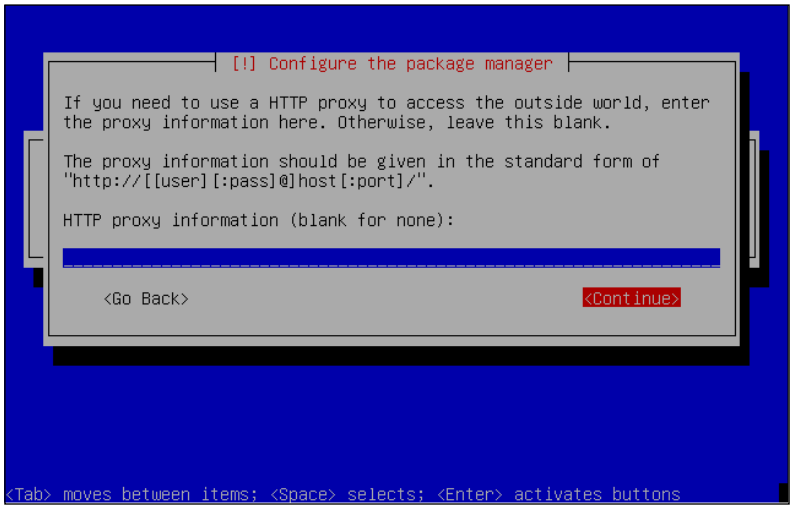
This screen will allow you to select from where you will download the packages.

Step 19: Select **ftp.debian.org** or your preferred mirror.



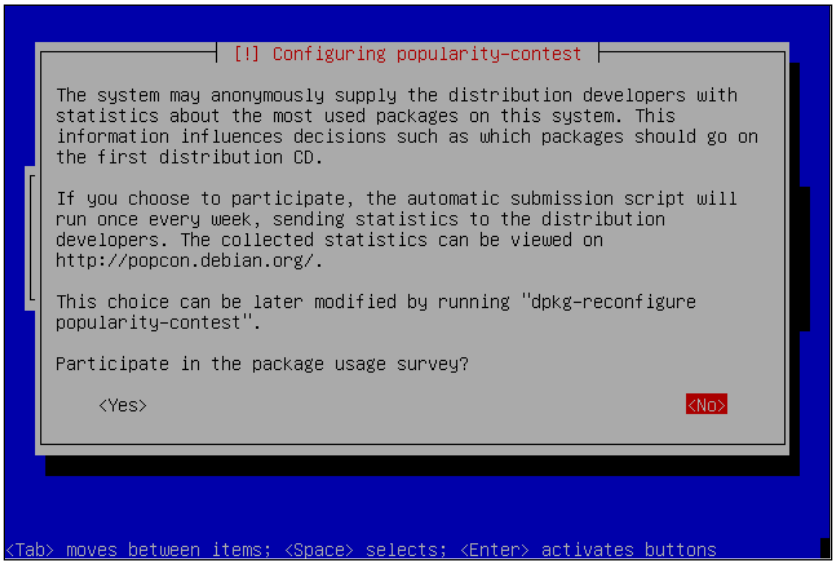
Select the nearest one to speed up the download of the packages.

Step 20: Leave the HTTP proxy blank or fill with the appropriate parameters if you use an HTTP proxy.



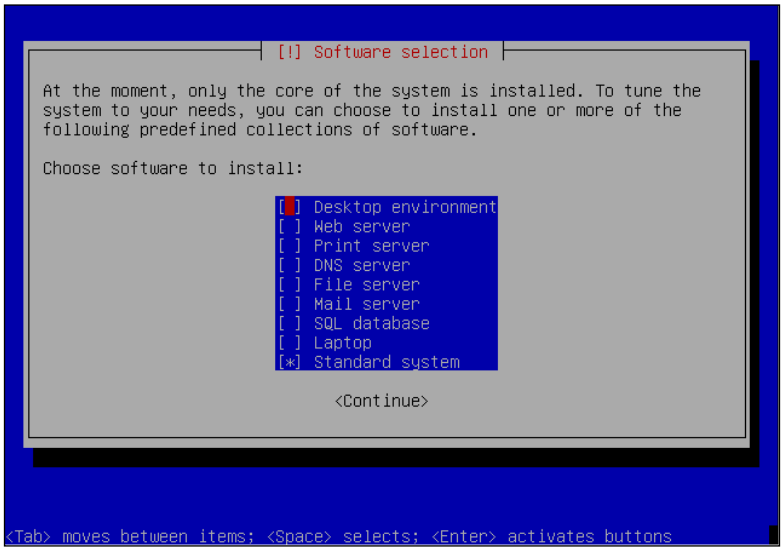
If you use an HTTP proxy such as Squid or Microsoft ISA Server, please fill in the appropriate parameters to allow internet access for the downloads.

Step 21: Select **Yes** if you want to to participate in the package popularity survey, or **No** if you don't.



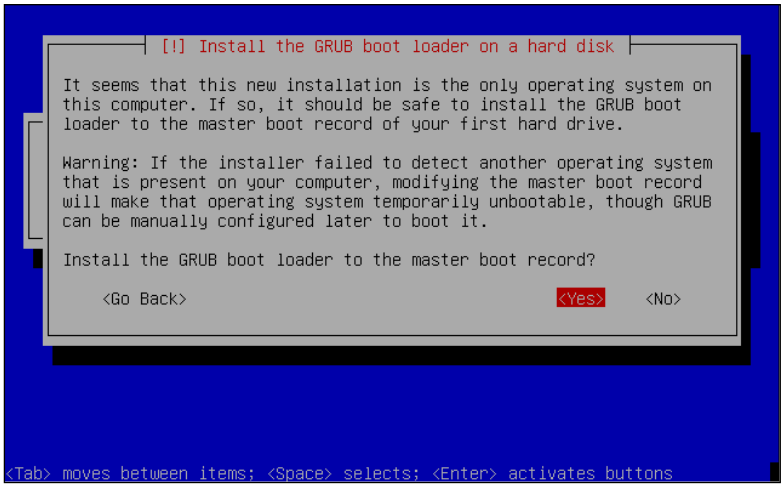
The popularity package survey generates statistics about the most downloaded packages.

Step 22: Select **Standard system**.



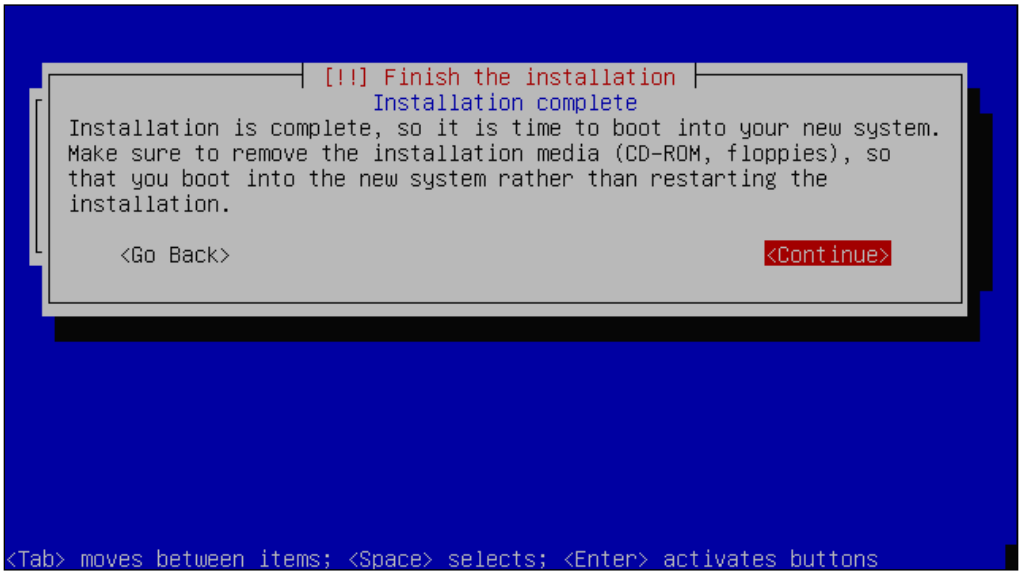
Debian comes in several pre-defined installations such as **Desktop**. The desktop installation, as an example installs a GUI for Linux such as GNOME or KDE. We don't need this for our installation. So please choose just **Standard system**. Later we will install manually components such as the Web Server, Mail Server, and SQL Database.

Step 23: Select **Yes** to install the GRUB boot loader.



GRUB is a boot load manager for your server. It allows you to dual boot systems and to do some tricks during the boot process.

Step 24: Finish the installation.



Finish the installation and boot the system.

The system will reboot automatically.

Step 25: Just after the reboot install SSH.

```
apt-get install ssh
```

Downloading and Installing OpenSER v1.2

Even though it is easier to install the OpenSER using the Debian packages we will go through the compilation process. It is more flexible and we may need to recompile OpenSER a few times in this material to include other modules. The installation process step by step follows:

Step 1: Install the dependencies.

```
apt-get install gcc bison flex make openssl libmysqlclient-dev
libradiusclient-ng2 libradiusclient-ng-dev mysql-server
```

The MySQL server is not really a dependency, but we will install it at this moment to make things easier.

Step 2: Download the source package and decompress it.

```
cd /usr/src
wget http://www.openser.org/pub/openser/1.2.2/src/openser-1.2.2-tls_
src.tar.gz
tar -xzvf openser-1.2.2-tls_src.tar.gz
```

Step 3: Use your favorite Linux editor to edit the Makefile

Remove from the "exclude_modules?=" line the `mysql` and any `radius`-related modules. This will make the compilation process include MySQL and RADIUS.

```
cd /usr/src/openser-1.2.2-tls/
vi Makefile
```

File before making changes:

```
exclude_modules?=       jabber cpl-c mysql pa postgres osp unixodbc \
                                          avp_radius auth_radius\
                                    group_radius uri_radius xmpp \
                                  presence pua pua_mi pua_usrloc \
                                         mi_xmlrpc perl snmpstats
```

File after making changes:

```
exclude_modules?=              jabber cpl-c pa postgres osp unixodbc \
                                                          xmpp \
                                        presence pua pua_mi pua_usrloc \
                                               mi_xmlrpc perl snmpstats
```

Step 4: Compile and install the core and modules.

```
cd openser-1.2.2-tls
make prefix=/ all
make prefix=/ install
```

Step 5: Make the required adjustments:

```
mkdir /var/run/openser
```

Lab—Running OpenSER at the Linux Boot

Step 1: Include openSER in the linux boot.

```
cd /usr/src/openser-1.2.2-tls/packaging/debian
cp openser.default /etc/default/openser
cp openser.init /etc/init.d/openser
update-rc.d openser defaults 99
```

Step 2: Edit the `/etc/openser/openser.cfg` file and remove the line `fork=no` (even if it has C-style remarks). The init script looks for the instruction `fork=no`, even if commented.

Step 3: Make sure that the script `openser.init` has the necessary permissions

```
cd /etc/init.d
chmod 755 openser
```

Step 4: Edit `/etc/default/openser.cfg`, change the memory parameter to 128MB and the RUN_OPENSER to yes.

Step 5: Edit the init script to make sure that the daemon is pointing to the right directory:

```
vi /etc/init.d/openser
```

File before making changes:

```
DAEMON=/usr/sbin/openser
```

File after making changes:

```
DAEMON=/sbin/openser
```

Step 6: Restart the computer to see if OpenSER starts. Confirm using:

```
ps-ef |grep openser.
```

It is highly recommended that you change the username and password used to run openser in the `/etc/init.d/openser` file.

OpenSER v1.2 Directory Structure

After the installation, OpenSER will create a file structure. It is important to understand the file structure to locate the main folders where the system is stored. You will need this information to update or remove the software.

Configuration Files (etc/openser)

openser-1:/etc/openser# ls -l

```
total 12
-rw-r--r-- 1 root root 1804 2007-09-10 14:02 dictionary.radius
-rw-r--r-- 1 root root 4077 2007-09-10 14:05 openser.cfg
-rw-r--r-- 1 root root 1203 2007-09-10 14:02 openserctlrccd
```

Modules (/lib/openser/modules)

openser-1:/lib/openser/modules# ls

```
acc.so            domain.so        msilo.so        sms.so
alias_db.so       enum.so          mysql.so        speeddial.so
auth_db.so        exec.so          nathelper.so    sst.so
auth_diameter.so  flatstore.so     options.so      statistics.so
auth_radius.so    gflags.so        path.so         textops.so
auth.so           group_radius.so  pdt.so          tm.so
avpops.so         group.so         permissions.so  uac_redirect.so
avp_radius.so     imc.so           pike.so         uac.so
dbtext.so         lcr.so           registrar.so    uri_db.so
dialog.so         mangler.so       rr.so           uri.so
dispatcher.so     maxfwd.so        seas.so         usrloc.so
diversion.so      mediaproxy.so    siptrace.so     xlog.so
domainpolicy.so   mi_fifo.so       sl.socd /lib/openser/modules
```

Binaries (/sbin)

openser-1:/sbin# ls -l op*

```
-rwxr-xr-x 1 root root 2172235 2007-09-10 14:02 openser
-rwxr-xr-x 1 root root   41862 2007-09-10 14:02 openserctl
-rwxr-xr-x 1 root root   38107 2007-09-10 14:02 openser_mysql.sh
-rwxr-xr-x 1 root root   13562 2007-09-10 14:02 openserunixcd /sbin
```

Log Files

The initialization log can be seen at syslog (/var/log/syslog):

```
Sep 10 14:25:56 openser-1 openser: init_tcp: using epoll_lt as the io watch
method (auto detected)
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: statistics manager
successfully initialized
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: StateLess module - initializing
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: TM - initializing...
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: Maxfwd module- initializing
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO:ul_init_locks: locks array
```

```
size 512
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: TextOPS - initializing
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init: SO_RCVBUF is
initially 109568
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init: SO_RCVBUF is
finally 262142
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init: SO_RCVBUF is
initially 109568
Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init: SO_RCVBUF is
finally 262142
Sep 10 14:25:56 openser-1 /sbin/openser[7792]: INFO:mi_fifo:mi_child_
init(1): extra fifo listener processes created
```

Startup Options

OpenSER can be started using the init scripts or using the openserctl utility. If you start openser using init scripts, you can only stop using init scripts. The same is valid if you start using openserctl utility.

Starting, stopping, and restarting OpenSER using the init scripts:

/etc/init/d/openser start|stop|restart

Starting, stopping, and restarting OpenSER using

/etc/init/d/openserctl start|stop|restart

The OpenSER executable has several startup options. These options, show, below, allow you to change the configuration of the DAEMON. Some of the most useful are:

- "-c" to check the configuration file
- "-D –E dddddd" to check module loading (don't use for production, it binds only the first interface)

There are lots of others to allow you to fine tune your configuration. For each option there is a correspondent core parameter that you can put in the configuration file.

```
Usage: openser -l address [-p port] [-l address [-p port]...] [options]
Options:
    -f file      Configuration file (default //etc/openser/openser.cfg)
    -c           Check configuration file for errors
    -C           Similar to '-c' but in addition checks the flags of
                  exported functions from included route blocks
    -l address   Listen on the specified address/interface (multiple -l
```

```
                mean listening on more addresses).  The address format
                is [proto:]addr[:port], where proto=udp|tcp and
                addr= host|ip_address|interface_name. E.g: -l locahost,
                -l udp:127.0.0.1:5080, -l eth0:5062 The default
                behavior is to listen on all the interfaces.
-n processes Number of child processes to fork per interface
                (default: 8)
-r           Use dns to check if is necessary to add a "received="
                field to a via
-R           Same as '-r' but use reverse dns;
                (to use both use '-rR')
-v           Turn on "via:" host checking when forwarding replies
-d           Debugging mode (multiple -d increase the level)
-D           Do not fork into daemon mode
-E           Log to stderr
-T           Disable tcp
-N processes Number of tcp child processes (default: equal to '-n')
-W method    poll method
-V           Version number
-h           This help message
-b nr        Maximum receive buffer size which will not be exceeded
                by auto-probing procedure even if  OS allows
-m nr        Size of shared memory allocated in Megabytes
-w dir       Change the working directory to "dir" (default "/")
-t dir       Chroot to "dir"
-u uid       Change uid
-g gid       Change gid
-P file      Create a pid file
-G file      Create a pgid file
-x socket    Create a unix domain socket
```

Summary

In this chapter you have learned how to install and prepare Linux for the OpenSER installation. We have downloaded and compiled OpenSER and MySQL modules. After the installation we included the OpenSER init file to start OpenSER at boot time.

4

OpenSER Standard Configuration

The OpenSER standard configuration file is installed at `/etc/openser/openser.cfg`. It is one of the simplest configuration files for OpenSER. It is the ideal script to start explaining the functioning of OpenSER. There are several sections that you should be familiar with, along with basic modules, parameters, and functions.

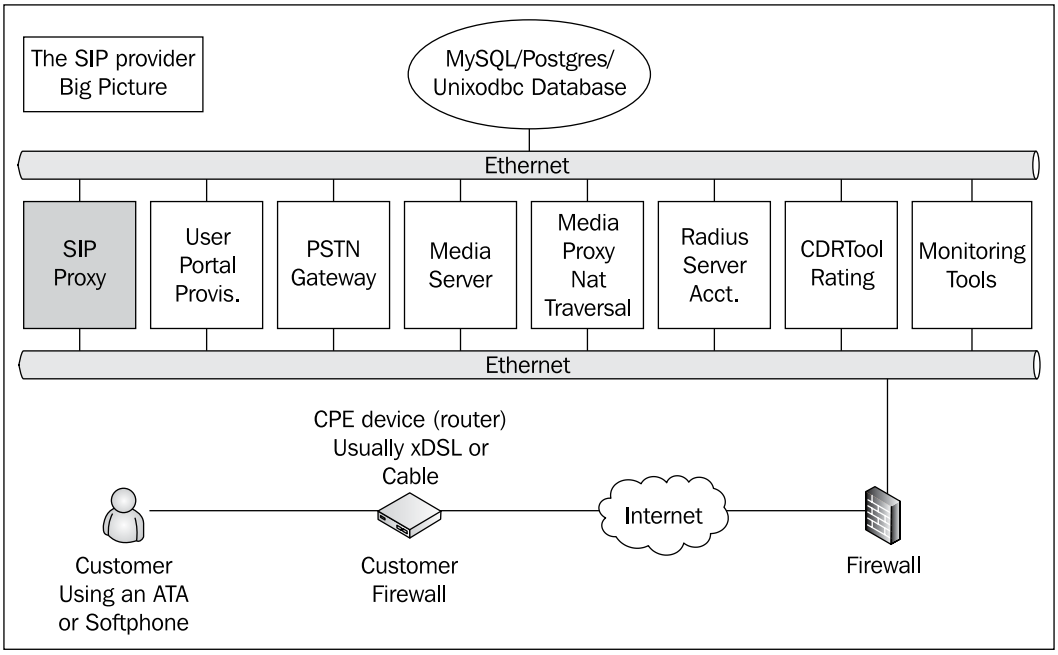
By the end of this chapter you will be able to:

- Identify the sections of the `openser.cfg` configuration file
- Identify the limitations of the standard configuration
- Use the `ngrep` utility to track SIP transactions
- Use the XLOG module to log the route processing
- Use the `append_hf` command to mark packets tracked in the `ngrep` utility

The standard configuration is a good starting point. It has a minimal functionality, it does not support authentication, so you can connect your SIP phones without a password. Anyway, you can call from one phone to another and we will test it later.

Where Are We?

Again, the solution for a VoIP provider has many components. To avoid loosing perspective, we will show this picture in most chapters. In this chapter, we are still working with the SIP proxy component in its standard configuration.



Analyzing the Standard Configuration

Below is shown the standard configuration of OpenSER version 1.2.2. In this section we will start to describe each line of the standard configuration with its commands and functions:

```
#
# $Id: openser.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $
#
#simple quick-start config script
#Please refer to the Core CookBook at http://www.openser.org/dokuwiki/
doku.php
#for a explanation of possible statements, functions and parameters.
#
# ----------- global configuration parameters ------------------------
debug=3            # debug level (cmd line: -ddddddddddd)
fork=yes
log_stderror=no    # (cmd line: -E)
```

```
children=4
port=5060
#uncomment the following lines for TLS support
#disable_tls = 0
#listen = tls:your_IP:5061
#tls_verify_server = 1
#tls_verify_client = 1
#tls_require_client_certificate = 0
#tls_method = TLSv1
#tls_certificate = "//etc/openser/tls/user/user-cert.pem"
#tls_private_key = "//etc/openser/tls/user/user-privkey.pem"
#tls_ca_list = "//etc/openser/tls/user/user-calist.pem"
# ------------------ module loading ----------------------------------
#set module path
mpath="/lib/openser/modules/"
#Uncomment this if you want to use SQL database
#loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
# Uncomment this if you want digest authentication
# mysql.so must be loaded !
#loadmodule "auth.so"
#loadmodule "auth_db.so"
# ----------------- setting module-specific parameters ---------------
# -- mi_fifo params --
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
# -- usrloc params --
modparam("usrloc", "db_mode", 0)
# Uncomment this if you want to use SQL database
# for persistent storage and comment the previous line
#modparam("usrloc", "db_mode", 2)
# -- auth params --
# Uncomment if you are using auth module
#
#modparam("auth_db", "calculate_ha1", yes)
```

```
#
# If you set "calculate_ha1" parameter to yes (which true in this
config),
# uncomment also the following parameter)
#
#modparam("auth_db", "password_column", "password")
# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)
# -------------------------  request routing logic -------------------
# main routing logic
route{
        # initial sanity checks -- messages with
        # max_forwards==0, or excessively long requests
        if (!mf_process_maxfwd_header("10")) {
                sl_send_reply("483","Too Many Hops");
                exit;
        };
        if (msg:len >=  2048 ) {
                sl_send_reply("513", "Message too big");
                exit;
        };
        # we record-route all messages -- to make sure that
        # subsequent messages will go through our proxy; that's
        # particularly good if upstream and downstream entities
        # use different transport protocol
        if (!method=="REGISTER")
                record_route();
        # subsequent messages withing a dialog should take the
        # path determined by record-routing
        if (loose_route()) {
                # mark routing logic in request
                append_hf("P-hint: rr-enforced\r\n");
                route(1);
        };
        if (!uri==myself) {
                # mark routing logic in request
                append_hf("P-hint: outbound\r\n");
                # if you have some interdomain connections via TLS
                #if(uri=~"@tls_domain1.net") {
                #       t_relay("tls:domain1.net");
                #       exit;
```

```
                #} else if(uri=~"@tls_domain2.net") {
                #       t_relay("tls:domain2.net");
                #       exit;
                #}
                route(1);
        };
        # if the request is for other domain use UsrLoc
        # (in case, it does not work, use the following command
        # with proper names and addresses in it)
        if (uri==myself) {
                if (method=="REGISTER") {
                        # Uncomment this if you want to use digest
                          authentication
                        #if (!www_authorize("openser.org",
                             "subscriber")) {
                        #       www_challenge("openser.org", "0");
                        #       exit;
                        #};
                        save("location");
                        exit;
                };
                lookup("aliases");
                if (!uri==myself) {
                        append_hf("P-hint: outbound alias\r\n");
                        route(1);
                };
                # native SIP destinations are handled using our
                  USRLOC DB
                if (!lookup("location")) {
                        sl_send_reply("404", "Not Found");
                        exit;
                };
                append_hf("P-hint: usrloc applied\r\n");
        };
        route(1);
}
route[1] {
        # send it out now; use stateful forwarding as it works
        # reliably even for UDP2TCP
        if (!t_relay()) {
                sl_reply_error();
        };
        exit;
}
```

This standard configuration is the simplest working configuration. We will start with it and then progressively include new commands and functions in the following chapters. With this configuration clients can register (without authentication) and UACs can communicate to each other. The Registrar, Location, and Proxy servers are working with a minimal configuration. Below we will explain some excerpts from the script:

```
debug=3            # debug level (cmd line: -ddddddddddd)
```

Set log level: this is number between -3 and 4. The default is 2. The higher the number, the more will be the information written to the log. With 4 the system's performance can become sluggish. The log levels are:

- L_ALERT (-3)—this level should be used to report only errors that require immediate action.
- L_CRIT (-2)—this level should be used to report only errors that cause a critical situation.
- L_ERR (-1)—this level should be used to report errors during data processing that do not cause system malfunctioning.
- L_WARN (1)—this level should be used to write warning messages.
- L_NOTICE (2)—this level should be used to report unusual situations.
- L_INFO (3)—this level should be used to write informational messages.
- L_DBG (4)—this level should be used to write messages for debugging.

```
fork=yes
```

The fork parameter defines if the OpenSER processes will run in background or foreground modes. To operate in background set `fork=yes`. Sometimes you will the find it useful to start it in the foreground to locate script errors.If fork is disabled, OpenSER will not be able to listen on more than one interface and TCP/TLS support will be automatically disabled. In a single process mode, only one UDP interface is accepted.

```
log_stderror=no    # (cmd line: -E)
```

If set to `yes`, the server will print its debugging information to standard error output. If set to `no`, syslog will be used.

```
children=4
```

The `children` core parameter informs OpenSER of how many *child* processes per interface to create the process incoming requests. Four processes seem to be a good starting point for most systems. This parameter only applies to UDP interfaces. It has no impact on TCP processes.

```
port=5060
```

This is the default port to be used if none is specified in the listen parameter.

```
mpath="/lib/openser/modules/"
```

Set the module search path. This can be used to simplify the loading of modules.

```
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
```

The lines above load OpenSER external modules. At this time, only the minimum required modules are loaded. Additional functionality will need other modules such as RADIUS and MYSQL to be loaded. All modules have a README file describing their functions.

```
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
```

The name of the FIFO file to be created for listening and reading external commands.

```
modparam("usrloc", "db_mode", 0)
```

The `modparam` core command configures the corresponding module. The `usrloc` module above is responsible for the location service. When a client registers, it saves the location information, also known as AOR (Address of Record) to the location indicated by the `db_mode` parameter. In this case 0, means memory. So if you turn off your server, you will loose all your register records. The location of this table depends on the value of the `db_mode` parameter. A `db_mode` set to `0` indicates that this data will not be saved into a database. In other words, if OpenSER is turned off, all the records are lost.

```
modparam("rr", "enable_full_lr", 1)
```

The statement above sets the `enable_full_lr` parameter of the module `rr` (Record Routing) to `1`. It tells OpenSER to be fully compliant with older SIP clients that do not manage `record_route` header fields. If set to `1` then `;lr=on` instead of just `;lr` will be used.

```
route {
```

This is the beginning of the routing logic for a SIP request. The block starts with a {. In this block the SIP requests will be processed. An overview can be seen below:

```
# initial sanity checks -- messages with
# max_forwards==0, or excessively long requests
if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
};
if (msg:len >=  2048 ) {
        sl_send_reply("513", "Message too big");
        exit;
};
```

When a request gets into the main routing block some checks are done.

The first check is about the maximum number of forwards. To avoid loops, we use the function `mf_process_maxfwd_header()` to check how many SIP hops the packet has passed. If a loop is found, the script sends a message "483 Too many hops" using the function `sl_send_reply()`.

The `msg:len` is a function of the OpenSER core that returns the length in bytes of the SIP request. This is a standard check to impose some limits on the message size.

```
if (!method=="REGISTER")
        record_route();
```

If the method is different from `REGISTER`, OpenSER will record-route. This instruction tells the SIP server to stay in the path of SIP requests between two UACs. The `record_route()` function simply adds a new record-route header field.

```
# Subsequent messages within a dialog should take the
        # Path determined by record-routing
        if (loose_route()) {
                # mark routing logic in request
                append_hf("P-hint: rr-enforced\r\n");
                route(1);
        };
```

The `loose_route()` function tests to see if the request will be routed using the record-route header fields. Requests identified by this function will be routed using the content of the top record-route header field.

If the request is from the same dialog, we will get into the `if` clause and forward the packet. Then we should simply forward the request. We do this by calling the `route(1)` secondary routing block were the `t_relay()` function will be invoked.

A new function called `append_hf` will add a header field with a hint that the request was processed according to the record-route header field (rr-enforced).

```
if (!uri==myself) {
        # mark routing logic in request
        append_hf("P-hint: outbound\r\n");
        # if you have some interdomain connections via TLS
        #if(uri=~"@tls_domain1.net") {
        #       t_relay("tls:domain1.net");
        #       exit;
        #} else if(uri=~"@tls_domain2.net") {
        #       t_relay("tls:domain2.net");
        #       exit;
        #}
        route(1);
};
```

The code above will treat the requests for a domain not served by our proxy, `if(!uri==myself)`, forwarding the request by calling `route(1)` where the `t_relay` will be invoked. This proxy by default is working as an open relay. In the following chapters we will discuss how to improve the handling of outbound calls. It is important to forward requests to other proxies; however, some identity checks should be in place. Now, we will treat the requests directed to the domains handled by our SIP proxy.

```
if (uri==myself) {
        if (method=="REGISTER") {
                # Uncomment this if you want to use digest
                 authentication
                #if (!www_authorize("openser.org",
                 "subscriber")) {
                #       www_challenge("openser.org", "0");
                #       exit;
                #};
                save("location");
                exit;
        };
```

If the request method is `REGISTER`, save the AOR to the location table using the `save("location")`. It is important to understand two concepts. At this time authentication is disabled (`www_authorize` commented) and the location database is not persistent because we don't have a database installed with the SIP proxy.

```
lookup("aliases");
if (!uri==myself) {
        append_hf("P-hint: outbound alias\r\n");
        route(1);
};
```

Aliases are alternative URIs (that is, `8590@voffice.com.br` can be an alias for the original URI `flavio@voffice.com.br`). The `lookup("aliases")` function simply seeks the canonical URI for the URI presented in the request. If the URI is found, it replaces the R-URI before to proceed. The resulting URI can be located inside or outside our domain. If it is outside, the system simply forwards the packet to the SIP proxy responsible for the domain. If it is outside it proceeds the request processing.

```
if (!lookup("location")) {
        sl_send_reply("404", "Not Found");
        exit;
};
append_hf("P-hint: usrloc applied\r\n");
```

The `lookup("location")` function will try to recover the (AOR) of the R-URI. If the AOR is located (the UA is registered) it will change the R-URI by the ip-address of the UA. If the AOR is not found we will simply send back an error message `("404 Not Found")`.

```
route(1);
```

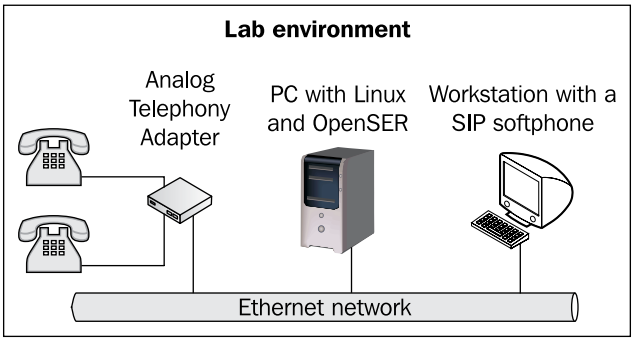
If the AOR is found we will end up with the `route(1);`

```
route[1] {
        # send it out now; use stateful forwarding as it works
reliably
        # even for UDP2TCP
        if (!t_relay()) {
                sl_reply_error();
        };
        exit;
}
```

Finally, the routing block is invoked. The `t_relay()` function forwards the request statefully based on the Request URI. The domain part is resolved using DNS helpers such as NAPTR, SRV, and A records. This function is exposed by the TRANSACTION module (`tm.so`) and is responsible for sending the requests and handling any resends and responses. If the request could not be sent to the destination successfully, an error message will be generated automatically by the `t_relay()` function. The function `sl_reply_error()` will send a reply back to the UA if a failure occurs.

Using the Standard Configuration

In this lab, we will use a protocol analyzer to capture a complete SIP call. We will analyze the headers and the message flow. You can create this environment with a PC and two UACs. The UACs can be Softphones, ATAs, or even IP phones.



Adapt this lab to your needs.

1. Start capturing packets using `ngrep`. If it is not installed, install it using:
   ```
   apt-get install ngrep
   ```
2. To capture the packets use:
   ```
   ngrep -p -q -W byline port 5060 >test.txt
   ```
3. Configure the UACs (Softphones, IP phones, or ATAs).

Configure the first UAC with the following configuration:

sip proxy 10.1.x.y – IP of your proxy

user: 1000

password: 1000

Configure the second UAC to the following configuration:

sip proxy 10.1.x.y – IP of your proxy

user: 1001

password 1001

After configuring the devices, you will need to register the IP Phone. Not all devices do auto registering.

1. Check if the phone is registered using:

   ```
   openserctl ul show
   ```

2. At the first UAC dial 1001. The second UAC will ring.
3. Verify this capture does not exhibit the "407 - Proxy authentication required" error for the INVITE request and the "401- Unauthorized" error for the REGISTER requests. This proves that an authentication is not being requested.
7. You can see the capture by issuing the command:

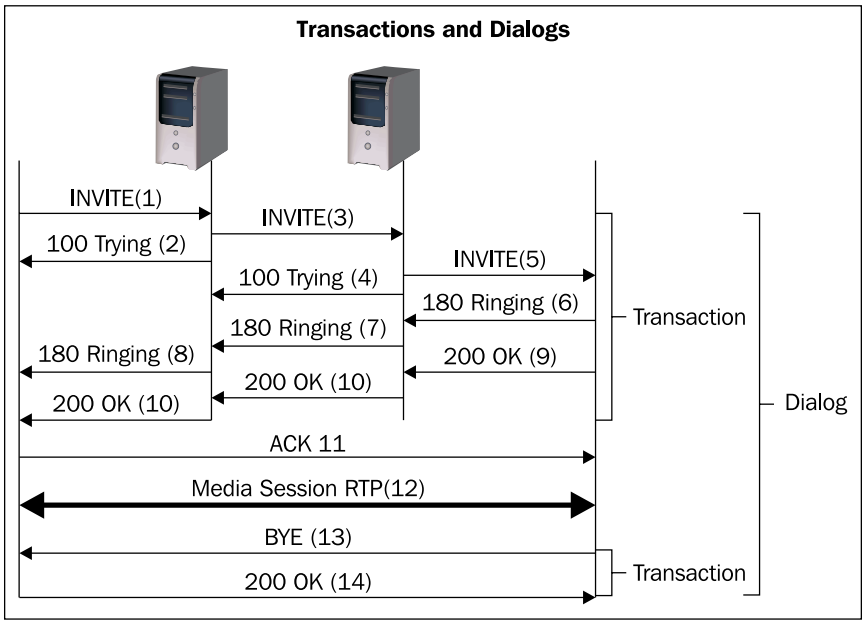
   ```
   more test.txt
   ```

Routing Basics

It is not easy to figure out how to route SIP packets. We will present in this section some of the basic concepts to route SIP packets over the proxy server. The first important concept is the one involving transactions and dialogs.

Transactions and Dialogs

A transaction starts with a request and finishes, usually, with a response code. The branch parameter in the VIA header field identifies a transaction. A dialog may start with an INVITE transaction and finish with a BYE transaction. A dialog is identified by the combination of the FROM, TO, and CALL-ID header fields. Not all SIP methods start a dialog, the REGISTER and MESSAGE methods do not.



Initial and Sequential Requests

It is important to understand the difference between initial requests and sequential requests. For initial requests, you have to decide how to route using a discovery mechanism, usually based on DNS or in a location table.

The initial request records the routing information using the **VIA** header and if you have record-routing enabled, in the **ROUTE** headers too. Inside a transaction, the packets are routed back using the **VIA** header field returning to every proxy passed before. Subsequent requests are routed using the **CONTACT** header field. However, if you had turned on record-routing, the subsequent requests will be routed back using the route set discovered.

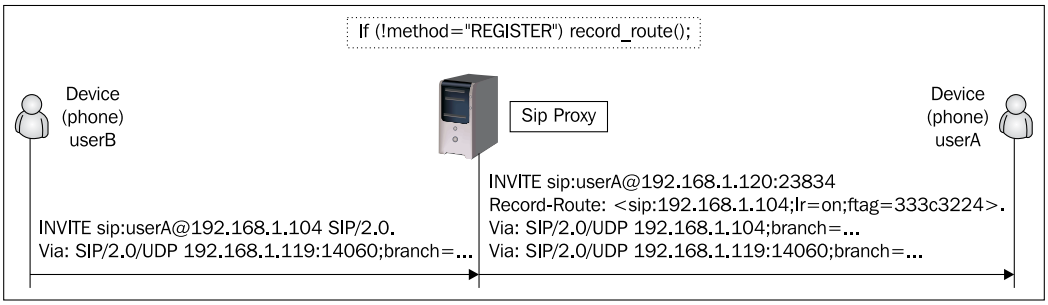
You can distinguish between initial and sequential requests using the **TAG** parameter in the **TO** header field.

Routing in a Context of a Transaction

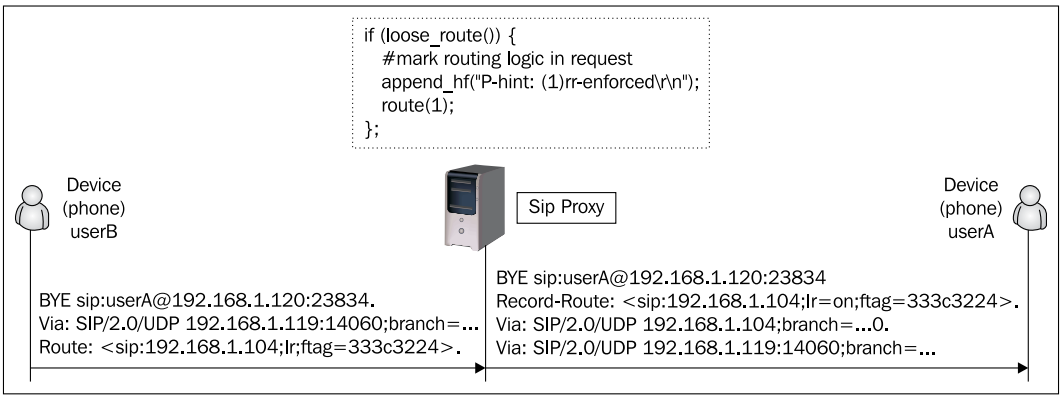
Inside a transaction all requests are routed using the **VIA** header field. So, all responses will go through the proxy before getting to the final destination. If you route the request using the function `t_relay()`, the SIP proxy operates in the stateful mode, so you can handle responses and failures using the sections `onreply_route[]` and `failure_route[]`.

Routing in the Context of a Dialog

Subsequent requests in the same dialog are routed directly peer-to-peer using the CONTACT header field. Most times you will want to force the subsequent transactions, such as BYE, through the proxy for billing and dialog control. You can do this by enabling a resource called record-routing. Doing this will instruct the script to record-routes.

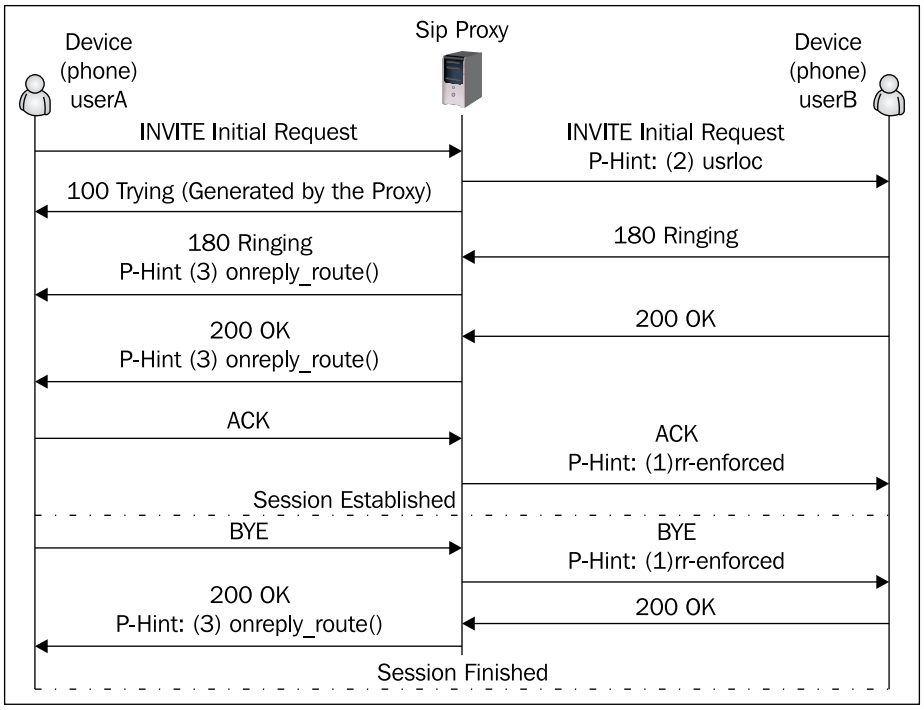


Later, you may use the pre-recorded routes, also known as route set, to forward the subsequent requests. This is the most common configuration and it is in the default configuration file.



Lab—Tracking a Complete Dialog

In this lab, we will use a simplified script to understand routing concepts. We will use the function `append_hf()` to append a header field to the packet marking the point in the script where the packet was processed.



Step 1: Use the script below (openser.chapter4-2). Restart OpenSER and the re-register the phones.

```
route{
    # All messages, except for REGISTER will pass here
    if (!method=="REGISTER") record_route();

    # subsequent messages withing a dialog should take the
    # path determined by record-routing
    if (loose_route()) {
           # mark routing logic in request
           append_hf("P-hint: (1)rr-enforced\r\n");
           route(1);
    };

    # We will route only intra-domain requests
    if (!uri==myself) {
           exit();
    };

    # main routing of intra-domain requests
    if (uri==myself) {
```

```
            if (method=="REGISTER") {
                  save("location");
                  exit;
            };

            # native SIP destinations are handled using our USRLOC DB
            if (!lookup("location")) {
                  sl_send_reply("404", "Not Found");
                  exit;
            };
            append_hf("P-hint: (2)usrloc applied\r\n");
      };
      route(1);
   }

   route[1] {
      # send it out now; use stateful forwarding
      t_on_reply("1");
      t_on_failure("1");
          if (!t_relay()) {
             sl_reply_error();
      };
      exit;
   }

   onreply_route[1] {
      append_hf("P-hint: (3)passed thru onreply_route[1]\r\n");
   }

   failure_route[1] {
      append_hf("P-hint: (4)passed thru failure_route[1]\r\n");
   }
```

Step 2: Use ngrep to capture the requests and responses to a file

```
ngrep -p -q -W byline port 5060 >rr-stateful
```

Step 3: Start a call from 1000 to 1001 (or any other phone registered)

Step 4: Stop ngrep using CTRL-C

Check the packets using a text editor and see the P-Hint header fields. They are supposed to be equal to the figure printed in the beginning of the Lab.

Lab—Running Stateless

If you replace the function `t_relay()` by the function `forward()` you will use the SIP proxy in the stateless mode. Everything will work, but you won't be able to process replies. The Proxy now does not correlate requests and responses in the same transactions. The responses are processed using the VIA header field as usual.

Step 1: Replace the `t_relay()` function by the `forward()` function.

Replace:

```
if (!t_relay()) {
          sl_reply_error();
};
```

By:

```
forward()
```

Step 2: Restart OpenSER and re-register the phones

Step 3: Use `ngrep` to capture the requests and responses to a file

```
ngrep -p -q -W byline port 5060 >rr-stateless
```

Step 4: Make a call from 1000 to 1001

Step 5: After ending the call, stop `ngrep`.

Step 6: Use a text editor to check the the file named rr-stateless. You will notice that responses now does not have the P-Hint header field. This indicates that they were not being processed in the `onreply_route` section. So, if you use stateless processing, you cannot do anything with the replies, unless forward them to the destination.

Lab—Disabling record-route

In this lab we will stop recording the routes. The subsequent requests in the dialog will go directly from one phone to another bypassing the SIP proxy. They use the information in the **CONTACT** header field to do so.

Step 1: Comment the line responsible for record routing

```
#if (!method=="REGISTER") record_route();
```

Step 2: Restart OpenSER and re-register the phones

Step 3: Use `ngrep` to capture the requests and responses to a file

```
ngrep -p -q -W byline port 5060 >norr-stateless
```

Step 4: Make a call from 1000 to 1001

Step 5: After ending the call, stop `ngrep`.

Step 6: Use a text editor to check the file norr-stateless. You will notice that now, you can't see the BYE and ACK requests. This happens because now, they are going directly from one peer to another. If you want to bill the calls, that's the behavior you simply don't want to have from your SIP proxy.

Summary

In this chapter you have learned some of the statements for each of the sections of the `openser.cfg` file. This is the simplest configuration file. In the next chapters we will increase the functionality and the complexity of the script. This chapter served as a starting point to develop more advanced scripts. Even though it is simple the script allows you to connect two phones and dial to each other.

5
Adding Authentication with MySQL

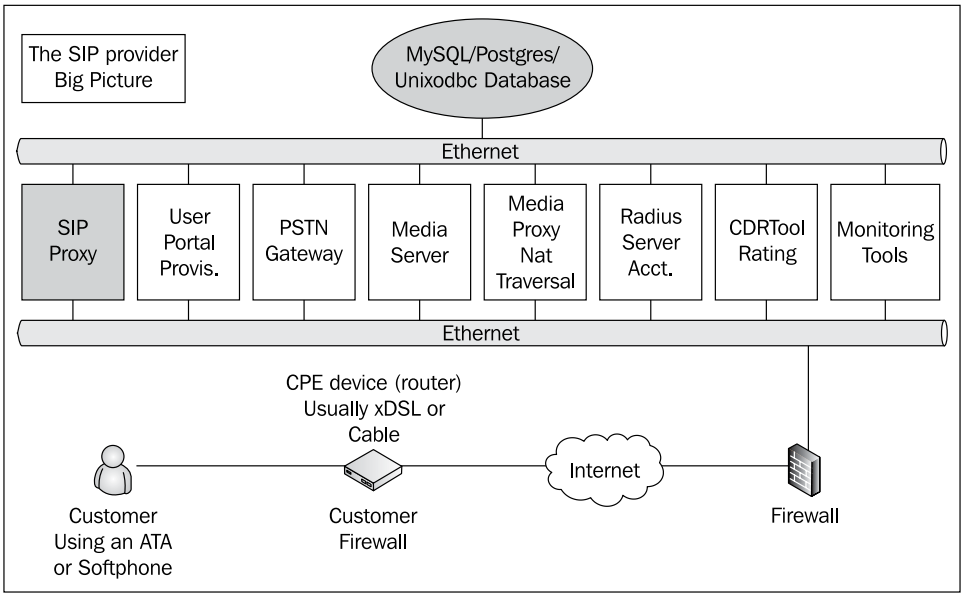
In this chapter we will learn how to use several database back-ends to authenticate SIP requests and provide persistence of data such as location and alias tables. Primarily, we will do everything with MySQL. This chapter is divided in two parts. In the first one we will learn how to implement authentication and in the second one we will learn how to deal with calls in each direction.

By the end of this chapter you will be able to:

- Configure MySQL to authenticate SIP devices
- Use the `openserctl` utility for basic operations such as adding and removing users
- Change the `openser.cfg` script to configure MySQL authentication
- Implement persistence for the subscriber table
- Implement persistence for the location tables
- Restart the server without loosing the location records
- Deal correctly with inbound-to-inbound, inbound-to-outbound, outbound-to-inbound, and outbound-to-outbound sessions.
- Deal correctly with CANCEL Requests.

Where Are We?

Now, we are still focusing on the SIP proxy. However, we are going to include a new component, the Database. OpenSER can use MySQL and PostgreSQL. For this book, we have chosen to work with MySQL. It is, by far, the most used database for OpenSER.



The AUTH_DB Module

Database-based authentication is performed by the module `AUTH_DB`. Other types of authentication such as radius and diameter can be performed using `AUTH_RADIUS` and `AUTH_DIAMETER` respectively. `AUTH_DB` works together with database modules such as MySQL and PostgreSQL. `AUTH_DB` has some parameters that are not explicitly declared in the script. Let's see the default parameters for the `AUTH_DB` module:

Parameter	Default	Description
db_url	"mysql://openserro:openserro@localhost/openser"	URL of the database
user_column	"username"	Name of the column holding domains of users
domain_column	"domain"	Name of the column holding domains of users
password column	"ha1"	Name of the column holding passwords

Parameter	Default	Description
password_column2	"ha1b"	Name of the column holding pre-calculated HA1 strings that were calculated including the domain in the username.
calculate_ha1	0 (server assumes that ha1 strings are already calculated in the database)	Tell the server whether it should expect plaintext passwords in the database or not.
use_domain	0 (domains won't be checked when looking up in the subscriber database)	Use this parameter set to 1 if you have a multi-domain environment.
load_credentials	"rpid"	Specifies the credentials to be fetch from the database when the authentication is performed. The loaded credentials will be stored in AVPs.

The `AUTH_DB` module exports two functions.

```
www_authorize(realm, table)
```

This function is used in the `REGISTER` authentication that occurs in accordance with RFC2617.

```
proxy_authorize(realm, table)
```

The function verifies credentials according to RFC2617 for the non-REGISTER requests. If the credentials are verified successfully, the credentials will be marked as authorized.

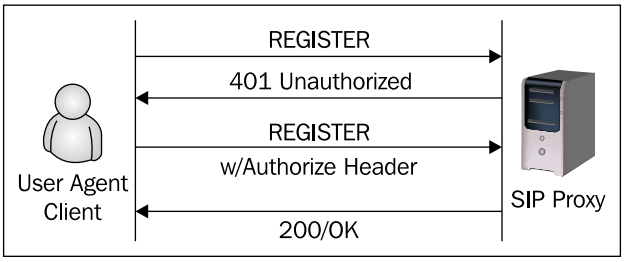
You have to use `www_authorize` when your server is the endpoint of the request. Use `proxy_authorize` when the request's final destination is not your server and you will forward the request ahead, actually working as a proxy.

The difference between `www_authorize` and `proxy_authorize` is that if the request's end point is you (`REGISTER`) you use `www_authorize`.

The REGISTER Authentication Sequence

The script should authenticate `REGISTER` and `INVITE` messages. Let's show how this happens before changing the `openser.cfg` script. When OpenSER receives the `REGISTER` message it checks for the existence of the Authorize header. If it does not find one, it will challenge UAC for the credentials and exit.

After being challenged the UAC should send a `REGISTER` message with an Authorize header field.



Register Sequence (Packets Captured by ngrep)

The register process can be seen in the packet capture shown below:

U 192.168.1.119:29040 -> 192.168.1.155:5060
REGISTER sip:192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-13517a5a8218ff45-1--d87543-
;rport.
Max-Forwards: 70.
Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>.
To: "1000"<sip:1000@192.168.1.155>.
From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.
Call-ID:
e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVlZTY0NDc..
CSeq: 1 REGISTER.
WWW-Authenticate: Digest realm="192.168.1.155", nonce="46263864b3abb96a423a7ccf052fa68
d4ad5192f".
Server: OpenSER (1.2.0-notls (i386/linux)).
Content-Length: 0.

U 192.168.1.119:29040 -> 192.168.1.155:5060
REGISTER sip:192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-da776d09bd6fcb65-1--d87543-
;rport.
Max-Forwards: 70.
Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>.
To: "1000"<sip:1000@192.168.1.155>.
From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.
Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVlZTY0NDc..
CSeq: 2 REGISTER.
Expires: 3600.

Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE, INFO.
User-Agent: X-Lite release 1003l stamp 30942.
Content-Length: 0.

U 192.168.1.155:5060 -> 192.168.1.119:29040
SIP/2.0 401 Unauthorized.
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-13517a5a8218ff45-1--d87543-;rport=29040.
To: "1000"<sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.41bb.
From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.
Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVlZTY0NDc..
CSeq: 1 REGISTER.
WWW-Authenticate: Digest realm="192.168.1.155", nonce="46263864b3abb96a423a7ccf052fa68d4ad5192f".
Server: OpenSER (1.2.0-notls (i386/linux)).
Content-Length: 0.

U 192.168.1.119:29040 -> 192.168.1.155:5060
REGISTER sip:192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-da776d09bd6fcb65-1--d87543-;rport.
Max-Forwards: 70.
Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>.
To: "1000"<sip:1000@192.168.1.155>.
From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.
Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVlZTY0NDc..
CSeq: 2 REGISTER.
Expires: 3600.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE, INFO.
User-Agent: X-Lite release 1003l stamp 30942.
Authorization: Digest username="1000",realm="192.168.1.155",nonce="46263864b3abb96a423a7ccf052fa68d4ad5192f",uri="sip:192.168.1.155",response="d7b33793a123a69ec12c8fc87abd4c03",algorithm=MD5.
Content-Length: 0.

U 192.168.1.155:5060 -> 192.168.1.119:29040
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-da776d09bd6fcb65-1--d87543-;rport=29040.
To: "1000"<sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.c577.
From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.

Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVlZTY0NDc..
CSeq: 2 REGISTER.
Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>;expires=3600.
Server: OpenSER (1.2.0-notls (i386/linux)).
Content-Length: 0.

Register Sequence Code Snippet

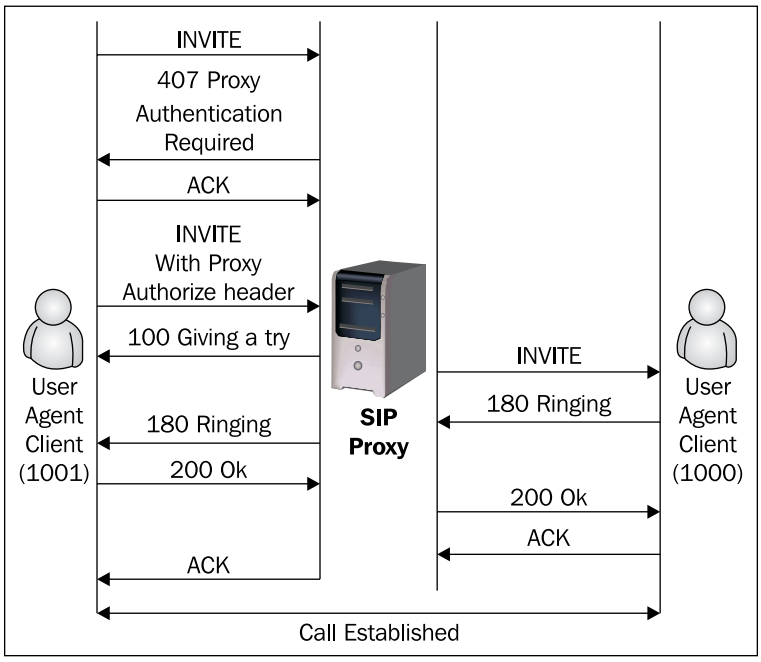
Let's show now how this sequence is coded in the `openser.cfg` script:

```
if (method=="REGISTER") {
      # Uncomment this if you want to use digest authentication
       if (!www_authorize("", "subscriber")) {
           www_challenge("", "0");
           exit;
       };
       save("location");
       exit;
};
```

In the sequence above, in the first pass the `REGISTER` packet is not authenticated by the `www_authorize` function. Then the instruction `www_challenge` is invoked. It sends the **"401 Unauthorized"** packet, which contains the authentication challenge, according to the digest authentication scheme. In the second pass the UAC sends the `REGISTER` packet with the correct Authorize header field, then the `save ("location")` function is invoked to save the AOR in the MySQL location table.

The INVITE Authentication Sequence

Opposite is the `INVITE` authentication sequence of an ordinary call. The proxy server always answers the first `INVITE` with a reply containing a message, "407 Proxy Authentication Required". This message has the "Authorize" header field, containing information about the digest authentication, such as realm and nonce. Once received by the UAC, this message is answered with a new `INVITE`. Now, the "Authorize" header field contains the digest calculated using the username, password, realm, and nonce using the MD5 algorithm. If there is a match between the digest passed in the request and the one calculated in the server using the same parameters, the user is authenticated.



INVITE Sequence Packet Capture

We have captured an `INVITE` authentication sequence using `ngrep`. This sequence will help you to understand the figure above. The SDP headers were striped off to avoid a long list.

U 192.168.1.169:5060 -> 192.168.1.155:5060
INVITE sip:1000@192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKf45d977e65cf40e0.
From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.
To: <sip:1000@192.168.1.155>.
Contact: <sip:1001@192.168.1.169>.
Supported: replaces.
Call-ID: 8acb7ed7fc07c369@192.168.1.169.
CSeq: 39392 INVITE.
User-Agent: TMS320V5000 TI50002.0.8.3.
Max-Forwards: 70.
Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE.
Content-Type: application/sdp.
Content-Length: 386.
(sdp header striped off).

U 192.168.1.155:5060 -> 192.168.1.169:5060
SIP/2.0 407 Proxy Authentication Required.
Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKf45d977e65cf40e0.

From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.
To: <sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.b550.
Call-ID: 8acb7ed7fc07c369@192.168.1.169.
CSeq: 39392 INVITE.
Proxy-Authenticate: Digest realm="192.168.1.155", nonce="4626420b4b162ef84a1a1d3966704d380194bb78".
Server: OpenSER (1.2.0-notls (i386/linux)).
Content-Length: 0.

U 192.168.1.169:5060 -> 192.168.1.155:5060
ACK sip:1000@192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKf45d977e65cf40e0.
From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.
To: <sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.b550.
Contact: <sip:1001@192.168.1.169>.
Call-ID: 8acb7ed7fc07c369@192.168.1.169.
CSeq: 39392 ACK.
User-Agent: TMS320V5000 TI50002.0.8.3.
Max-Forwards: 70.
Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE.
Content-Length: 0.

U 192.168.1.169:5060 -> 192.168.1.155:5060
INVITE sip:1000@192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKcdb4add5db72d493.
From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.
To: <sip:1000@192.168.1.155>.
Contact: <sip:1001@192.168.1.169>.
Supported: replaces.
Proxy-Authorization: Digest username="1001", realm="192.168.1.155", algorithm=MD5, uri="sip:1000@192.168.1.155", nonce="4626420b4b162ef84a1a1d3966704d380194bb78", response="06736c6d7631858bb1cbb0c86fb939d9".
Call-ID: 8acb7ed7fc07c369@192.168.1.169.
CSeq: 39393 INVITE.
User-Agent: TMS320V5000 TI50002.0.8.3.
Max-Forwards: 70.
Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE.
Content-Type: application/sdp.
Content-Length: 386.
(sdp header striped off)

INVITE Code Snippet

In the code below, the SIP proxy will challenge the user for credentials on any request different from REGISTER. We consume the credentials after authentication, for security reasons, to avoid sending encrypted material ahead.

```
        if (!proxy_authorize("","subscriber")) {
            proxy_challenge("","0");
```

```
            exit;
        };
        consume_credentials();

        lookup("aliases");
        if (!uri==myself) {
            append_hf("P-hint: outbound alias\r\n");
            route(1);
        };
        # native SIP destinations are handled using
          our USRLOC DB
        if (!lookup("location")) {
            sl_send_reply("404", "Not Found");
            exit;
        };
        append_hf("P-hint: usrloc applied\r\n");
};
route(1);
```

Digest Authentication

The digest authentication is based on the RFC2617 "HTTP Basic and Digest Access Authentication". Our objective in this chapter is to show the basics of a system with digest authentication. It is not an answer to all possible security problems with SIP, but it is certainly a good method to protect names and passwords traversing the network.

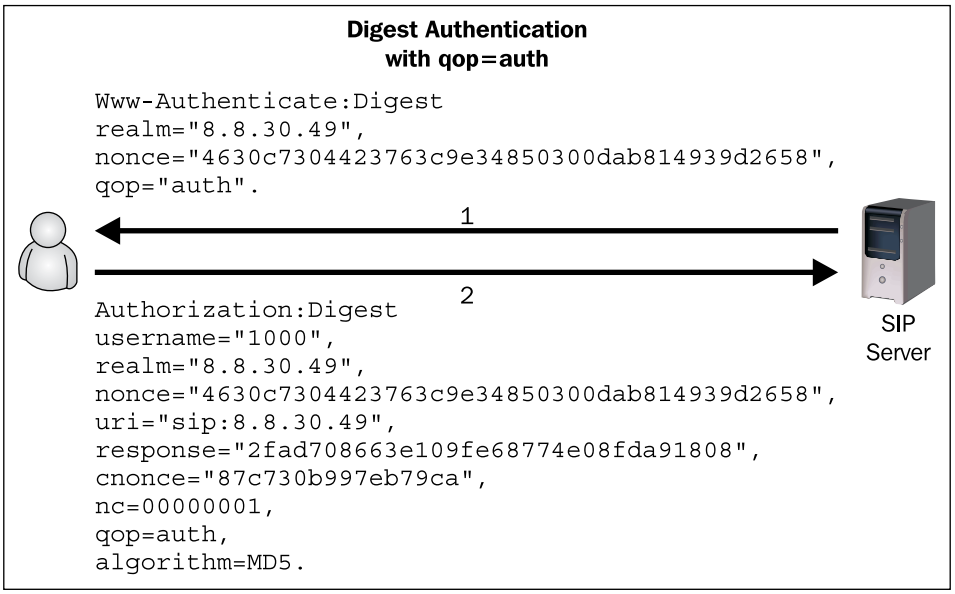
The digest scheme is a simple challenge-response mechanism. It challenges the UA using a **nonce** value. A valid response includes a checksum with all the parameters. Thus the password is never transmitted as simple text.

WWW-Authenticate Response Header

If a server receives a `REGISTER` or an `INVITE` request and a valid "Authorize" header field is not sent, the server replies "401 unauthorized" with a header field "WWW-Authenticate". This header contains a realm and a nonce.

The Authorization Request Header

The client is expected to try again, now passing the "Authorize" header field. It contains the user name, realm, and nonce (passed by the server), uri, a hexadecimal answer with 32 digits, and an algorithm method of authentication (in this case MD5). This answer is the checksum generated by the client using the specified algorithm.



QOP—Quality of Protection

The **qop** parameter indicates the quality of protection that the client has applied to the message. If present this value should be one of the alternatives that the server supports. These alternatives are indicated in the "WWW-Authenticate" header field. These values affect the digest computation. This directive is optional to preserve the compatibility with a minimum implementation of RFC2809.

You can configure the qop parameter on both function calls `www_challenge(realm,qop)` and `proxy_challenge(realm,qop)`. If configured to 1, the server will ask for the qop parameter. Always use `qop=1` (enabled), it will help you to avoid "replay" attacks. However, some clients can be incompatible with qop. A detailed description of the digest authentication can be found in RFC2617.

Installing MySQL Support

To allow persistence, in other words, keep the user credentials in a database, where they are protected from power outages and reboots, OpenSER will need to be configured to use a database such as MySQL. Before you proceed it is important to verify that you have MySQL installed and the OpenSER MySQL module compiled and installed.

In Chapter 3 we compiled OpenSER with MySQL support. Check the directory `/lib/openser/modules` for the `mysql.so` module.

Some additional tasks have to be done before you can use OpenSER with MySQL.

Step 1: Verify the existence of the module `mysql.so` in the directory:

```
ls /lib/openser/modules/mysql.so
```

If the module does not exist, please compile OpenSER with MySQL support.

Step 2: Create MySQL tables using the `openser_mysql.sh` shell script.

This script will create the MySQL tables with the following parameters:

```
DBNAME="openser"
DBHOST="localhost"
DBRWUSER="openser"
DBRWPW="openserrw"
DBROUSER="openserro"
DBROPW="openserro"
DBROOTUSER="root"
cd/sbin
./openser_mysql.sh create
MySQL password for root:
Enter password:
Enter password:
creating database openser ...
Core OpenSER tables succesfully created.
Install presence related tables ?(y/n):y
creating presence tables into openser ...
Presence tables succesfully created.
```

Install extra tables - imc,cpl,siptrace,domainpolicy ?(y/n):y
creating extra tables into openser ...
Extra tables succesfully created.
Install SERWEB related tables ?(y/n):n
Domain (realm) for the default user 'admin': voffice.com.br

A password will be solicited to access the database. The password is empty at this moment. The script will ask for the password twice; press *Enter* in both. The script will ask for a domain (realm); inform your domain for the admin user.

Step 3: Configure OpenSER to use MySQL.

Make the highlighted changes in the file as below.

```
# ------------------ module loading ----------------------------------
#set module path
mpath="//lib/openser/modules/"
# Uncomment this if you want to use SQL database
loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"
# ----------------- setting module-specific parameters ---------------
# -- mi_fifo params --
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
# -- usrloc params --
#modparam("usrloc", "db_mode",   0)
# Uncomment this if you want to use SQL database
# for persistent storage and comment the previous line
modparam("usrloc", "db_mode", 2)
# -- auth params --
# Uncomment if you are using auth module
#
```

```
modparam("auth_db", "calculate_ha1", yes)
#
# If you set "calculate_ha1" parameter to yes (which true in this
config),
# uncomment also the following parameter)
#
modparam("auth_db", "password_column", "password")

# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)

# -------------------------  request routing logic -------------------
# main routing logic

route{
        # initial sanity checks -- messages with
        # max_forwards==0, or excessively long requests
        if (!mf_process_maxfwd_header("10")) {
                sl_send_reply("483","Too Many Hops");
                exit;
        };
        if (msg:len >=  2048 ) {
                sl_send_reply("513", "Message too big");
                exit;
        };
        # we record-route all messages -- to make sure that
        # subsequent messages will go through our proxy; that's
        # particularly good if upstream and downstream entities
        # use different transport protocol
        if (!method=="REGISTER")
                record_route();
        # subsequent messages withing a dialog should take the
        # path determined by record-routing
        if (loose_route()) {
                # mark routing logic in request
                append_hf("P-hint: rr-enforced\r\n");
                route(1);
        };
        if (!uri==myself) {
                # mark routing logic in request
                append_hf("P-hint: outbound\r\n");
                # if you have some interdomain connections via TLS
                #if(uri=~"@tls_domain1.net") {
                #       t_relay("tls:domain1.net");
```

```
                #       exit;
                #} else if(uri=~"@tls_domain2.net") {
                #       t_relay("tls:domain2.net");
                #       exit;
                #}
                route(1);
        };
        # if the request is for other domain use UsrLoc
        # (in case, it does not work, use the following command
        # with proper names and addresses in it)
        if (uri==myself) {
                if (method=="REGISTER") {
                        # Uncomment this if you want to use digest.
                        if (!www_authorize("", "subscriber")) {
                                www_challenge("", "0");
                                exit;
                        };
                        save("location");
                        exit;
          };
                if (!proxy_authorize("","subscriber")) {
                        proxy_challenge("","0");
                        exit;
                };
                consume_credentials();
                lookup("aliases");
                if (!uri==myself) {
                        append_hf("P-hint: outbound alias\r\n");
                        route(1);
                };
            # native SIP destinations are handled using our USRLOC DB
                if (!lookup("location")) {
                        sl_send_reply("404", "Not Found");
                        exit;
                };
                append_hf("P-hint: usrloc applied\r\n");
        };
        route(1);
}

route[1] {
        # send it out now; use stateful forwarding as it works
```

```
reliably
        # even for UDP2TCP
        if (!t_relay()) {
                sl_reply_error();
        };
        exit;
}
```

openser.cfg File Analysis

Now the configuration is ready to authenticate `REGISTER` transactions. Now, we can save the AOR in the location database implementing persistence. This allows us to restart the server without loosing the AOR records and affecting the UACs.

Another important fact is that OpenSER is now authenticating `REGISTER` requests. Later we will implement authentication also for `INVITE` requests. Now it is required that UACs authenticate to register.

```
loadmodule "mysql.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
```

The MySQL support is added easily by including `mysql.so` in the list of loaded modules. MySQL should be loaded before the other modules. Some modules, such as `uri_db`, depend on MySQL to load.

The authentication capability is provided by the `auth.so` and `auth_db.so` modules. These modules are required to enable the authentication functionality. The module `uri_db` exposes some authentication.

```
modparam("auth_db", "calculate_ha1", 1)
modparam("usrloc", "db_mode", 2)
```

The parameter `calculate_ha1` tells the `auth_db` module to use plaintext passwords. We will use plaintext password for compatibility with SerMyAdmin.

The `db_mode` parameter tells the `usrloc` module to store and retrieve AOR records in the MySQL database.

```
if (method=="REGISTER") {
       # Uncomment this if you want to use digest auth.
         if (!www_authorize("", "subscriber")) {
                 www_challenge("", "0");
                 exit;
         };
         save("location");
```

```
        exit;
} else if (method=="INVITE") {
        if (!proxy_authorize("","subscriber")) {
                proxy_challenge("","0");
                exit;
        };
        consume_credentials();
};
```

In the code snippet shown above, we will check the authentication for both methods `INVITE` and `REGISTER`.

If the method is `REGISTER` and the credentials are correct `www_authorize` returns `true`. After the authentication the system saves the location data for this UAC. The first parameter specifies the realm where the user will be authenticated. Realm usually is the domain name or host name. The second parameter tells OpenSER which MySQL table to look for.

```
www_challenge("","0");
```

If the packet does not have an Authorize header field we will send to the UAC the message "401 unauthorized". This tells the UAC to retransmit the request with the included digest credentials. The command `www_challenge` receives two parameters. The first one is the realm the UAC should use to compute the digest. The second one affects the inclusion of the qop parameter in the challenge. Using 1 will include the qop in the digest. Some phones may not support qop. You can try 0 in these circumstances.

```
consume_credentials();
```

We don't want to take risks sending the digest credentials to servers ahead. Therefore we use the function `consume_credentials()` to remove the Authorize header field from the request before relaying.

```
 if (!proxy_authorize("","subscriber")) {
```

We use the `proxy_authorize()` function to check for the authentication headers. If we didn't check the credentials we could be considered an open relay. The arguments are similar to those for `www_authorize`.

The Openserctl Shell Script

The utility `openserctl` is a shell script installed at `/usr/sbin`. It is used to manage OpenSER from the command line. It can be used to:

- Start, stop, and restart OpenSER
- Show, grant, and revoke ACLs
- Add, remove, and list aliases
- Add, remove, and configure an AVP
- Manage LCR (low cost routes)
- Manage RPID
- Add, remove, and list subscribers
- Add, remove, and show the usrloc table "in-ram"
- Monitor OpenSER

We will learn several of its options in the next sections. Below is the output of the `openserctl help` command:

```
/etc/openser# openserctl help
database engine 'MYSQL' loaded
Control engine 'FIFO' loaded
/usr/sbin/openserctl 1.2 - $Revision: 1.3 $
Existing commands:

 -- command 'start|stop|restart'

 restart ..... restart OpenSER
 start ..... start OpenSER
 stop ..... stop OpenSER

 -- command 'acl' - manage access control lists (acl)

 acl show [<username>] ..... show user membership
 acl grant <username> <group> ..... grant user membership (*)
 acl revoke <username> [<group>] .... grant user membership(s) (*)

 -- command 'alias_db' - manage database aliases

 alias_db show <alias> ..... show alias details
 alias_db list <sip-id> ..... list aliases for uri
 alias_db add <alias> <sip-id> ..... add an alias (*)
 alias_db rm <alias> ..... remove an alias (*)
 alias_db help ..... help message
   - <alias> must be an AoR (username@domain)"
   - <sip-id> must be an AoR (username@domain)"

 -- command 'avp' - manage AVPs

 avp list [-T table] [-u <sip-id|uuid>]
    [-a attribute] [-v value] [-t type] ... list AVPs
 avp add [-T table] <sip-id|uuid>
    <attribute> <type> <value> ..... add AVP (*)
```

```
avp rm [-T table]  [-u <sip-id|uuid>]
   [-a attribute] [-v value] [-t type] ... remove AVP (*)
avp help ..... help message
  - -T - table name
  - -u - SIP id or unique id
  - -a - AVP name
  - -v - AVP value
  - -t - AVP name and type (0 (str:str), 1 (str:int),
                    2 (int:str), 3 (int:int))
  - <sip-id> must be an AoR (username@domain)
  - <uuid> must be a string but not AoR

-- command 'db' - database operations

db exec <query> ..... execute SQL query
db show <table> ..... display table content

-- command 'lcr' - manage least cost routes (lcr)

        * lcr *
  * IP addresses must be entered in dotted quad format e.g. 1.2.3.4 *
  * <uri_scheme> and <transport> must be entered in integer or text,*
  * e.g. transport '2' is identical to transport 'tcp'.            *
  *   scheme: 1=sip, 2=sips;   transport: 1=udp, 2=tcp, 3=tls      *
  * Examples:  lcr addgw_grp usa 1                                 *
  *           lcr addgw level3 1.2.3.4 5080 sip tcp 1              *
  *           lcr addroute +1 % 1 1                              *
lcr show .....
        ..... show routes, gateways and groups
lcr reload .....
        ..... reload lcr gateways
lcr addgw_grp <grp_name> .....
        ..... add gateway group, autocreate grp_id
lcr addgw_grp <grp_name> <grp_id> .....
        ..... add gateway group with grp_id
lcr rmgw_grp  <grp_id> .....
        ..... delete the gw_grp
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> .....
        ..... add a gateway
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix> .....
        ..... add a gateway with prefix
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix> <strip>
        ..... add a gateway with prefix and strip
lcr rmgw  <gw_name> .....
        ..... delete a gateway
lcr addroute <prefix> <from> <grp_id> <prio> .....
        ..... add a route
```

```
lcr rmroute  <prefix> <from> <grp_id> <prio> .....................
        ..... delete a route

-- command 'rpid' - manage Remote-Party-ID (RPID)

rpid add <username> <rpid> ..... add rpid for a user (*)
rpid rm <username> ..... set rpid to NULL for a user (*)
rpid show <username> ..... show rpid of a user

-- command 'speeddial' - manage speed dials (short numbers)

speeddial show <speeddial-id> ..... show speeddial details
speeddial list <sip-id> ..... list speeddial for uri
speeddial add <sip-id> <sd-id> <new-uri> [<desc>] ...
        ..... add a speedial (*)
speeddial rm <sip-id> <sd-id> ..... remove a speeddial (*)
speeddial help ..... help message
  - <speeddial-id>, <sd-id> must be an AoR (username@domain)
  - <sip-id> must be an AoR (username@domain)
  - <new-uri> must be a SIP AoR (sip:username@domain)
  - <desc> a description for speeddial
-- command 'add|mail|passwd|rm' - manage subscribers

add <username> <password> <email> .. add a new subscriber (*)
passwd <username> <passwd> ..... change user's password (*)
rm <username> ..... delete a user (*)
mail <username> ..... send an email to a user

-- command 'cisco_restart' - restart CISCO phone (NOTIFY)

cisco_restart <uri> ..... restart phone configured for <uri>

-- command 'online' - dump online users from memory

online ..... display online users

-- command 'monitor' - show internal status

monitor ..... show server's internal status

-- command 'ping' - ping a SIP URI (OPTIONS)

ping <uri> ..... ping <uri> with SIP OPTIONS

-- command 'ul|alias' - manage user location or aliases

ul show [<username>]..... show in-RAM online users
ul rm <username> [<contact URI>].... delete user's UsrLoc entries
ul add <username> <uri> ..... introduce a permanent UrLoc entry
ul add <username> <uri> <expires> .. introduce a temporary UrLoc entry

-- command 'fifo'

fifo ..... send raw FIFO command
```

Openserctl Resource File

In version1.1 a resource file called `openserctlrc` was introduced. This script is found at `/etc/openser`. It is parsed by the `openserctl` utility to configure the database authentication and some communication parameters. Usually it uses the FIFO mechanism to send commands to the OpenSER daemon.

For security reasons is important to change the default user and password used to database access.

Openserctlrc File

To show the file, issue a command:

cat /etc/openser/openserctlrc

```
# $Id: openserctlrc,v 1.2 2006/07/05 19:37:20 miconda Exp $
#
# openser control tool resource file
#
# here you can set variables used in the openserctl

## your SIP domain
SIP_DOMAIN=voffice.com.br

## database type: MYSQL or PGSQL, by defaulte none is loaded
DBENGINE=MYSQL

## database host
DBHOST=localhost

## database name
DBNAME=openser

## database read/write user
DBRWUSER=openser

## database read only user
DBROUSER=openserro

## password for database read only user
DBROPW=openserro

## database super user
DBROOTUSER="root"

## type of aliases used: DB - database aliases; UL - usrloc aliases
## - default: none
ALIASES_TYPE="DB"

## control engine: FIFO or UNIXSOCK
```

```
## - default FIFO
CTLENGINE="FIFO"

## path to FIFO file
OSER_FIFO="FIFO"

## check ACL names; default on (1); off (0)
VERIFY_ACL=1

## ACL names - if VERIFY_ACL is set, only the ACL names from below list
## are accepted
ACL_GROUPS="local ld int voicemail free-pstn"

## verbose - debug purposes - default '0'
VERBOSE=1
## do (1) or don't (0) store plaintext passwords
## in the subscriber table - default '1'
# STORE_PLAINTEXT_PW=0
```

Using OpenSER with Authentication

Now, let's implement the authentication in a practical way.

Step 1: Make the changes described in this chapter to the `openser.cfg` file.

Step 2: Restart OpenSER with:

```
/etc/init.d/openser restart
```

Step 3: Configure `openserctlrc` with the default parameters used with `openserctl`.

```
# $Id: openserctlrc 1827 2007-03-12 15:22:53Z bogdan_iancu $
#
# openser control tool resource file
#
# here you can set variables used in the openserctl

## your SIP domain
SIP_DOMAIN=voffice.com.br

## database type: MYSQL or PGSQL, by defaulte none is loaded
DBENGINE=MYSQL

## database host
DBHOST=localhost

## database name
DBNAME=openser

## database read/write user
DBRWUSER=openser
```

```
## database read only user
DBROUSER=openserro

## password for database read only user
DBROPW=openserro

## database super user
DBROOTUSER="root"

## type of aliases used: DB - database aliases; UL - usrloc aliases
## - default: none
ALIASES_TYPE="DB"

## control engine: FIFO or UNIXSOCK
## - default FIFO
CTLENGINE="FIFO"

## path to FIFO file
OSER_FIFO="/tmp/openser_fifo"

## check ACL names; default on (1); off (0)
VERIFY_ACL=1

## ACL names - if VERIFY_ACL is set, only the ACL names from below
list
## are accepted
ACL_GROUPS="local ld int voicemail free-pstn"

## presence of serweb tables - default "no"
HAS_SERWEB="yes"

## verbose - debug purposes - default '0'
VERBOSE=1

## do (1) or don't (0) store plaintext passwords
## in the subscriber table - default '1'
STORE_PLAINTEXT_PW=1
```

Step 4: Configure two user accounts using the `openserctl` utility.

```
/sbin/openserctl add 1000 password 1000@voffice.com.br
/sbin/openserctl add 1001 password 1001@voffice.com.br
```

If you have a problem with "Duplicate Keys", please check that you haven't installed the SerWEB tables. If you had done so, Just change the line HAS_SERWEB to "yes".

When asked for the password use **openserrw**.

You can remove users using `openserctl rm` and change a password using `openserctl passwd`.

Step 5: Use the `ngrep` utility to see the SIP messages:

```
#ngrep -p -q -W byline port 5060 >register.pkt
```

Step 6: Register both phones now using name and password:

Step 7: Verify that the phones are registered using:

```
#openserctl ul show
```

Step 8: You can verify which users are online using:

```
#openserctl online
```

Step 9: You can ping a client using:

```
#openserctl ping 1000
```

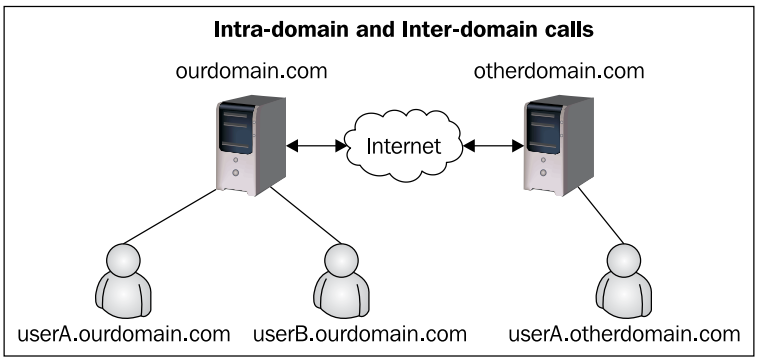
Step 10: Verify the authentication messages using the `ngrep` utilty.

Step 11: Make a call from one phone to the other.

Step 12: Verify the authentication in the `register.pkt` file using:

```
#pg register.pkt
```

Enhancing the Script



The calls handled by the SIP proxy could be classified as:

- Intra-domain
- Outbound Inter-domain
- Inbound Inter-domain
- Outbound-to-Outbound

Let's describe some problems with our current script:

Problem #1: At present we are not checking the identity for outbound calls coming from other domains. This makes our server an open relay. Thus any caller can use our server to hide their identity.

Problem #2: Our script does not accept a call coming from another domain.

Problem #3: A user can forge a FROM header field of an INVITE request using credentials from another user.

Problem #4: A user can forge a TO header field of a REGISTER request using credentials from another user.

Problem #5: The script is not prepared to manage multiple domains.

Managing Multiple Domains

Until now, we have verified the requests using the instruction `uri==myself`. However, this instruction verifies only local names and addresses. If we need to manage multiple domains we will have to use the domain module and its respective functions `is_from_local()` and `is_uri_host_local()`.

As I said before the domain module exports two functions that will be used in our script. The first one is `is_from_local()`, which verifies if the FROM header field contains one of the domains managed by our proxy. The second function, `is_uri_host_local()`, replaces the `uri==myself` instruction. The advantage of the domain exported functions is that they check the domain on a MySQL table (DOMAIN). Then you can handle multiple domains in your configuration.

This function requires that all served domains be inserted in the database. A fairly common mistake for users of this material is to forget to insert the domains in the MySQL database before starting to register the phones!

Alternative Routes

To simplify our script we will create several alternative routes. We have seen that the script can become very complex and confusing. To avoid this, we have created alternate routes working in a way similar of subroutines. Using alternative routes, allows us to separate certain pieces of code to enhance the readability.

Register Requests (route[2])

The register-request route is responsible for handling all REGISTER requests. The code authenticates the user and save the location of the UAC.

```
route[2] {
        #
        # -- Register request handler --
        #
        if (is_uri_host_local()) {
                if (!www_authorize("", "subscriber")) {
                        www_challenge("", "1");
                        exit;
                };
                if (!check_to()) {
                        sl_send_reply("401", "Unauthorized");
                        exit;
                };
                save("location");
                exit;
        } else if {
                sl_send_reply("401", "Unauthorized");
        };
}
```

Non-Register Requests (route[3])

The non-register request route handles all other requests. Again, the request needs to be authenticated. We have decided to separate requests into:

- Inbound-to-Inbound (route[10])
- Inbound-to-Outbound (route[11])
- Outbound-to-Inbound (route[12])
- Outbound-to-Outbound (route[13])

Usually, you will permit Inbound-to-Inbound requests with authentication. It is the normal situation. Inbound-to-Outbound and Outbound-to-Inbound are used to handle inter-domain requests. Most VoIP providers do not allow inter-domain communications, because it can potentially reduce the income. Outbound-to-Outbound calls are rarely permitted. In most cases, they are considered a security hole.

```
route[3] {
     #
     # -- non-register requests handler --
     #
     # Verify the source (FROM)
     if (is_from_local()){
     # From an internal domain -> check the credentials and the FROM
          if (!proxy_authorize("","subscriber")) {
               proxy_challenge("","1");
               exit;
          } else if (!check_from()) {
               sl_send_reply("403", "Forbidden, use From=ID");
               exit;
          };
          consume_credentials();
          # Verify aliases
          lookup("aliases");
          # Verify the destination (URI)
          if (is_uri_host_local()) {
               # -- Inbound to Inbound
               route(10);
          } else {
               # -- Inbound to outbound
                route(11);
          };
     } else {
          #Verify aliases, if found replace R-URI.
          lookup("aliases");
          # Verify the destination (URI)
           if (is_uri_host_local()) {
                #-- Outbound to inbound
                route(12);
           } else {
                # -- Outbound to outbound
                 route(13);
           };
     };
}
```

Managing Calls Coming from Our Domain

Our script `openser.cfg` now differentiates calls by source and destination. It determines the destination using the function `if(is_uri_host_local())` and the source using `if(is_from_local())`.

For inbound originated calls, we will first check the identity and remove the credentials to avoid sending them ahead. We will resolve any alias defined before checking the destination and forwarding the request.

If the call destination is one of our managed domains (checked using `is_uri_host_local()`) we will send it to route(10) else if it is an external domain we will send it to route(11).

Inbound-to-Inbound—**route[10]**

Inbound destinations will be handled by the user location database.

```
route[10] {
     #from an internal domain -> inbound
     #Native SIP destinations are handled using the location table
     append_hf("P-hint: inbound->inbound \r\n");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
}
```

Inbound-to-Outbound—**route[11]**

We will route calls to external destinations using a DNS search.

```
route[11] {
     # from an internal domain -> outbound
     # Simply route the call outbound using DNS search
     append_hf("P-hint: inbound->outbound \r\n");
     route(1);
}
```

Outbound-to-Inbound—**route[12]**

We will allow call from external domains to our phones. This configuration allows dialing spam to our phones, but at this time this is not a common practice. I believe the benefits are bigger than the risks. In the future nobody knows. It seems to me logical to be open to receive calls in the same way we are open to receive calls on our phones and we are open to receive emails.

```
route[12] {
     # From an external domain -> inbound
     # Verify aliases, if found replace R-URI.
     lookup("aliases");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
}
```

Outbound-to-Outbound—**route[13]**

We don't want to relay external messages as an open-relay. Someone else could use our proxy to route calls anonymously if this configuration were not in place.

```
route[13] {
     #From an external domain outbound
     #we are not accepting these calls
      sl_send_reply("403", "Forbidden");
      exit;
}
```

The Functions check_to() and check_from()

When operating a SIP proxy, you should guarantee that a valid account won't be used by non-authenticated users. The `check_to()` and `check_from()` functions are used to map the SIP users with the authentication user. The SIP user is in the FROM and TO header fields and the **auth** user is only used for authentication (Authorize header field) and it has its own password. In the current example, the function checks that a SIP user and the auth user are the same. This is to prevent a user form using the authentication of another user. These functions are enabled by the `URI_DB` module.

Using Aliases

In some cases you want to allow a user to have several addresses, such as the phone number associated to a main address. You can use Aliases for this purpose.

To add an Alias, use:

```
#openserctl alias add flavio@asteriskguide.com
sip:1000@asteriskguide.com
database engine 'MYSQL' loaded
Control engine 'FIFO' loaded
MySql password for user 'openser@localhost':
lookup("aliases");
```

The function `lookup("aliases")` checks the alias table in the database and if a register is found it translates it to the canonical address (the one in the subscribers table). This feature is also used to redirect DIDs to the final user. There is also the `Alias_db` module. It searches the alias directly from the database instead of memory. Even, having a small performance penalty, it can simplify the provisioning of alias directly in the database.

Handling CANCEL requests and retransmissions

Cancel requests according to the RFC3261, needs to be routed in the same way as the INVITE requests. The script below checks if the CANCEL request matches an existing transaction and takes care of all the necessary routing. Sometimes, we have retransmissions associated with an existing transaction. If this is the case, the function `t_check_trans()` will handle it and exit the script.

```
#CANCEL processing
if (is_method("CANCEL"))
{
if (t_check_trans())
    t_relay();
    exit;
}

t_check_trans();
```

Full Script with All the Resources Above

```
# ------------------ module loading ----------------------------------

#set module path
mpath="//lib/openser/modules/"

# Uncomment this if you want to use SQL database
loadmodule "mysql.so"

loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"
# ----------------- setting module-specific parameters ---------------

# -- mi_fifo params --

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")

# -- usrloc params --

#modparam("usrloc", "db_mode",   0)

# Uncomment this if you want to use SQL database
# for persistent storage and comment the previous line
modparam("usrloc", "db_mode", 2)

# -- auth params --
# Uncomment if you are using auth module
#
modparam("auth_db", "calculate_ha1", 0)
#
# If you set "calculate_ha1" parameter to yes,
# uncomment also the following parameter)
#
#modparam("auth_db", "password_column", "password")

# -- rr params --
# add value to ;lr param to make some broken UAs happy
```

```
modparam("rr", "enable_full_lr", 1)

# ------------------------  request routing logic -------------------
# main routing logic

route{
        # initial sanity checks -- messages with
        # max_forwards==0, or excessively long requests
        if (!mf_process_maxfwd_header("10")) {
                sl_send_reply("483","Too Many Hops");
                exit;
        };

        if (msg:len >=  2048 ) {
                sl_send_reply("513", "Message too big");
                exit;
        };

        # we record-route all messages -- to make sure that
        # subsequent messages will go through our proxy; that's
        # particularly good if upstream and downstream entities
        # use different transport protocol
        if (!method=="REGISTER")
                record_route();

        # subsequent messages withing a dialog should take the
        # path determined by record-routing
        if (loose_route()) {
                # mark routing logic in request
                append_hf("P-hint: rr-enforced\r\n");
                route(1);
        };

   #CANCEL processing
        if (is_method("CANCEL")) {
            if (t_check_trans()) t_relay();
            exit;
        }
        if (method=="REGISTER") {
                route(2);
        } else {
                route(3);
        };
}

route[1] {
        # send it out now; use stateful forwarding as it works reliably
        # even for UDP2TCP
        if (!t_relay()) {
```

```
                sl_reply_error();
        };
        exit;
}
route[2] {
        #
        # -- Register request handler --
        #
        if (is_uri_host_local()) {
                if (!www_authorize("", "subscriber")) {
                        www_challenge("", "1");
                        exit;
                };
                if (!check_to()) {
                        sl_send_reply("40=3", "Forbidden");
                        exit;
                };
                save("location");
                exit;
        } else if {
                sl_send_reply("403", "Forbidden");
        };
}
route[3] {
        #
        # -- INVITE request handler --
        #
        if (is_from_local()){
            # From an internal domain -> check the credentials
              and the FROM
                if (!proxy_authorize("","subscriber")) {
                        proxy_challenge("","1");
                        exit;
                } else if (!check_from()) {
                       sl_send_reply("403", "Forbidden, use From=ID");
                        exit;
                };
                consume_credentials();
                # Verify aliases
                lookup("aliases");
                if (is_uri_host_local()) {
                        # -- Inbound to Inbound
```

```
                    route(10);
                } else {
                    # -- Inbound to outbound
                    route(11);
                };
        } else {
                # From an external domain -> do not check credentials
                #Verify aliases, if found replace R-URI.
                lookup("aliases");
                if (is_uri_host_local()) {
                   #-- Outbound to inbound
                   route(12);
                } else {
                   # -- Outbound to outbound
                   route(13);
                };
        };
}
route[10] {
     #from an internal domain -> inbound
     #Native SIP destinations are handled using the location table
     append_hf("P-hint: inbound->inbound \r\n");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
}
route[11] {
     # from an internal domain -> outbound
     # Simply route the call outbound using DNS search
     append_hf("P-hint: inbound->outbound \r\n");
     route(1);
}
route[12] {
     # From an external domain -> inbound
     # Verify aliases, if found replace R-URI.
     lookup("aliases");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
```

```
}
route[13] {
     #From an external domain outbound
     #we are not accepting these calls
     append_hf("P-hint: outbound->inbound \r\n");
     sl_send_reply("403", "Forbidden");
     exit;
}
```

Lab—Enhancing the Security

Step 1: Try to register your phone with the new configuration. You will probably note an error on your phone registration.

Step 2: The configuration above now uses the module `domain.so`. Now, to authenticate, the domain has to be inside the domain table in the MySQL database.

To add a domain, use the openserctl utility.

openserctl domain add your-ip-address
openserctl domain add your-domain

Repeat the process for every domain.

Step 3: Try again to register the phone. Now probably the register process will work fine.

Lab—Using Aliases

Step1: Add an alias to the subscriber 1000.

#openserctl alias add john@youripordomain sip:1000@youripordomain
database engine 'MYSQL' loaded
Control engine 'FIFO' loaded
MySql password for user 'openser@localhost':

Use openserrw as the password.

Step 2: From the softphone registered as "1001" dial John.

The call was completed?, Why?

Summary

In this chapter you have learned how to integrate MySQL with OpenSER. Now our script is authenticating users, checking the TO and FROM header fields, and handling accordingly inbound and outbound calls. It's important to remember that domains now have to be inserted in the database, because of the multiple domain support. If you change your domain or IP addresses, please remember to update your database.

6
Building the User Portal with SerMyAdmin

In the last chapter we implemented authentication using a MySQL database. Now we will need a tool to help users and administrators. Obviously, this tool has to be easier than **openserctl**. It is very hard to manage thousands of users manually, so a user provisioning tool becomes very important in our process. In this chapter will well look at the SerMyAdmin tool, created specifically to help building user and administrator portals.

By the end of this chapter you will be able to:

- Identify why you need a user portal for administration
- Install SerMyAdmin and its dependencies
- Configure resources such as administrator and user access
- Add and remove domains
- Customize the portal with the colors and logos of your company

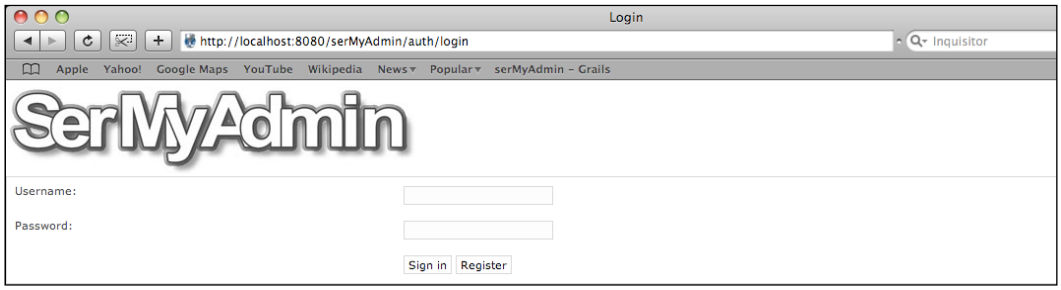
SerMyAdmin

Originally, this material was written for SerWeb. SerWeb was originally developed for the SER project. Unfortunately, SerWeb became incompatible with newer versions of OpenSER. Another important aspect of SerWeb to be considered is its vulnerabilities. There are very few options for web interfaces to OpenSER. One of the tools we have found is OpenSER administrator. This tool is beeing developed using Ruby on Rails. While it seems to be a very good tool to administer an OpenSER server, it does not permit to provisioning users in the same way that SerWeb did and it lacks multi-domain support. OpenSER administrator can be found at `http://sourceforge.net/projects/openseradmin`.

Since a tool to build an OpenSER portal was not available , we decided to build our own tool named SerMyAdmin using Java. After a slow start, it is now ready and we are using it to build this book. It is licensed according to GPLv2 and developed in Grails (Groovy on rails). It can be downloaded at `http://sourceforge.net/projects/sermyadmin`.

What you are seeing here is the standalone tool. In our roadmap, we intend to integrate SerMyAdmin into the Liferay portal. Using a content management system such as Liferay (`www.liferay.com`) will make your task of building a portal much easier than it is today.

The SerMyAdmin project can be found at `sermyadmin.sourceforge.net`. The idea is to facilitate the administration of the OpenSER database. SerMyAdmin is licensed under the GPLv2.



Lab—Installing SerMyAdmin

SerMyAdmin uses the Grails framework, so it needs an application server. You can choose from many application servers, such as IBM WebSphere, JBoss, Jetty, Tomcat, and so on. In this book we will use Apache Tomcat, because it's free and easy to install. Because we use some Java 1.5 features, we'll need Sun's Java JDK, not the free alternative GCJ.

Step 1: Create an administrator for SerMyAdmin:

```
mysql -u root
    use openser
    INSERT INTO 'subscriber' ( 'id' , 'username' , 'domain' , 'password'
    , 'first_name' , 'last_name' , 'email_address' , 'datetime_created' ,
    'ha1' , 'ha1b' , 'timezone' , 'rpid' , 'version' , 'password_hash' ,
    'auth_username' , 'class' , 'domain_id' , 'role_id' )
    VALUES (
    NULL , 'admin', 'openser.org', 'senha', 'Admin', 'Admin', 'admin@
    openser.org', '0000-00-00 00:00:00', '1', '1', '1', '1', '1', NULL ,
    'admin@openser.org', NULL , '1', '3'
    );
```

Step 2: The next step we will take is to update our source's list to use the `contrib` repository and `non-free` packages. Our `/etc/apt/sources.list`, should look like below:

```
# /etc/apt/souces.list
deb http://ftp.br.debian.org/debian/ etch main contrib non-free
deb-src http://ftp.br.debian.org/debian/ etch main contrib non-free

deb http://security.debian.org/ etch/updates main contrib non-free
deb-src http://security.debian.org/ etch/updates main contrib non-free
/etc/apt/sources.list
```

Notice that we have added only the keywords `contrib` and `non-free` after our repository definitions.

Step 3: Update the package listing using the following command:

```
openser:~# apt-get update
```

Step 4: Install Sun's Java 1.5, running the command below:

```
openser:~# apt-get install sun-java5-jdk
```

Step 5: Make sure you are using Sun's Java. Please, run the command below to tell Debian that you want to use Sun's Java as your default Java implementation.

```
openser:~# update-java-alternatives -s java-1.5.0-sun
```

Step 6: If everything has gone well so far, you should run the following command and get a similar output.

```
openser:~# java -version
```

java version "1.5.0_14"

Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_14-b03)

Java HotSpot(TM) Client VM (build 1.5.0_14-b03, mixed mode, sharing)

Step 7: Install Tomcat. You can obtain Tomcat at:

`http://tomcat.apache.org/download-60.cgi`.

To install Tomcat, just run the commands below:

```
openser:/usr/local/etc/openser# cd /usr/local
openser:/usr/local# wget http://mirrors.uol.com.br/pub/apache/tomcat/
tomcat-6/v6.0.16/bin/apache-tomcat-6.0.16.tar.gz
openser:/usr/local# tar zxvf apache-tomcat-6.0.16.tar.gz
openser:/usr/local# ln -s apache-tomcat-6.0.16 tomcat6
```

Step 8: To start Tomcat on your server initialization, please copy the following script to `/etc/init.d/tomcat6`.

```
#! /bin/bash –e
#### BEGIN INIT INFO
# Provides:          Apache's Tomcat 6.0
# Required-Start:    $local_fs $remote_fs $network
# Required-Stop:     $local_fs $remote_fs $network
# Default-Start:     2 3 4 5
# Default-Stop:      S 0 1 6
# Short-Description: Tomcat 6.0 Servlet engine
# Description:       Apache's Tomcat Servlet Engine
### END INIT INFO
#
# Author: Guilherme Loch Góes <glwgoes@gmail.com>
#
set -e
PATH=/bin:/usr/bin:/sbin:/usr/sbin:
CATALINA_HOME=/usr/local/tomcat6
CATALINA_BIN=$CATALINA_HOME/bin
test -x $DAEMON || exit 0
. /lib/lsb/init-functions
case "$1" in
    start)
        echo "Starting Tomcat 6" "Tomcat6"
        $CATALINA_BIN/startup.sh
        log_end_msg $?
    ;;
  stop)
        echo "Stopping Tomcat6" "Tomcat6"
        $CATALINA_BIN/shutdown.sh
        log_end_msg $?
    ;;
  force-reload|restart)
    $0 stop
    $0 start
    ;;
  *)
    echo "Usage: /etc/init.d/tomcat6 {start|stop|restart}"
    exit 1
    ;;
esac

exit 0
```

Step 9: Instruct Debian to run your script on startup; we do this with the command below.

```
openser: chmod 755 /etc/init.d/tomcat6
openser:/etc/init.d# update-rc.d tomcat6 defaults 99
```

Step 10: To make sure everything is running correctly, reboot the server and try to open in your browser the URL `http://localhost:8080`; if everything is OK you'll be greeted with Tomcat's start page.

Step 11: Install the MySQL driver for Tomcat, so that SerMyAdmin can access your database. This driver can be found at `http://dev.mysql.com/downloads/connector/j/5.1.html`. You should download the driver and unpack it, then copy the connector to Tomcat's shared library directory, as follows.

```
openser:/usr/src# tar zxf mysql-connector-java-5.1.5.tar.gz
openser:/usr/src# cp mysql-connector-java-5.1.5/mysql-connector-java-
5.1.5-bin.jar /usr/local/tomcat6/lib
```

Step 12: Declare the data source for SerMyAdmin to connect to OpenSER's database. You can do this in an XML file found at `/usr/local/tomcat6/conf/context.xml`. The file should look as below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/serMyAdmin">
  <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver"
maxActive="20" maxIdle="10" maxWait="-1" name="jdbc/openser_MySQL"
type="javax.sql.DataSource" url="jdbc:mysql://localhost:3306/openser"
username="sermyadmin" password="secret"/>
</Context>
```

In the file above, please change the highlighted parameters according to your scenario. SerMyAdmin can be installed in a different server than the one that holds the database. Do this for better scalability when possible. The default MySQL installation on Debian only accepts requests from localhost, so you should edit the file `/etc/mysql/my.cnf`, for MySQL to accept requests from external hosts.

Step 13: Create a user to be referenced in the file `context.xml`. This user will have the required access to the database. Please, run the commands below:

```
openser:/var/lib/tomcat5.5/conf# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
```

Server version: 5.0.32-Debian_7etch5-log Debian etch distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

```
mysql> grant all privileges on openser.* to sermyadmin@'%' identified
by 'secret';
```

Query OK, 0 rows affected (0.00 sec)

Step 14: We're almost there. The next step is to deploy the SerMyAdmin WAR file. Please, download and copy the file `serMyAdmin.war` to Tomcat's webapps directory. Restart it, to activate the changes.

```
openser:/usr/src# cp serMyAdmin-0.4.war /usr/local/tomcat6/webapps/
serMyAdmin.war
openser:/usr/src# invoke-rc.d tomcat6 restart
```

Don't worry about database modifications; SerMyAdmin will automatically handle that for you.

Step 15: Configure Debian's MTA (Message Transfer Agent) to allow SerMyAdmin to send a confirmation email to new users. Run the command below to configure Exim4 (default MTA for Debian). Ask your company's email administrator.

```
openser:/# apt-get install exim4
openser:/# dpkg-reconfigure exim4-config
```

You will be greeted with a dialog-based configuration menu; on this menu it's import to pay attention to two options: **General type of mail configuration**, which should be set to **Internet Site** so that we can send and receive mails directly using SMTP, and **Domains to relay mail for**, which should be set to the domain from which you want the emails from SerMyAdmin to appear to come.

Step 16: Customize the file `/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/WEB-INF/spring/resource.xml`, which contains the parameters that specify which email server is used to send mails and from whom these emails should appear to come from. The following is an example of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.
springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="mailSender" class="org.springframework.mail.javamail.
JavaMailSenderImpl">
        <property name="host"><value>localhost</value></property>
    </bean>

    <!-- You can set default email bean properties here, eg: from/to/
subject -->
    <bean id="mailMessage" class="org.springframework.mail.
SimpleMailMessage">
        <property name="from"><value>admin@sermyadmin.org</value></
property>
    </bean>

</beans>
```

The first parameter to change is the server that we will use to send emails. The second is the parameter specifying from whom those emails will appear to come.

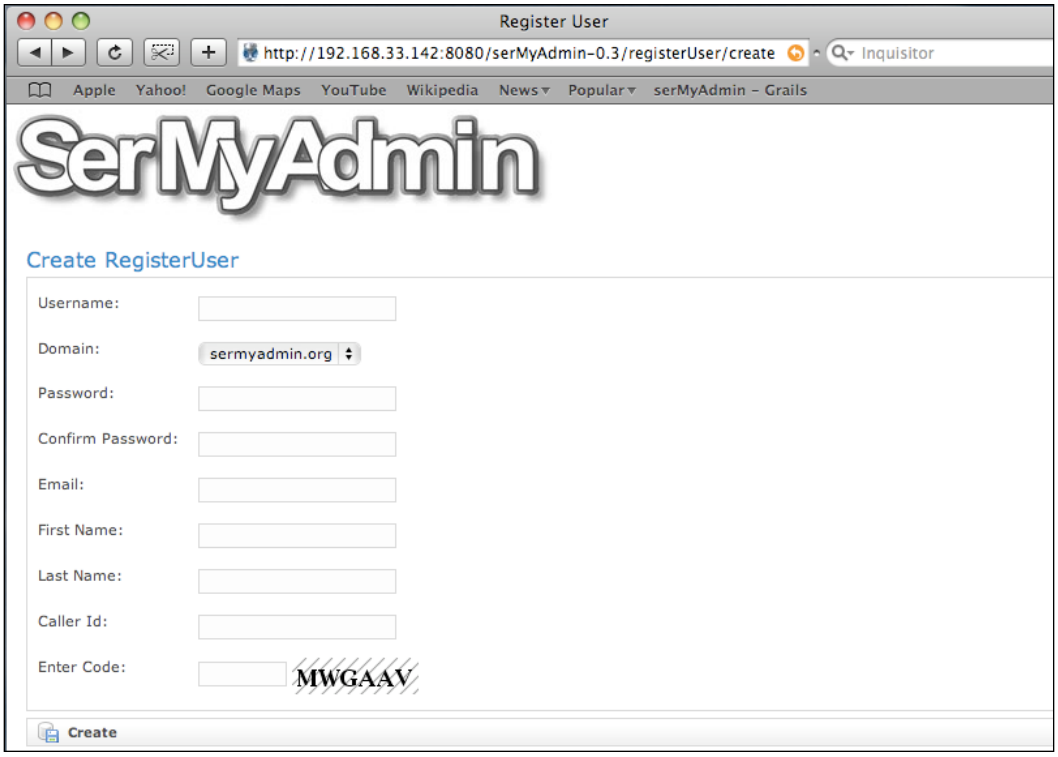
Restart Tomcat again and we're ready to go. When you point your browser to `http://<server address>:8080/serMyAdmin` you should be greeted with the login page, the same as we have shown at the start on this chapter.

Basic Tasks

You can now use SerMyAdmin for a bunch of tasks. In this chapter we will show you how to create and administer new users and groups. In the next chapters we will use SerMyAdmin for other tasks such as managing the trust table and the LCR module.

Registering a New User

To register a new user, in the login screen simply click on the **Register** Button.



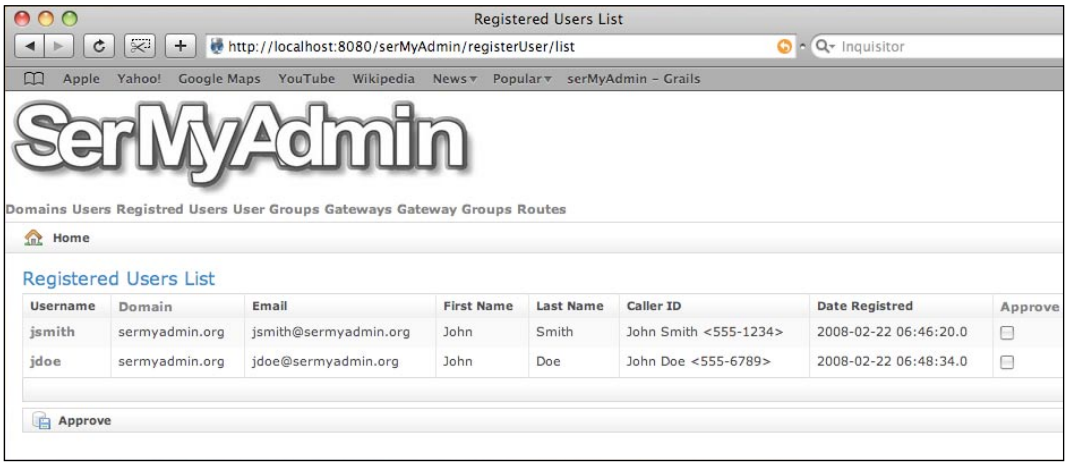
Fill up the fields, **Username**, **Password**, **Domain**, **Email**, **First Name**, **Last Name**, **Caller Id**, and the confirmation code. Press the **Create** button at the end of the screen. The user will be added to the database. Both the system administrator and the user will receive an email about the registration. Before the user can make any calls, the administrator will have to approve the user.

Approving a New User

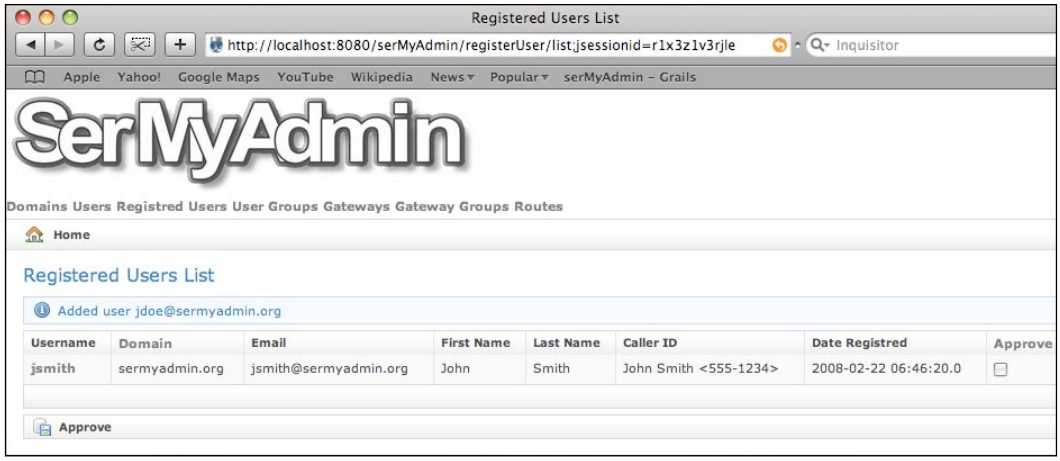
Follow this step-by-step procedure to approve a new user:

Step 1: Log in with the `admin@localhost` account, password `openserrw` created during the OpenSER installation. The installation process has created a new attribute named **Role** for every user. The purpose of this column is to differentiate normal users, domain administrators, and global administrators. The admin user was automatically set to **Global Administrator**. This new field will help us to provide multi-domain support.

Step2: Select the menu item **Registered Users**.



In the screen above, select the users you want to add and check the box **Approve**. Press the button **Approve** to add the user; the user will be removed from the `register_user` table and moved to the subscriber table, and then, he will be able to register to OpenSER and make calls.

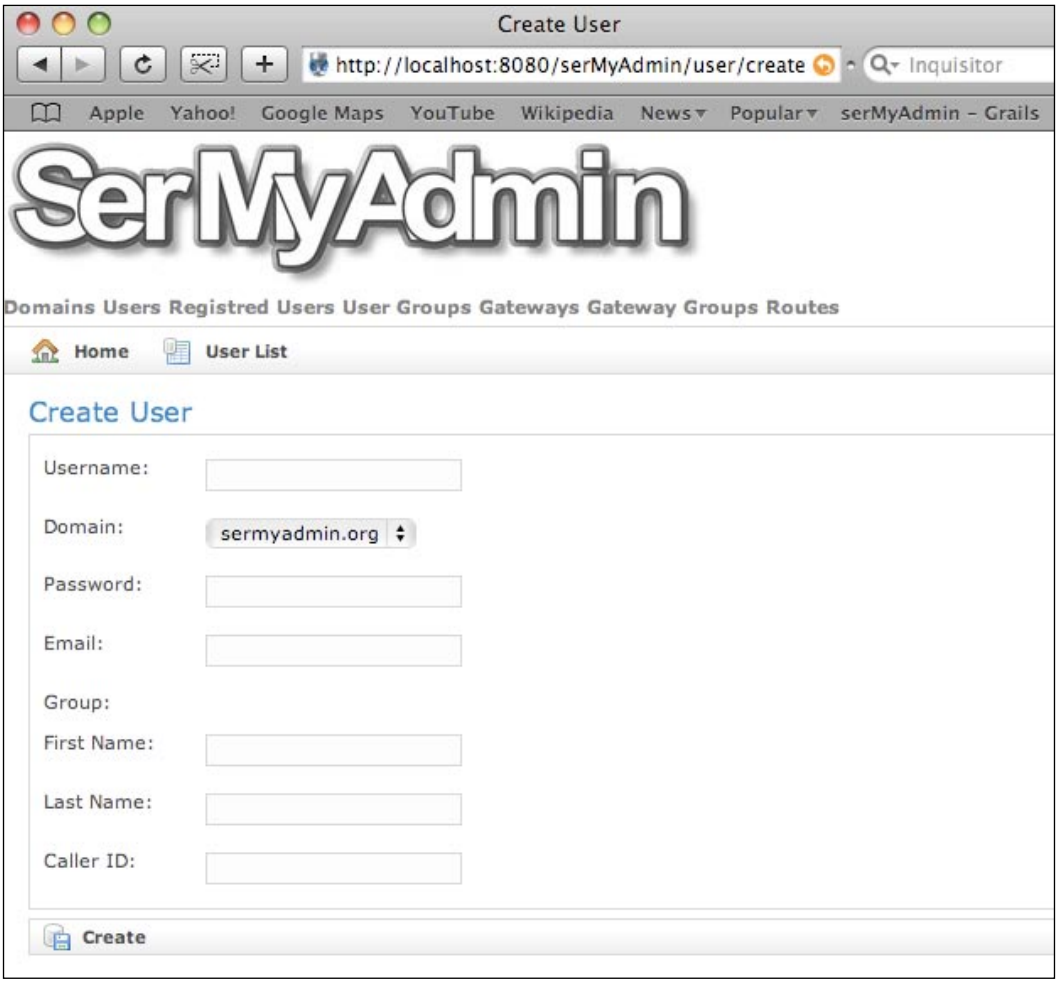


Step 3: The user now should appear on your user list. Check this by clicking the menu item **Users**.

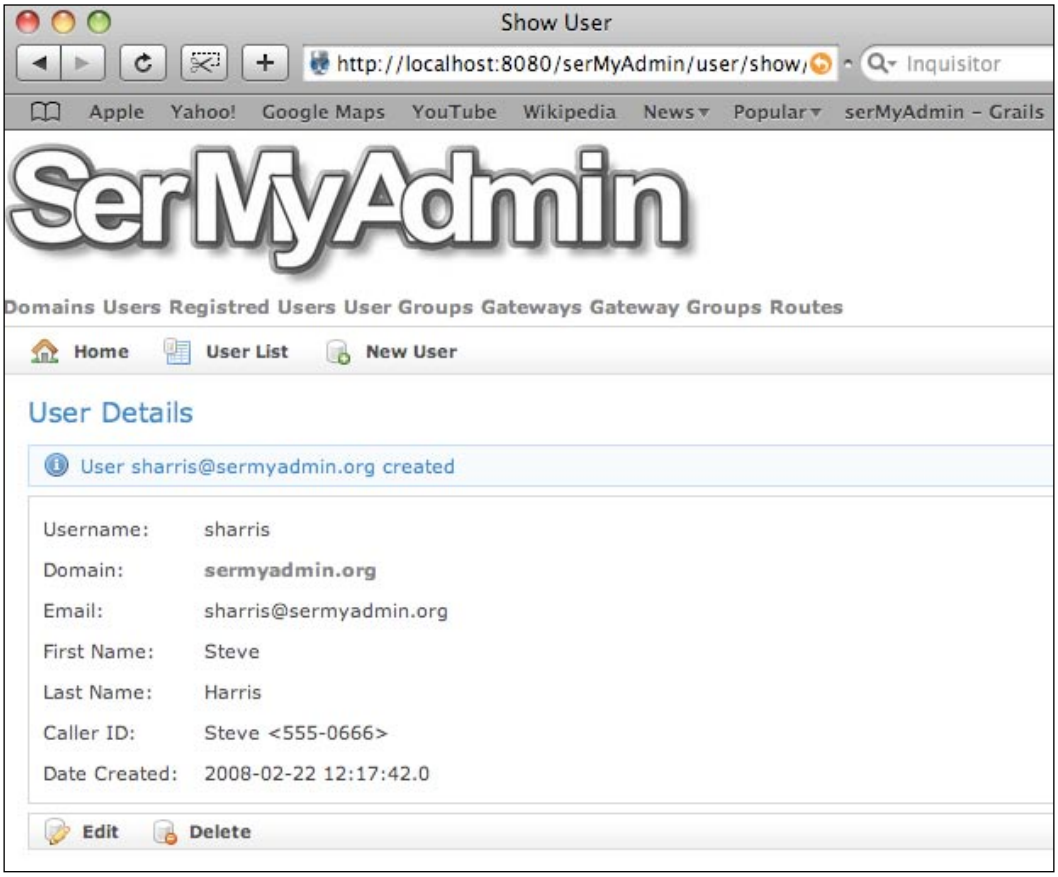
User Management

You can view, add, edit, and delete users on the **Users** menu. When you click it, you'll have all users on the system displayed.

To add a new user you must click on the **New User** link. You'll be directed to the page below:

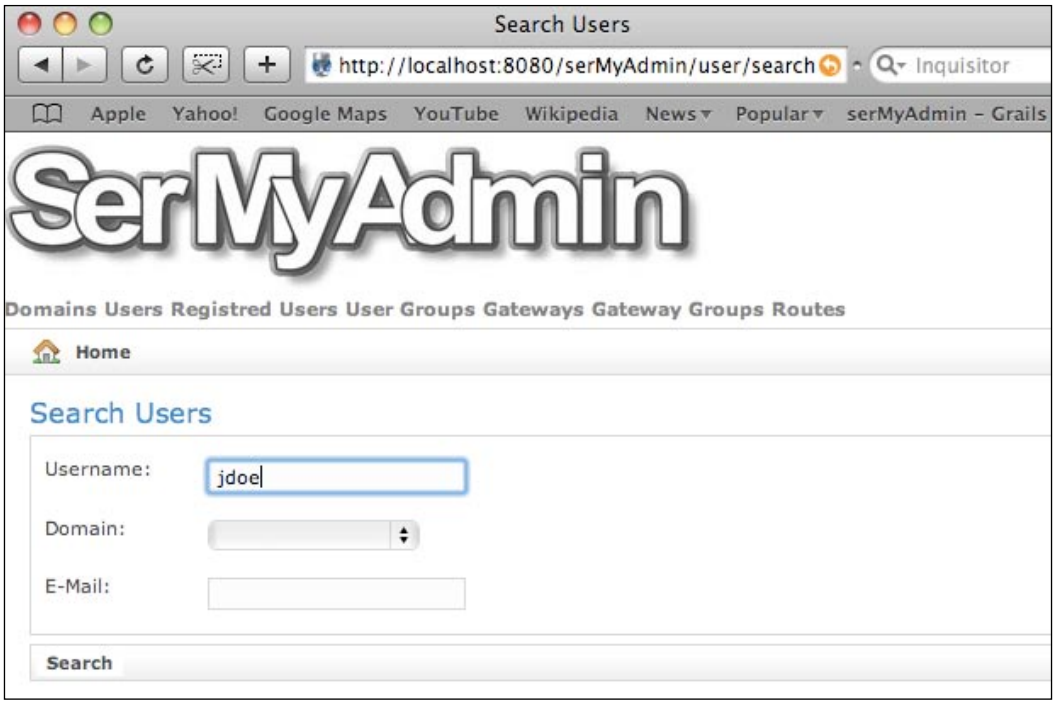


In the preceding page you must complete the fields and click on the **Create** link, then the user will be added to the subscriber table. The following page will be shown after this:



In this page you can modify the information inserted by clicking on **Edit**, or delete the user by clicking on **Delete**.

On the **User List** page you can search for users based on username, domain, and email; just click on the **Search** link, and fill out only the desired criteria. On the following page, we will search for all users with the username **jdoe**. Click **Search**; you'll be directed to the **User List** that matches your criteria.



Domain Management

You can manage your domains in the same way as you manage your users. Click on **Domains** to get a domain list. There, you can add a new domain, delete an existing domain, and so on. It is important to note that SerMyAdmin doesn't allow a user to exist without a domain, so when you delete a domain you also delete all users that belong to that domain.

Interface Customization

For its site layout SerMyAdmin uses a SiteMesh framework, so it's pretty simple to customize the look of SerMyAdmin to your taste. SiteMesh displays the pages based on a template that can be found at `openser:/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/WEB-INF/grails-app/views/layouts`. There you'll find `main.gsp` and `notLoggedIn.gsp`; these files are Groovy Server Pages that control how the pages are displayed.

SiteMesh uses HTML meta tags to choose which layout to use; those tags should be found in the head element of each page, that is, if a page has the tag `<meta content="main" name="layout"/>` inside its head element, SiteMesh will use the `main.gsp` layout to display it.

You can change `main.gsp` and `notLoggedIn.gsp` as you wish, but it's important to understand that `<g:layoutHead />` and `<g:layoutBody />` will hold the head and body tags of the pages using this layout. Another thing to know is that `<g:render template="/menu" />` is used to render page fragments; these page fragments are GSP files, and their filename should start with an underscore ("_").

To replace the SerMyAdmin logo with one of your own, just put your logo on `/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/images`, and edit the tag that points to the `logo_voffice.png` in the layout files, just as shown below:

```
<div class="logo"><img src="${createLinkTo(dir:'images',file:
'my_new_logo.png')}" alt="Grails" /></div>
```

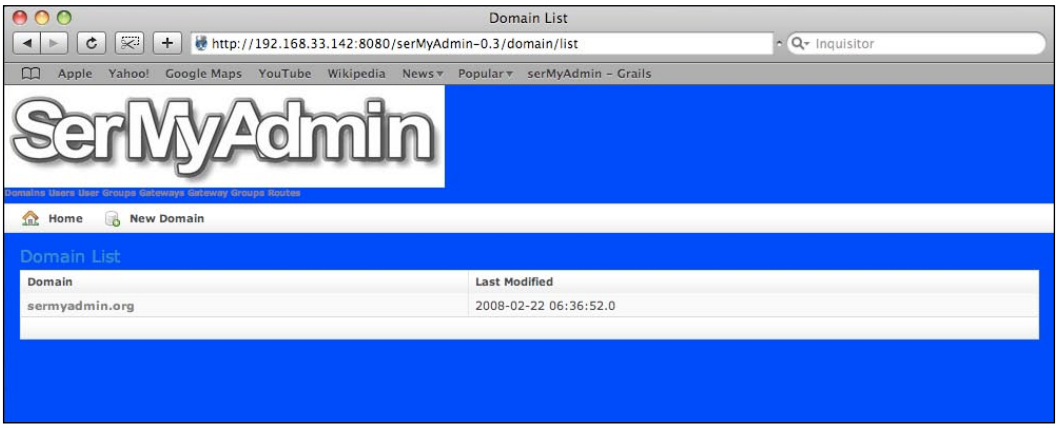
In the above tag we replaced the SerMyAdmin logo with one of our own, just changing by the parameter in bold.

You can also change the look and feel of SerMyAdmin by modifying its CSS file, which can be found at `/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/css`; in the `main.css` file we'll find every class to change SerMyAdmin behavior.

Example: if we change the background class in this file with the following parameters:

```
body {
    background: #00f;
    color: #333;
    font: 8px verdana, arial, helvetica, sans-serif;
}
```

we will end getting a page that looks like this:



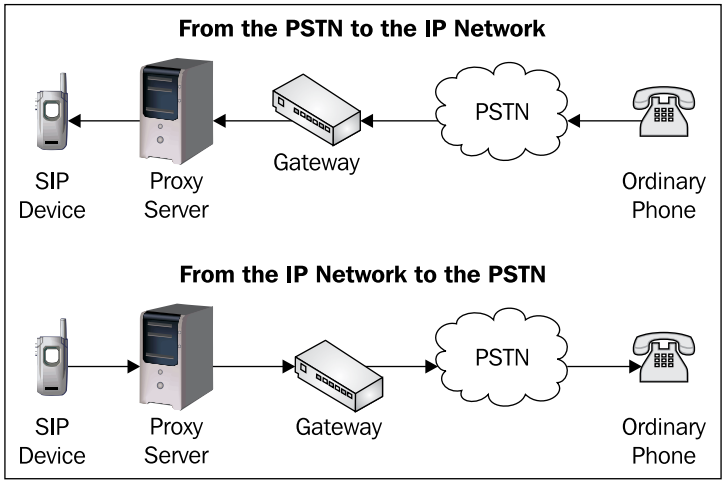
This page isn't the prettiest thing on the planet, but you can follow this example to make it look a lot better.

Summary

In this chapter, you have learned why it is important to have a user and administrator portal. It is a piece of software to which you should pay a lot of attention. Several VoIP providers fail to allocate time and resources to the important task of building the portal. OpenSER is an amazing SIP proxy, but a SIP proxy is just one of the components in a VoIP Provider. Without a good administrator and user interface a VoIP provider project can easily fail. SerMyAdmin is our contribution to your project. Developed in Java using Groovy on Rails it is licensed according to GPL version 2. You have learned how to install, manage users and domains, and how to customize the appearance. The tool can do a lot more things, and we will show it again in the next chapters in some other tasks.

7
Connectivity to the PSTN

In the last two chapters, we have prepared OpenSER to handle calls using authentication and a database. SerMyAdmin was used to handle the database records. However, you still can't send calls to ordinary phones, because you are not connected to the PSTN. The challenge now is to route calls from and to the PSTN (Public Switched Telephone Network).



To send calls to the PSTN, you will need a device called a SIP PSTN Gateway. There are several manufacturers of this kind of equipment in the market such as Cisco, AudioCodes, Nortel, Quintum, and others. You can also use an Asterisk PBX box for this task. Asterisk makes an affordable PSTN gateway that is very competitive with the big players mentioned above. It is fully open source, licensed according to GPL.

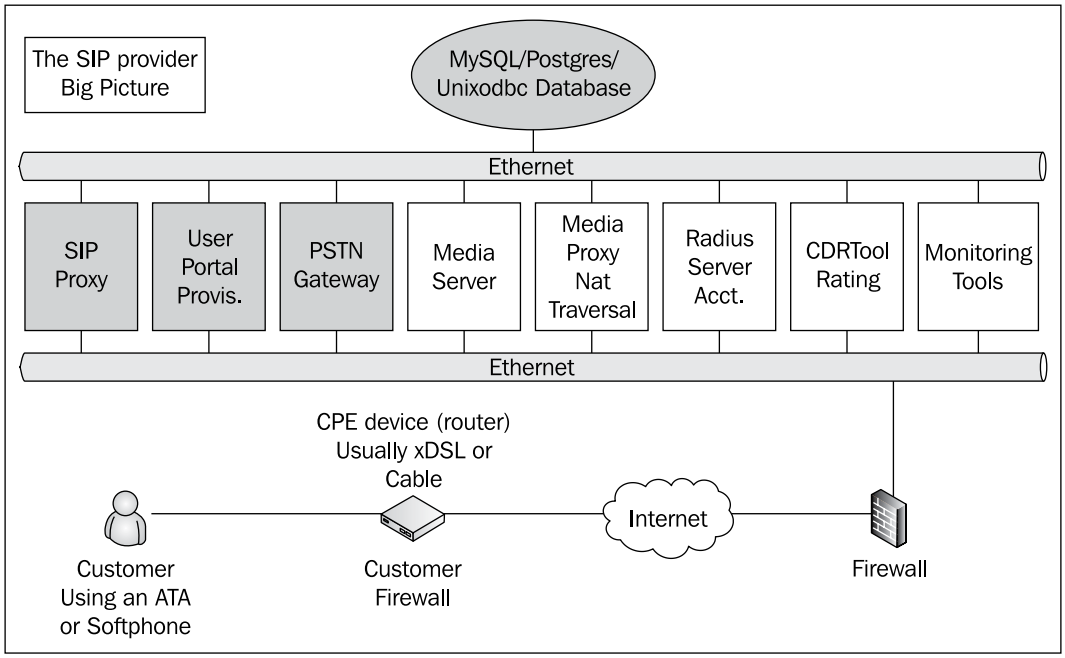
By the end of this chapter, you will be able to:

- Interconnect OpenSER to a SIP gateway
- Apply permissions to inbound calls
- Use ACLs to protect the PSTN gateway from unauthorized use
- Use the LCR (Least Cost Route) module to route your calls
- Use SerMyAdmin to manage Trusted Hosts, Gateways, and Routes

In this chapter you will learn how to send calls to the PSTN. We will introduce three new modules (LCR, PERMISSIONS, and GROUP), which will help you to route and secure these calls. It is important to understand a little about regular expressions, because they will be used to route the calls. It is very easy to find a tutorial for *regexps* on the Internet. If you are not familiar with regular expressions or regexps, this quick reference card can help: `http://www.visibone.com/regular-expressions/`.

Where Are We?

The VoIP provider solution has many components. To avoid loosing the perspective, we will show this picture on every chapter. In this chapter, we are working with the SIP proxy component together with the PSTN Gateway.



After this chapter our VoIP provider will be able to send calls to the PSTN using a SIP gateway.

Requests Sent to the Gateway

In the requests addressed to the gateway, we have to verify to which group a certain user belongs to check if he or she is allowed to use the PSTN.

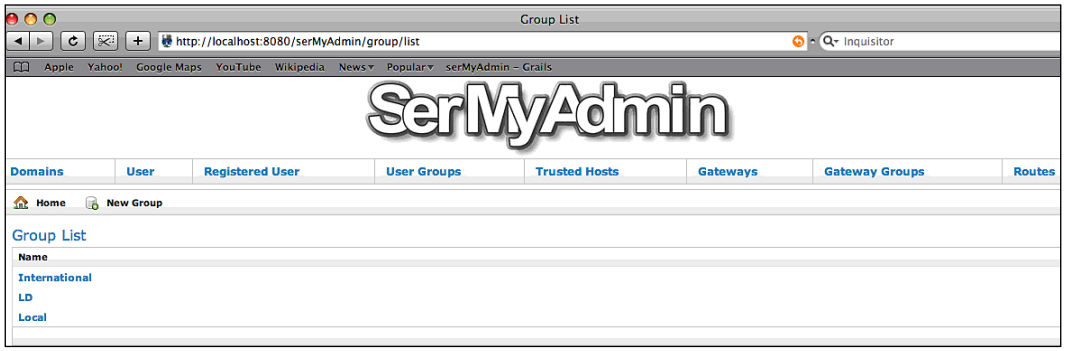
```
if (uri=~"^sip:[2-9][0-9]{6}@") {
        if (is_user_in("credentials","local")) {
                route(5);
                exit;
        } else {
                sl_send_reply("403", "No permissions for local calls");
                exit;
        };
};
if (uri=~"^sip:[2-9][0-9]{9}@") {
        if (is_user_in("credentials","ld")) {
                route(5);
                exit;
        } else {
                sl_send_reply("403", "No permissions for long distance calls");
                exit;
        };
};
if (uri=~"^sip:011[0-9]*@") {
        if (is_user_in("credentials","int")) {
                route(5);
                exit;
        } else {
                sl_send_reply("403", "No permissions for international calls");
        };
};
```

For this purpose we are going to use the 'group' module. This module exports the function `is_user_in ("credentials", "group")` to check if a user belongs to a specified group. In the above example we had created three groups: `local` for local calls, `ld` for long distance, and `int` for international calls. In the script we use regular expressions to check if the calls are local, long distance, or international.

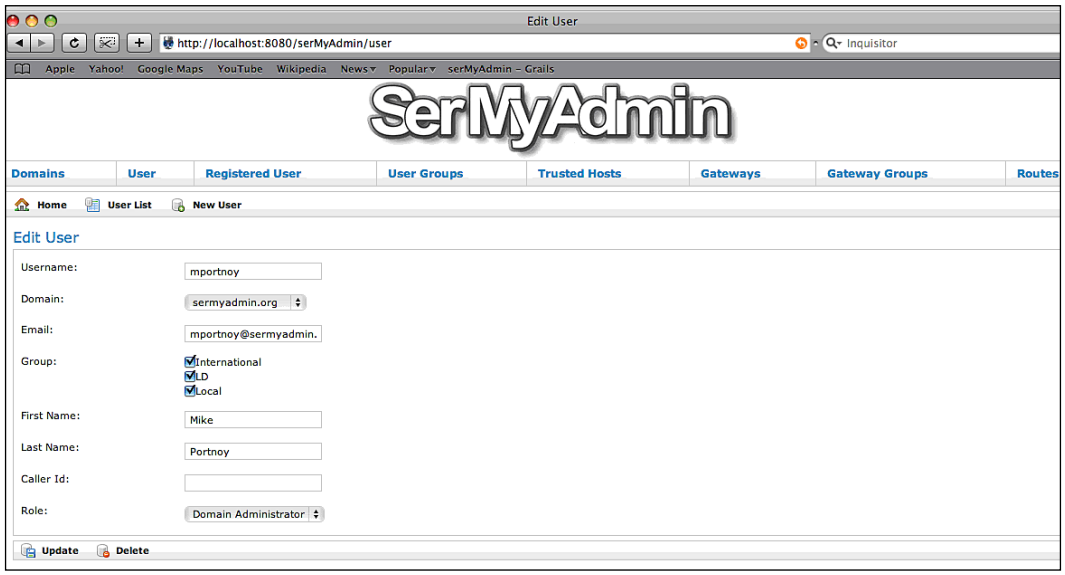
You have to insert the groups in the MySQL table called `group` before to use it. You can easily insert, remove, and show group membership using:

- `openserctl acl show [<username>]`
- `openserctl acl grant <username> <group>`
- `openserctl acl revoke <username> [<group>]`

It is possible too, to manage your tables using SerMyAdmin. To add and remove groups, you can browse the **User Groups** section. There, you can add and delete groups.



To change user's group membership, you can go to the user menu as shown here:



Use the checkboxes to select the specific groups to which the user belongs as shown here.

Requests Coming From the Gateway

Now, we will use the module PERMISSIONS to authorize calls coming from the PSTN Gateway without the digest authentication process.

Handling of calls coming from the PSTN gateway

```
if (!allow_trusted()) {
        if (!proxy_authorize("","subscriber")) {
                proxy_challenge("","0");
                exit;
        } else if (!check_from()) {
                sl_send_reply("403", "Use From=ID");
                exit;
        };
};
```

The function `allow_trusted()` that we will use is exported by the PERMISSIONS module. The permissions module can be used to authorize REGISTER, REFER, and INVITE requests. It can be regulated by the files `permissions.allow`, `permissions.deny`, `register.allow`, and `register.deny`. However, the `allow_trusted()` function used in this module checks the source IP address of the request against the 'trusted' table of our database.

When called, the function `allow_trusted()` tries to find a rule matching the request. The rules contain the following fields <IP source address>, <transport protocol>, <regular expression>.

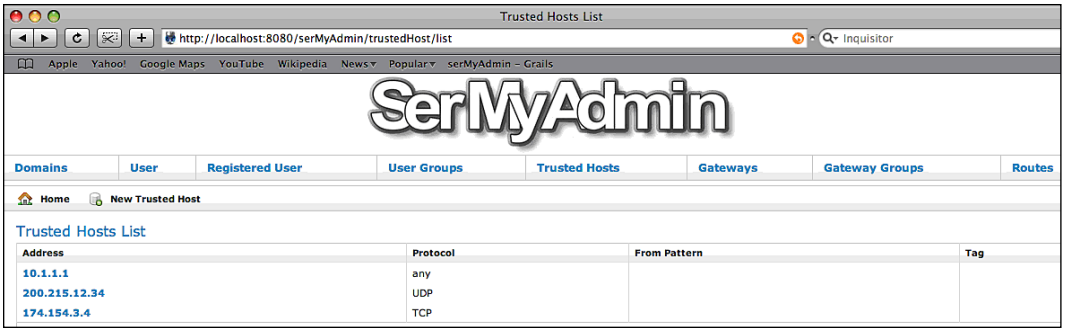
A request is accepted if a rule exists where:

- The IP address is equal to the IP source address of the request.
- The transport protocol is *any* or matches the transport protocol of the request.
- The regular expression is empty or matches the request.

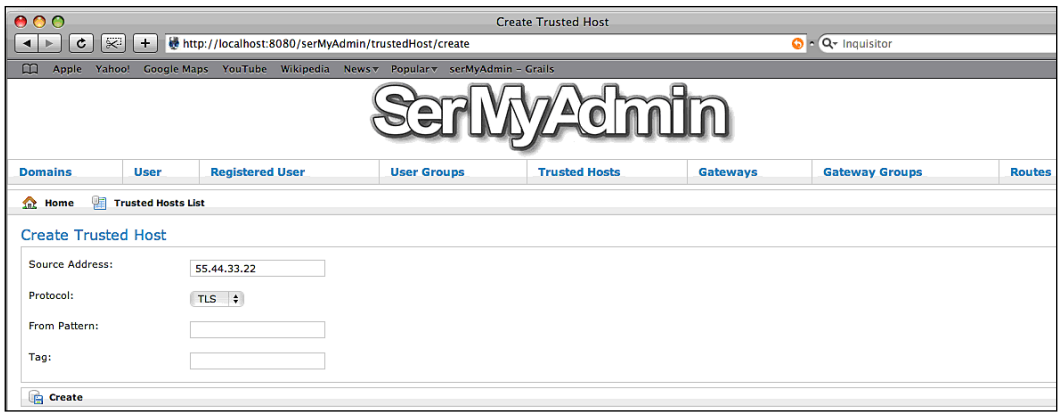
It is very usual for gateways not to register on the SIP proxy. Thus, requests coming from the gateway should not receive a "407 Proxy Authentication Required" response. In our current script all `INVITE` requests coming from our domain are challenged for their credentials. However, if a request like this is sent from the gateway, it probably won't have the credentials to be sent and the call will fail. To fix this we will just check the source IP address using the `allow_trusted()` function instead of checking the credentials.

Don't forget to insert the trusted IP addresses in the trusted table of our MySQL database for this script to work.

You can view and update the trusted hosts list using SerMyAdmin. Use the trusted hosts menu as shown below:



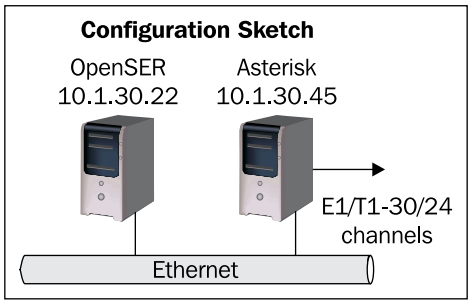
To add new hosts, simply click in the **New Trusted Host** menu item.



To forward the call to the PSTN gateway use the function `rewritehostport()`.

Route the call to the PSTN Gateway

```
route[4] {
    ##--
    ## Send the call to the PSTN
    ## Change the IP address below to reflect
    ## your network
    ##--
    rewritehostport("10.1.30.45");
    route(1);
}
```



This script is named `openser.pstn`. It is found at `http://www.sermyadmin.org/openser`. A copy is shown below. Changes from previous scripts are highlighted.

```
# ------------------ module loading ----------------------------------
#set module path
mpath="//lib/openser/modules/"

loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"

# ----------------- setting module-specific parameters ---------------

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|usrloc","db_url",
         "mysql://  openser:openserrw@localhost/openser")
```

```
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
# ------------------------  request routing logic -------------------
# main routing logic
route{
    #
    # -- 1 -- Request Validation
    #
    if (!mf_process_maxfwd_header("10")) {
           sl_send_reply("483","Too Many Hops");
           exit;
    };
    if (msg:len >=  2048 ) {
           sl_send_reply("513", "Message too big");
           exit;
    };
    #
    # -- 2 -- Routing Preprocessing
    #
    ## Record-route all except Register
    if (!method=="REGISTER") record_route();
    ##Loose_route packets
    if (loose_route()) {
           # mark routing logic in request
           append_hf("P-hint: rr-enforced\r\n");
           route(1);
    };
#CANCEL processing
     if (is_method("CANCEL")) {
         if (t_check_trans()) t_relay();
         exit;
     };

     t_check_trans();
    #
    # -- 3 -- Determine Request Target
    #
    if (method=="REGISTER") {
           route(2);
    } else {
           route(3);
    };
}
route[1] {
    #
```

```
    # -- 4 -- Forward request to target
    #
    ## Forward statefully
    if (!t_relay()) {
           sl_reply_error();
    };
    exit;
}

route[2] {
    ## Register request handler
    if (is_uri_host_local()) {
           if (!www_authorize("", "subscriber")) {
                  www_challenge("", "1");
                  exit;
           };

           if (!check_to()) {
                  sl_send_reply("403", "Forbidden");
                  exit;
           };

           save("location");
           exit;
    } else if {
           sl_send_reply("403", "Forbidden");
    };
}

route[3] {
    ## INVITE request handler
    if (is_from_local()){
            # From an internal domain -> check the credentials
              and the FROM
         if(!allow_trusted()){
              if (!proxy_authorize("","subscriber")) {
                   proxy_challenge("","1");
                   exit;
              } else if (!check_from()) {
                   sl_send_reply("403", "Forbidden, use From=ID");
                   exit;
              };
         } else {
              log("Request bypassed the auth.using allow_trusted");
         };

         consume_credentials();
```

```
            #Verify aliases, if found replace R-URI.
            lookup("aliases");

            if (is_uri_host_local()) {
                # -- Inbound to Inbound
                route(10);
        } else {
                # -- Inbound to outbound
                route(11);
        };
    } else {
          #From an external domain ->do not check credentials
          #Verify aliases, if found replace R-URI.
          lookup("aliases");
          if (is_uri_host_local()) {
                   #-- Outbound to inbound
                   route(12);
          } else {
                   # -- Outbound to outbound
                   route(13);
          };
    };
}

route[4] {
      # routing to the public network
      rewritehostport("10.1.30.45");
      route(1);
}

route[10] {
      #from an internal domain -> inbound
      #Native SIP destinations are handled using the location table
      #Gateway destinations are handled by regular expressions
      append_hf("P-hint: inbound->inbound \r\n");

      if (uri=~"^sip:[2-9][0-9]{6}@") {
          if (is_user_in("credentials","local")) {
              route(4);
              exit;
          } else {
              sl_send_reply("403", "No permissions for local calls");
              exit;
          };
      };
```

```
        if (uri=~"^sip:1[2-9][1-9]{9}@") {
            if (is_user_in("credentials","ld")) {
                 route(4);
                 exit;
            } else {
               sl_send_reply("403", "No permissions for long distance");
                 exit;
            };
        };

        if (uri=~"^sip:011[0-9]*@") {
            if (is_user_in("credentials","int")) {
                route(4);
                exit;
            } else {
                sl_send_reply("403", "No permissions for
                                    international calls");
            };
        };
        if (!lookup("location")) {
            sl_send_reply("404", "Not Found");
            exit;
        };
        route(1);
}
route[11] {
     # from an internal domain -> outbound
     # Simply route the call outbound using DNS search
     append_hf("P-hint: inbound->outbound \r\n");
     route(1);
}
route[12] {
     # From an external domain -> inbound
     # Verify aliases, if found replace R-URI.
     lookup("aliases");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
}
route[13] {
```

```
        #From an external domain outbound
        #we are not accepting these calls
        append_hf("P-hint: outbound->inbound \r\n");
        sl_send_reply("403", "Forbidden");
        exit;
    }
```

openser.cfg Inspection

The PERMISSIONS module exports some essential functions to control access to our SIP proxy. One of this functions is `allow_trusted()` which will permit us to control the gateway access to the proxy using the IP address instead of the authentication credentials. The `trusted` table is the repository for the trusted addresses. You should insert the IP address and transport protocol of each gateway to this database. This will allow the requests coming from the gateway to avoid the standard digest authentication.

The PERMISSIONS module, has standard permission and deny files too. We are not using these features at this time. To avoid messages in the log, please copy the files from the `config` directory of the PERMISSIONS module to the `/etc/openser` directory.

```
cp /usr/src/openser-1.2.2/modules/permissions/config/* /etc/openser
```

In the permissions files, it is possible to filter requests based on regular expressions, improving the security of the environment. Check the sample files for the right syntax.

The module `group.so` will be used to check the user's group membership. This is called an ACL (Access Control List). You can add, remove, or show the user ACLs using the `openserctl` utility.

```
loadmodule "permissions.so"
loadmodule "group.so"
```

The first line below informs the module where to find the database passing the required credentials. The second line instructs the modules to use caching on the database access to enhance the performance.

```
modparam("auth_db|permissions|uri_db|usrloc","db_url",
"mysql://openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
```

When your Proxy server receives an `INVITE` request the usual behavior is to challenge the UAC for its credentials. However, PSTN gateways usually do not respond to the authentication. Thus you need to adopt a special procedure. The function `allow_trusted()` will check the source IP address of the `INVITE` request against the 'trusted' table of our database. If it matches the request will be allowed. If it doesn't match the requester will be challenged for the credentials.

```
if(!allow_trusted()){
   if (!proxy_authorize("","subscriber")) {
           proxy_challenge("","0");
           exit;
   } else if (!check_from()) {`
         sl_send_reply("403","Forbidden, use FROM=ID");
           exit;
   };
 };
```

It is important that the gateway's IP address be properly inserted in the database.

You can use utilities such as SerMyAdmin or phpMyAdmin to maintain the database. It is easier than doing it manually in the MySQL CLI (command line interface).

In the 'trusted' table insert the gateway's IP address, transport protocol (udp, tcp, tls, any), and regular expression.

Below we have the routing of calls according to regular expressions:

```
if (uri=~"^sip:[2-9][0-9]{6}@") {
  if (is_user_in("credentials","local")) {
     route(4);
     exit;
  } else {
     sl_send_reply("403", "No permissions for local calls");
     exit;
  };
};
if (uri=~"^sip:1[2-9][0-9]{9}@") {
  if (is_user_in("credentials","ld")) {
    route(4);
    exit;
  } else {
    sl_send_reply("403", "No permissions for long distance");
    exit;
```

```
    };
  };
  if (uri=~"^sip:011[0-9]*@") {
    if (is_user_in("credentials","int")) {
      route(4);
      exit;
    } else {
      sl_send_reply("403", "No permissions for
                    internat. calls");
      exit;
    };
  };
```

Local calls are identified by the number of digits (7) and starting with a number in the range of 2 to 9 (`"^sip:[2-9][0-9]{6}@"`). Long distance numbers will match the regular expression `"^sip:1[2-9][0-9]{9}@"`, numbers starting with 1 followed by 2 to 9, plus 9 digits will be considered long distance. Finally, International numbers are prefixed with 011+Country Code+Area Code+Dialed Number. In all cases the script is sent to route 4.

It is important to insert the ACL data in the database for this script to work:

You can do this using the `openserctl` utility, SerMyAdmin, or phpMyAdmin.

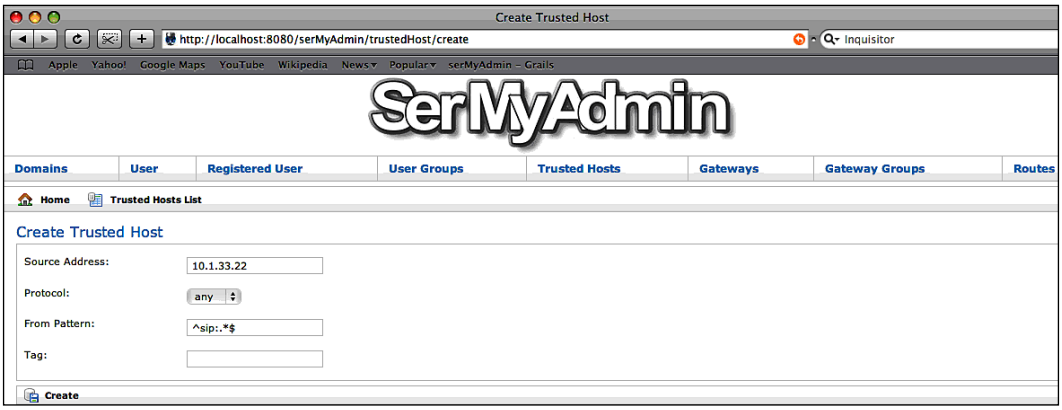
Finally, we have routing block 4 to handle PSTN destinations. The function `rewritehostport()` is used to change the host part of the URI in such a way that it will be sent to the gateway when you relay the request using `t_relay()`.

```
route[4] {
        ##--
        ## PSTN gateway handling
        ##--
        rewritehostport("10.1.30.45");
        route(1);
}
```

Lab—Using Asterisk as a PSTN Gateway

Using the sketch provided in this chapter, write a script to send PSTN calls to a PSTN Gateway. You can use phpMyAdmin, or the MySQL command line to insert data on the database.

Step 1—Add the gateway address in the trusted table using SerMyAdmin:



If desired or convenient, you can instead use the MySQL command line interface to achieve the same result.

```
#mysql -u openser -p
-- enter your mysql password --
mysql> use openser;
mysql> INSERT INTO trusted ( src_ip, proto, from_pattern )
VALUES ( '10.1.30.22', 'any', '^sip:.*$');
```

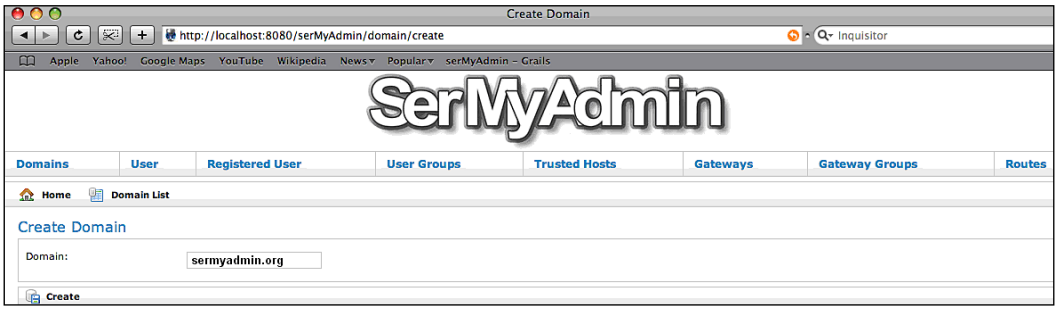
The records above tell the OpenSER script to allow requests coming from the IP address `10.1.30.22` with any transport protocol, matching the regular expression `^sip:.*$`. You can use the following command if you don't want to reload OpenSER.

```
#openserctl fifo trusted_reload
```

Step 2—Include your served domains in the domain table (if you have not done before).

```
openserctl domain add sermyadmin.org
```

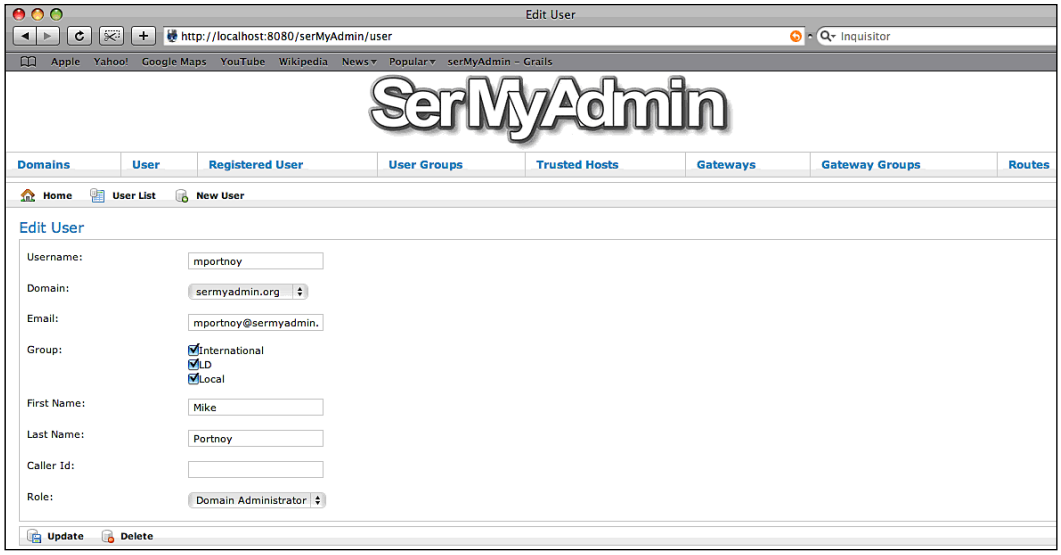
You can also use SerMyAdmin to do this.



Step 3—Include the user into the groups (local, ld, and int):

```
#opensercctl acl grant 1000@sermyadmin.org local
#opensercctl acl grant 1000@sermyadmin.org ld
#opensercctl acl grant 1000@sermyadmin.org int
#opensercctl acl grant 1001@sermyadmin.org local
```

To use SerMyAdmin, just go to the screen below:



Step 4—Configuring Asterisk as a gateway.

Two very popular gateways for OpenSER are Asterisk and Cisco AS5300. Gateways from other manufacturers can be used too; check their documentation for instructions. Let's see how to configure a Cisco 2601 with two FXO interfaces and an Asterisk with an E1 PSTN card.

Warning

It is important to prevent the direct sending of SIP packets to gateways. The SIP proxy should be in front of the gateway and a firewall should prevent users from sending SIP requests directly to the gateway.

Step 5—Setting up the Asterisk Server or the Cisco Gateway.

We will assume that the PSTN side of the Asterisk gateway is already configured. Now let's change the SIP configuration (`sip.conf`) of our gateway and its dial plan (`extensions.conf`). We will configure Asterisk to send to the proxy each call coming from the PSTN and vice versa. We are using the guest feature of the SIP channel on the Asterisk Server. Prior knowledge of Asterisk is required here. Below is the simplest configuration allowing Asterisk to communicate with OpenSER. Please, adapt this script to your topology.

Warning

Allow SIP packets to your asterisk server, coming only from your SIP server. Do not allow SIP packets coming from other destinations. You can use IP Tables to do this, consult a Linux security specialist, if you are in doubt.

Asterisk Gateway (sip.conf)

```
[general]
context=sipincoming
#calls incoming from the SIP proxy to be terminated in the PSTN lines

[sipproxy]
#calls incoming from the PSTN to be forwarded to clients behind the
SIP
#proxy
type=peer
host=10.1.30.22
Asterisk (extensions.conf)
[general]

[globals]

[sipincoming]
exten=>_[0-9].,1,Dial(Zap/g1/${EXTEN:1})
exten=>_[0-9].,2,hangup()
[sipoutgoing]
# If you have a digital interface use the lines below
exten=_[0-9].,1,Answer()
exten=_[0-9].,2,dial(SIP/${EXTEN}@sipproxy)
```

```
exten=_[0-9].,3,Hangup()

#If you have analog FXO interfaces use the lines below.
exten=s,1,Answer()
exten=s,2,dial(SIP/${EXTEN}@sipproxy)
exten=s,3,Hangup()
```

Cisco 2601 Gateway

The following explanation could help, but prior knowledge of Cisco gateways is required to complete this configuration. The call routing on Cisco gateways is done by the instruction dial peer. Any call with the number called starting with 9 followed by any number (9T) is forwarded to the PSTN on the ports 1/0 or 1/1 as instructed by the dial peer voice 1 and 2 POTS lines (plain old telephone system). Called numbers starting from 1 to 9 with any number of digits following will be directed to the SIP proxy in the IP address 10.1.3.22 as instructed in the 'dial-peer voice 123 voip' line.

```
voice class codec 1
 codec preference 2 g711ulaw
!
interface Ethernet0/0
 ip address 10.1.30.38 255.255.0.0
 half-duplex
!
ip classless
ip route 0.0.0.0 0.0.0.0 10.1.0.1
no ip http server
ip pim bidir-enable
!
voice-port 1/0
!
voice-port 1/1
!
mgcp profile default
!
! The dial-peer pots commands will handle the calls coming from SIP
!dial-peers. Any call matching 9 followed by any number of digits will
be !forwarded to the PSTN with the 9 striped.

dial-peer voice 1 pots
 destination-pattern 9T
    port 1/0
!
dial-peer voice 2 pots
```

```
 destination-pattern 9T
   port 1/1
!
!The dial-peer voip commands will handle the calls coming from the
pots !dial peers (PSTN). You can prefix a number (80 in this example)
and send the DID number ahead.
!
dial-peer voice 123 voip
 destination-pattern ....T
 prefix 80
 forward all
 session protocol sipv2
 session target ipv4:10.1.30.22
 dtmf-relay sip-notify
```

Step 6— Test the configuration making and receiving calls.

Using LCR (Least Cost Routes)

The last configuration is fine, if you have a few gateways and the routes do not change often. However, most SIP providers change routes very often. Besides, most of them have multiple gateways and connections to VoIP wholesale providers. It is counterproductive to change the script every time a route needs to be changed. So the LCR module will be used. It allows you to insert the routes and gateways in the database and change them dynamically to adjust the system to your requirements.

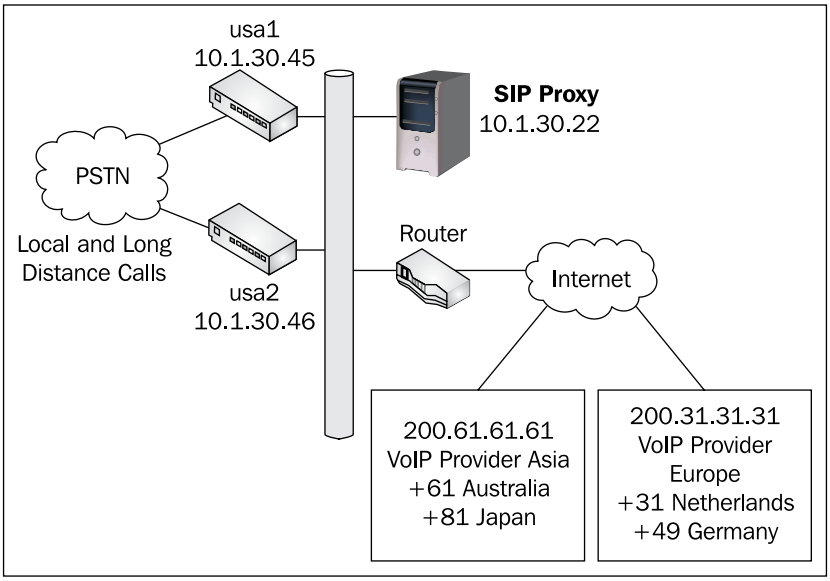
Imagine a situation where you have two wholesale providers, one in Europe and another in Asia. You want to send local and long distance calls to your own gateways, calls to the Netherlands, Germany, and France to the provider in Europe, and calls to Japan and Australia to the Asia provider. If the Asia provider fails, you want to fallback the call to your own gateways.

The LCR Module

The LCR module implements two capabilities. The most important is the sequential forwarding of a request to one or more gateways (`load_gws()` and `next_gw()`). These functions will be used to send the calls to the gateways and even failover the gateways using `failure_route` and `next_gw()`. The gateways can have a priority assigned to them to specify which gateway to select first.

You can also use the LCR module for sequential forwarding of contacts based on **q** value using `load_contacts()` and `next_contacts()`.

Configuration Diagram



To implement a diagram such as that above, we need to understand the three tables involved, `lcr`, `gw`, and `gw_grp`.

VoIP Provider Dial Plan

It is time to start elaborating a dial plan for our VoIP provider. We will implement the E.164 numbering scheme to direct calls to the right gateways. When the customer calls a local number we will use regular expressions to transform the number to their canonical E.164 address before selecting a gateway using the `load_gw()` function. This provider won't accept any number in a format different from the E.164 except for the subscribers. We are assuming that the user interface will help the user to dial the E.164 number.

Destination	Numbering	Description
VoIP provider callers	"8[0-9]{5}"	Six digits starting with 8
Long distance calls	"1[2-9][0-9]{9}"	1+Area Code+Subscriber(7 digits)
International Calls	"+[0-9]*"	Any number starting with +
International calls	"011[0-9]*"	Any number starting with 011

The LCR Table

In the `lcr` table you will implement the routes. The `lcr` fields are described below. Local calls and long distance calls will be prefixed with the number +1305

Prefix	From_uri	Grp_id	Priority
+31		3	1
+49		3	1
+61		2	1
+81		2	1
+1		1	1

- Prefix—This prefix is matched against the user part of the URI (phone number).
- From_uri—The From_uri may contain a URI to be matched. It is used sometimes when you want to route using caller information.
- Grp_id—The Grp_id identifies the gateway group.
- Priority—Gateway priority.

The Gateways Table

gw_name	grp_id	ip_addr	port	uri_scheme	transport	strip	prefix
Usa1	1	10.1.30.45	5060	1	1	2	
Usa2	1	10.1.30.46	5060	1	1	2	
Asia1	2	200.61.61.61	5060	1	1	0	
Europe1	3	200.31.31.31	5060	1	1	0	

- gw_name—Gateway name
- grp_id—Gateway group identification
- ip_addr—IP address of the gateway (for versions below 1.2.x, use inverse decimal notation)
- port—(UDP/TCP port)
- uri_scheme—sip (1), sips (2)
- transport—udp (1), tcp (2), tls (3)
- strip—Number of characters to be striped
- prefix—Prefix to be applied before sending to the gateway

The Gateway Groups Table

The gateway groups table is used for administrative purposes only. It associates the name to the gateway.

Adding, Removing, and Showing LCR and Gateways

You can add, remove, and show the LCR table and Gateways table using `openserctl` or SerMyAdmin. You can do it manually too.

If you are using OpenSer 1.0.x or 1.1.x, just observe that the IP address for this field must be in the inverse decimal notation.

To convert the IP address to the decimal inverse notation, do the following calculation:

- IP Address=a.b.c.d
- IP Address in the inverse decimal notation = $d*2^{24}+c*2^{16}+b*2^{8}+a$

If you don't want to calculate, just use the `openserctl` utility. It works fine for all versions. In the OpenSER 1.2.x you can insert the IP address in the usual decimal dotted notation directly in the database.

Openserctl LCR-Related Commands.

Command	Description
lcr show	Show routes, gateways, and groups
lcr reload	Reload lcr gateways
lcr addgw_grp <grp_name>	Add gateway group, autocreate grp_id
lcr addgw_grp <grp_name> <grp_id>	Add gateway group with grp_id
lcr rmgw_grp <grp_id>	Delete the gw_grp
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id>	Add a gateway
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix>	Add a gateway with prefix
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix> <strip>	Add a gateway with prefix and strip
lcr rmgw <gw_name>	Delete a gateway
lcr addroute <prefix> <from> <grp_id> <prio>	Add a route
lcr rmroute <prefix> <from> <grp_id> <prio>	Delete a route

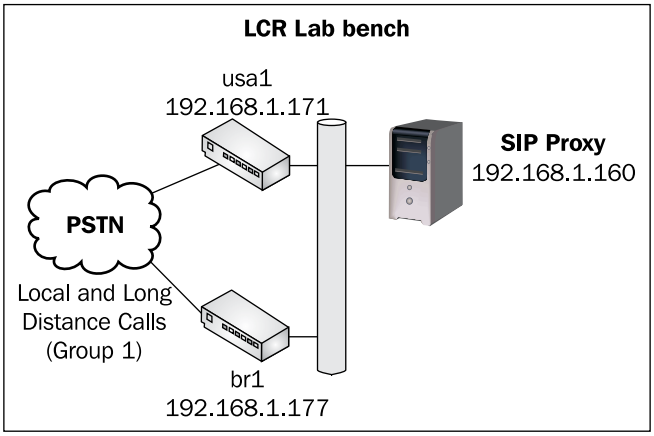
Notes:

- IP addresses must be entered in dotted quad format e.g. 1.2.3.4.
- <uri_scheme> and <transport> must be entered in integer or text:
    - transport '2' is identical to transport 'tcp'
    - scheme: 1=sip, 2=sips; transport: 1=udp, 2=tcp, 3=tls

Examples:

```
lcr addgw_grp usa 1
lcr addgw level3 1.2.3.4 5080 sip tcp 1
lcr addroute +1 ' ' 1 1
```

Lab—Using the LCR Feature



Let's execute a simple lab with LCR. For this lab we will need one OpenSER server, two gateways, and an IP phone. You can easily simulate this lab using Asterisk as the gateways and virtual machines.

Step 1: Build the LAB with the gateways. Configure the gateway named usa1 to receive calls with the +1 prefix and the br1 gateway to receive calls prefixed by +55. In the gateway you can prefix and strip the data before sending to the PSTN. To prefix and strip numbers, use the `strip()` and `prefix()` core functions. See the documentation for futher details at `www.openser.org`.

Step 2: Download the configuration file from `http://www.sermyadmin.org/openser/openser.lcr` and copy it to `openser.cfg`.

```
cd /etc/openser
wget http://www.sermyadmin.org/openser/openser.lcr
cp openser.lcr openser.cfg
The script can be seen below with the modifications highlighted.
# ------------------ module loading ----------------------------------
#set module path
mpath="//lib/openser/modules/"

loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"
loadmodule "lcr.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"

# ----------------- setting module-specific parameters ---------------

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|usrloc","db_url","mysql://
openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")

# -------------------------  request routing logic -------------------

# main routing logic

route{
```

```
    #
    # -- 1 -- Request Validation
    #
    if (!mf_process_maxfwd_header("10")) {
           sl_send_reply("483","Too Many Hops");
           exit;
    };
    if (msg:len >=  2048 ) {
           sl_send_reply("513", "Message too big");
           exit;
    };
    #
    # -- 2 -- Routing Preprocessing
    #
    ## Record-route all except Register
    if (!method=="REGISTER") record_route();
    ##Loose_route packets
    if (loose_route()) {
           # marca a logica de roteamento no pedido
           append_hf("P-hint: roteado por loose_route\r\n");
           route(1);
    };
    #
    # -- 3 -- Determine Request Target
    #
    if (method=="REGISTER") {
           route(2);
    } else {
           route(3);
    };
}

route[1] {
    #
    # -- 4 -- Forward request to target
    #
    ## Forward statefully
    t_on_failure("1");
    if (!t_relay()) {
           sl_reply_error();
    };
    exit;
}
```

```
route[2] {
    ## Register request handler
    if (is_uri_host_local()) {
           if (!www_authorize("", "subscriber")) {
                  www_challenge("", "1");
                  exit;
           };
           if (!check_to()) {
                  sl_send_reply("403", "Forbidden");
                  exit;
           };
           save("location");
           exit;
    } else if {
           sl_send_reply("403", "Forbidden");
    };
}
route[3] {
    ## INVITE request handler
    if (is_from_local()){
        # From an internal domain -> check the credentials and the FROM
         if(!allow_trusted()){
              if (!proxy_authorize("","subscriber")) {
                   proxy_challenge("","1");
                   exit;
              } else if (!check_from()) {
                   sl_send_reply("403", "Forbidden, use From=ID");
                   exit;
              };
         } else {
              log("Request bypassed the auth.using allow_trusted");
         };
        consume_credentials();
          #Verify aliases, if found replace R-URI.
          lookup("aliases");
          if (is_uri_host_local()) {
                # -- Inbound to Inbound
                route(10);
        } else {
                # -- Inbound to outbound
                route(11);
        };
    } else {
           #From an external domain ->do not check credentials
           #Verify aliases, if found replace R-URI.
           lookup("aliases");
```

```
            if (is_uri_host_local()) {
                    #-- Outbound to inbound
                    route(12);
            } else {
                    # -- Outbound to outbound
                    route(13);
            };
    };
}
route[4] {
        # routing to the public network
    if (!load_gws()) {
         sl_send_reply("503", "Unable to load gateways");
         exit;
        }
        if(!next_gw()){
         sl_send_reply("503", "Unable to find a gateway");
         exit;
    }
    route(1);
    exit;
}
route[10] {
     #from an internal domain -> inbound
     #Native SIP destinations are handled using the location table
     #Gateway destinations are handled by regular expressions
     #In our example we will normalize the number to e164 +1305XXXXXX
     #to facilitate the posterior billing.
     append_hf("P-hint: inbound->inbound \r\n");
     if (uri=~"^sip:[2-9][0-9]{6}@") {
          if (is_user_in("credentials","local")) {
              # Assuming your country is USA (+1) and area code (305)
              prefix("+1305");
              route(4);
              exit;
          } else {
              sl_send_reply("403", "No permissions for local calls");
              exit;
          };
     };
     if (uri=~"^sip:1[2-9][0-9]{9}@") {
          if (is_user_in("credentials","ld")) {
               prefix("+");
               route(4);
               exit;
          } else {
             sl_send_reply("403", "No permissions for long distance");
```

```
                    exit;
            };
        };

        if (uri=~"^sip:011[0-9]*@") {
            if (is_user_in("credentials","int")) {
                strip(2);
                prefix("+");
                route(4);
                exit;
            } else {
                sl_send_reply("403", "No permissions for
                                    international calls");
            };
        };

        if (!lookup("location")) {
            sl_send_reply("404", "Not Found");
            exit;
        };
        route(1);
    }
    route[11] {
        # from an internal domain -> outbound
        # Simply route the call outbound using DNS search
        append_hf("P-hint: inbound->outbound \r\n");
        route(1);
    }
    route[12] {
        # From an external domain -> inbound
        # Verify aliases, if found replace R-URI.
        lookup("aliases");
        if (!lookup("location")) {
            sl_send_reply("404", "Not Found");
            exit;
        };
        route(1);
    }
    route[13] {
        #From an external domain outbound
        #we are not accepting these calls
        append_hf("P-hint: outbound->inbound \r\n");
        sl_send_reply("403", "Forbidden");
        exit;
    }
    failure_route[1] {
       if(!next_gw()) {
         t_reply("503", "Service not available, no more gateways");
             exit;
```

```
}
t_on_failure("1");
t_relay();
```

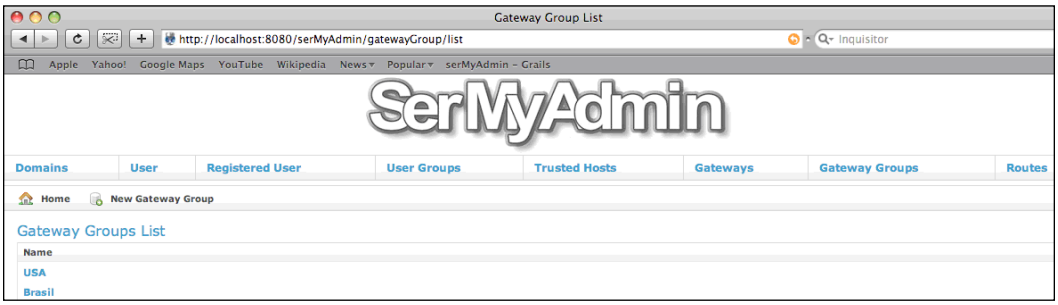
Step 3: Use `ngrep` to capture the packets and certify that the packets are going to the right destinations.

Step 4: Add the routes and gateways according to the table below, using "openserctl lcr" commands.

lcr Gateway Groups

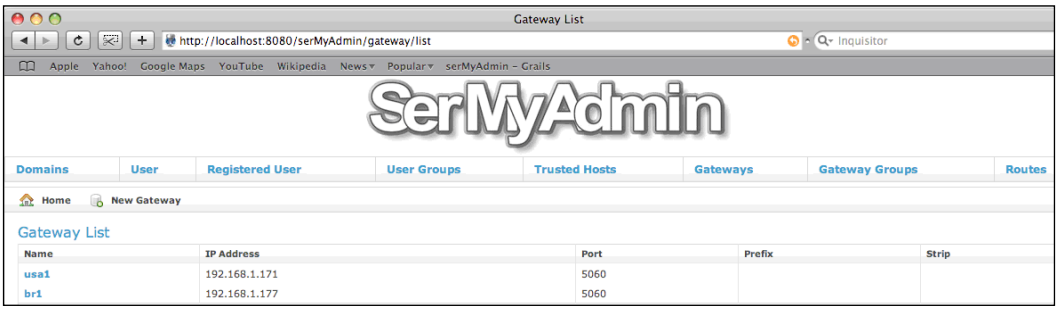
grp_id	grp_name
1	Usa
2	Br

You can use SerMyAdmin to insert gateway groups.



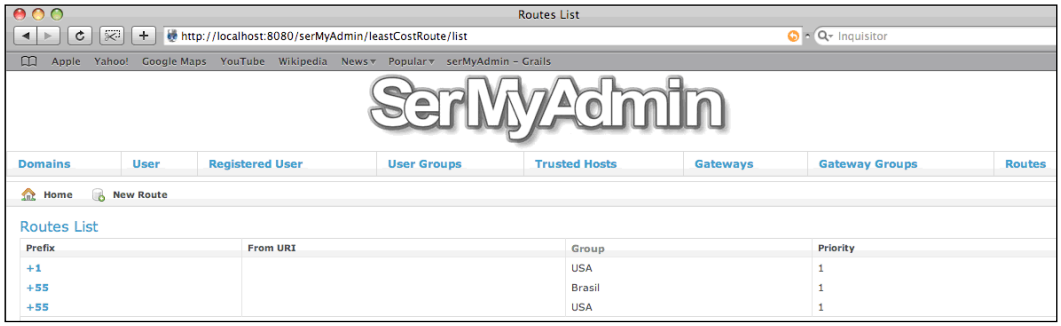
lcr Gateways

gw_name	Ip	port	uri_scheme	transport	grp_id	strip	Prefix
usa1	192.168.1.171	5060	1	1	1	0	
br1	192.168.1.177	5060	1	1	2	0	



lcr Routes

Prefix	From_uri	Grp_id	Priority
+1		1	1
+55		2	1
+55		1	2



Step 5: Test the calls to any number starting with +1, or +55.

Step 6: Turn off the gateway br1 and test the call to +55 again. The call should go to the alternative gateway, now the br1 is turned off.

Securing re-INVITES

Now that we are connected to the PSTN it is important to take care of some security considerations. Re-INVITES are being processed under the loose route section. These re-INVITES are not being challenged for its credentials. To enhance the security add the script below to your `loose_route` section. If the request is sequential (`has_totag()`) it need to have a ROUTE header. If it does not have (checked by the function `loose_route()`) we will discard the request with an error type "404, Not Here". Check the file openser.chapter7-3 if you have any doubt.

```
if (has_totag()) {
          # sequential request withing a dialog should
          # take the path determined by record-routing
          if (loose_route()) {
               #Check authentication of re-invites
               if(method=="INVITE" && (!allow_trusted())) {
                       if (!proxy_authorize("","subscriber")) {
                       proxy_challenge("","1");
                       exit;
                   } else if (!check_from()) {
```

```
                        sl_send_reply("403", "Forbidden, use From=ID");
                        exit;
                    };
                    };
                    route(1);
            } else {
                  sl_send_reply("404","Not here");
            }
            exit;
}
```

Blacklists and "473/Filtered Destination" messages

The DNS blacklist is a feature used for DNS Failover. If you send a call to a gateway and this gateway is not accessible or is responding with a response code type 5xx or 6xx, OpenSER uses a resource called "dns blacklist" and insert your gateway in the blacklist. Your gateway will stay in the blacklist for 4 minutes (may be changed in compile time in blacklists.h) , before you can send traffic to it again. While the gateway is in the blacklist, you will receive the message "473/Filtered Destination", if you try to send calls to this specific gateway. To disable this feature use (it is enabled by default):

```
disable_dns_blacklist=yes
```

You can also create your own lists to blacklist gateway permanently or temporarily out of service.

Summary

In this chapter you have learned how to configure OpenSER to forward calls to a gateway. It is important to take care of the security. Using the permissions modules you can allow the gateways to bypass the digest authentication and permit them validating only its IP addresses. The group module is important for controlling the access from the UACs. It is interesting also to validate re-INVITES for their credentials. From the point you connect to the PSTN, take a lot of care about toll fraud. I recommend you to have a security specialist verifying your environment periodically. Often, analyze your call detail records to detect abnormal activity.

8
Call Forward and Voice Mail

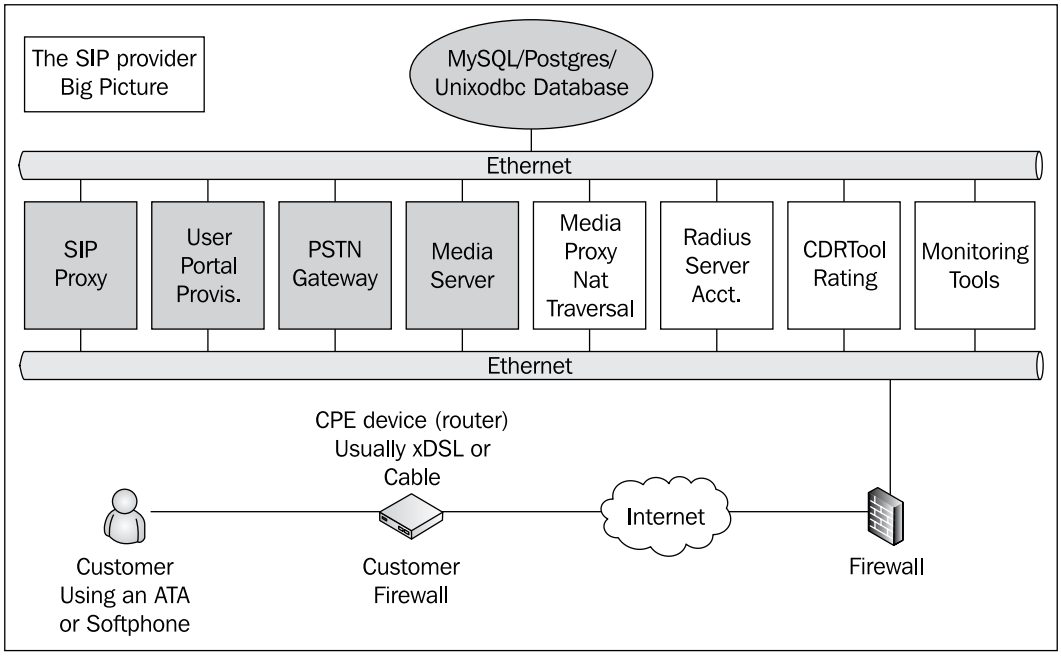
The call forwarding feature is an important feature for VoIP providers. It is implemented using forking or redirection. When you use forking, a new call leg is created, sending a new INVITE to the new destination after the first destination fails (busy or timeout). Using redirection, the proxy sends an answer to the call originator and gives it the address where it should redirect the call.

By the end of this chapter you will be able to:

- Describe concepts such as forking and redirection
- Implement call forwarding
- Implement call forward on busy
- Use the AVP resource to store call forward data
- Use the failure route to forward unanswered and busy calls

We will use just forking on this material, because it is safer than redirect and will allow us to bill the calls. The redirection method is almost useless for VoIP providers, because the proxy stays out of the signaling path and billing in this situation is not possible.

Let's verify our progress. The VoIP provider solution has many components. To avoid loosing perspective, we will show the picture overleaf in every chapter. In this chapter, we are working with the Media Server component (Asterisk Voice Mail). There are many other media servers available such as SEMS (SIP Express Media Server `www.iptel.org/sems`), Yate, FreeSwitch, and others. We have chosen Asterisk because of its popularity. The call forwarding feature is important to send the calls to the media server in several situations. The situation covered in this chapter is voicemail. The media server might be used for several applications such as IVRs, to play prompts, text to speech, and voice recognition. Remember that the SIP Proxy never handles the media, so you will need a Media Server for these situations.



After this chapter our VoIP provider will be able to send unanswered and busy calls to the voicemail server using serial forking.

Call Forwarding

In this chapter, we will implement three kinds of call forward. This forwarding is important for voicemail operations.

- **Blind call forwarding**—All INVITE requests sent to this phone number will be redirected immediately to the phone number stored in the `user_preferences` table. The SIP router will fork the call creating a new leg to the new destination. The phone with call forward configured won't even ring, registered or not.
- **Forward on busy**—In this case, we will use the `failure_route` feature to intercept the "486 Busy" message and create a new leg sending the INVITE request to the final destination.
- **Forward on no answer**—If a phone replies to an INVITE request with a "408 Request Timeout", OpenSER again will use the `failure_route` feature to intercept the message and create a new leg sending the INVITE to the final destination.

All call forwarding destinations are stored in the table `user_preferences`.
We will introduce new concepts in this chapter such as AVPs (Attribute-Value Pairs) and pseudo-variables. AVPs are made available by the OpenSER core. The AVPOPS (Attribute-Value Pairs Operations) module provides several functions for manipulating AVPs, such as interaction with SIP requests and the database, operations with strings, and operations with regular expressions.

Pseudo-Variables

Pseudo-variables are system variables that you can use in your script as parameters or inside functions. These variables are replaced by the corresponding values before the script execution. Some modules can receive pseudo-variables, such as:

- ACC
- AVPOPS
- TEXTOPS
- UAC
- XLOG

The name of a pseudo-variable always starts with $. If you want to use the $ character in your script you will have to escape it with $$. There is a pre-defined set of pseudo-variables. OpenSER pseudo variables used with OpenSER 1.1:

$ar	Auth realm
$au	Auth username
$br	Request's first branch
$bR	All Request's branches
$ci	Call-ID header
$cl	Content length
$cs	Cseq
$ct	Contact Header
$cT	Content Type
$dd	Domain of destination URI
$di	Diversion header URI
$dp	Port of destination URI
$dP	Transport protocol of destination URI
$ds	Destination set
$du	Destination URI
$fd	From URI domain

$fn	From display name
$ft	From Tag
$fu	From URI
$fU	From URI username
$mb	SIP message buffer
$mf	Flags
$mF	Flags in hexadecimal
$mi	SIP message ID
$ml	SIP message length
$od	Domain in SIP request's original URI
$op	Port of Sip request's original URI
$oP	Transport protocol of SIP request's original URI
$ou	Request's original URI
$oU	Username in SIP request's original URI
$pp	Process id
$rd	Domain in SIP request's URI
$rb	Body of request/reply
$rc	Returned code
$rm	SIP request's method
$rp	SIP request's port of R-URI
$rP	Transport protocol of SIP request URI
$rr	SIP reply reason
$rs	SIP reply status
$rt	Refer-to URI
$ru	SIP request's URI
$rU	Username in SIP request's URI
$Ri	Received IP address
$Rp	Received Port
$si	IP source address
$sp	Source port
$td	To URI domain
$tn	To display name
$tt	To tag
$tu	To URI

$tU	To URI username
$Tf	String formatted time
$Ts	Unix time stamp
$ua	User agent header
$re	Remote-Party-ID header URI

AVP (Attribute-Value Pair) Overview

Operations with attribute-value pairs permit the access and manipulation of user preferences. An AVP can be seen as a value associated to an identifier (string or integer). In the OpenSER processing, an AVP is tied to a transaction. The AVP is allocated when the transaction begins and unallocated when finished.

The introduction of AVPs in the OpenSER processing created several new possibilities for service implementation and user preference processing per user or domain. The AVPs can be used directly in the configuration scripts and to load data from a MySQL database.

An attribute-value pair is referenced in a way very similar to a variable.

```
$avp(id[N])
```

Where ID is:

- `si:name`—AVP identifier name. "s" and "i" specify the string or integer.
- `name`—The name of an alias AVP. It can be a string or integer.

Examples:

```
$avp(i:700)
$avp(s:blacklist)
```

For those who know Asterisk, the AVPOPS moduleis to OpenSER what AstDB functions are for Asterisk. However, the implementation is quite different and AVPs are much more powerful, allowing advanced features such as queries in a database and pushing data directly to the SIP packet. There are a lot of functions associated with the AVPs:

- `avp_db_load`: Loads AVPs from the database to the memory
- `avp_db_store`: Store AVPs into the database
- `avp_db_delete`: Delete AVPs from database
- `avp_db_query`: Make a database query and store the results in AVP
- `avp_delete`: Delete AVPs from memory

- `avp_pushto`: Push the AVP values into the SIP message
- `avp_check`: Check the value of the AVP using an operator (equal, greater than, etc.) and a value
- `avp_copy`: Copy an AVP to another
- `avp_printf`: Format a string to an AVP
- `avp_subst`: Find and replace values into an AVP
- `avp_op`: Allows math operations on AVPs
- `is_avp_set`: Check if this AVP name is set
- `avp_print`: Print all the AVPs in memory (for debugging purposes)

You can check the syntax for these functions in the documentation. For now, we have to understand how to use `avp_db_load` and `avp_pushto`, which will be used on our script. There is an excellent tutorial about AVPs at: `http://www.voice-system.ro/docs`.

AVPs are not exactly simple. But if you think of them as simple pairs of attributes and values, they are not so complex too. However, the loading of AVPs from the database is very confusing. The default table is `usr_preference` (user preferences). Sometimes the value that we want is not associated to a specific user, but to a domain. Anyway, all AVPs being loaded from a database come from the `usr_preference` table.

Example: For call forward, we have a call forward associated to user. It is actually a `usr_preference`. Let's check the `usr_preference` table structure.

id	uuid	username	domain	attribute	type	value	Last_ modified
		1001		callfwd	0	sip:1004@yourdomain	

The `id` is an auto-increment field.

- `uuid` is a unique user identifier
- `username` for username
- `domain` for domain
- `attribute` (the AVP name)
- `type`
  (0–AVP str | Val Str, 1–AVP str | Val Int, 2–AVP int | Val Str, 3-AVP int | Val int)
- `value` (the AVP value)
- `last modified` (the date of the last modification)

The AVPs can be associated to a user or to a domain. So you can load the AVPs associated with either of these parameters. You can associate an AVP to a uuid (unique user ID), to a username (single domain setup), or to username and domain (multi-domain) setups.

In the function `avp_db_load` the first parameter is the source and the second is the `avp_name`. So, the function below will load in the string AVP `callfwd` the value of the attribute `callfwd` for the user matching the requested URI (`$ruri`) in the column username.

```
(avp_db_load("$ruri/username","s:callfwd")
```

Later on we will push this AVP to the SIP packet changing the original `$ruri` to the new one.

```
avp_pushto("$ruri","s:callfwd");
```

In other words, if a call forward number is set for this user, instead of calling the original user, we will call the user stored in the AVP `s:callfwd`. The magic for the call forward is to insert the call forward numbers in the `usr_preference` table.

AVPOPS Module Loading and Parameters

In the module load, we specify the database location, access parameters, and the AVP table.

```
loadmodule "/usr/lib/openser/modules/avpops.so"
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/
openser")
modparam("avpops", "avp_table", "usr_preferences")
```

Implementing Blind Call Forwarding

In the first place let's implement the blind call forward service. In the INVITE processing we will load the AVP named `callfwd` from the user preference table in the database. If the `callfwd` preference was set for this specific user, we will push it to the R-URI before forwarding the request.

```
if(avp_db_load("$ruri/username","s:callfwd")){
  #Check the existence of the callfwd attribute on the
  #usr_preferences table. If found, assign the vaue to the AVP
  # and push the value to the ruri of the SIP header.
  avp_pushto("$ruri","s:callfwd");
  route(1);
  exit;
};
```

To make this feature work, it is important to insert the right entries in the database. The table used by AVPs is the `usr_preferences`.

Username	Type	Attribute	Value
1001	0	callfwd	`sip:1004@yourdomain`

You can modify the user preferences with the help of SerMyAdmin; just browse to **user preferences** on the menu. There you can view all user preferences, edit them, and create new ones.

If you are working in a multi-domain environment, please enable the multi-domain parameter of the module AVPOPS and also populate the database with the domain name.

With the record above we are telling the system to forward any call to 1001 to the URI `sip:1004@yourdomain`.

Lab—Implementing Blind Call Forwarding

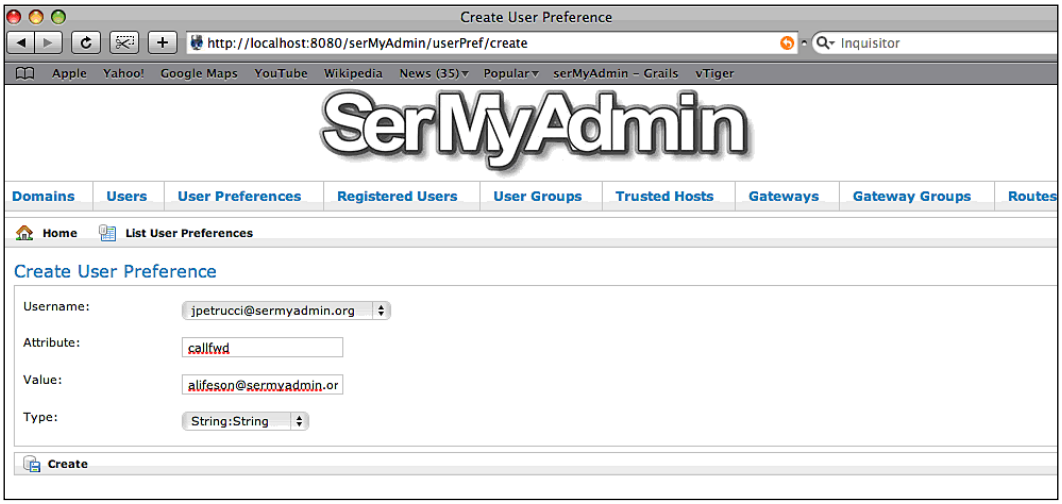
Step 1: Let's insert the AVP pairs using the SerMyAdmin interface first seen in Chapter 6.

In your browser access the SerWEB admin interface:

```
http://<your-ip-server-address>:8080/serMyAdmin
```

Step 2: Log in to the interface using a user in the "Global Administrator" role.

Step 3: Click on the **User Preferences** tab. In this menu click on the "New Preference" button and create an AVP for the user you want to forward the calls from; in this case it should be `1000@sermyadmin.org`. Its attribute must be called `callfwd` and the value will be the `URI` you want to forward the calls to; here it should be set to `1004@sermyadmin.org`.



Step 4: Edit the `openser.cfg` file to include the instructions explained above. The file should end up as below. Include the following lines in the `openser.cfg` file. Or simply copy the file `openser.callfwd1` from `http://www.sermayadmin.org/openser` to the `openser.cfg` file.

In the module loading section:

```
loadmodule "avpops.so"
loadmodule "xlog.so"
```

In the module parameters section:

```
modparam("avpops", "avp_url",
         "mysql://openser:openserrw@localhost/openser")
modparam("avpops", "avp_table", "usr_preferences")
```

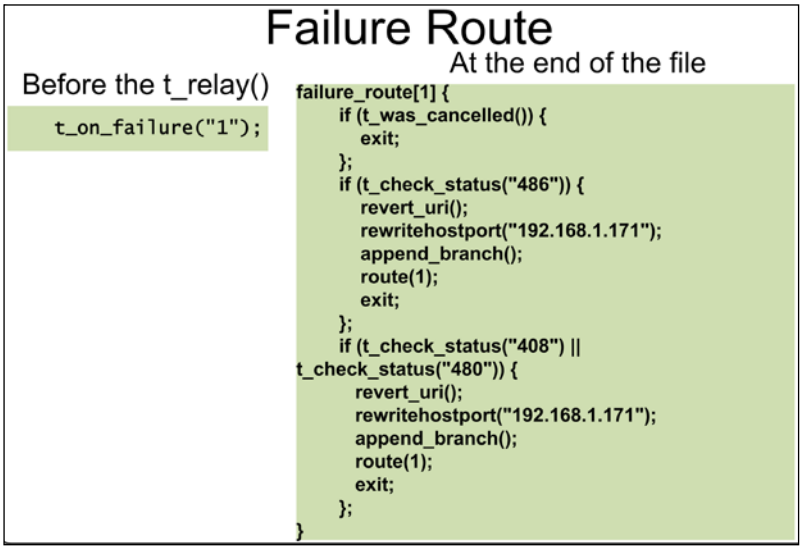
In the route[3] section:

```
route[3] {
    #
    # -- INVITE request handler --
    #
    if (is_from_local()){
```

```
# From an internal domain -> check the credentials and the FROM
 if(!allow_trusted()){
      if (!proxy_authorize("","subscriber")) {
           proxy_challenge("","1");
           exit;
      } else if (!check_from()) {
           sl_send_reply("403", "Forbidden, use From=ID");
           exit;
      };
 } else {
      log("Request bypassed the auth. using allow_trusted");
 };

if(avp_db_load("$ru/username","$avp(s:callfwd)")) {
  avp_pushto("$ru", "$avp(s:callfwd)");
  xlog("forwarded to: $avp(s:callfwd)");
  route(1);
      exit;
}
consume_credentials();
```

Step 5: Register the phones 1001 and 1004. Call from phone 1001 to phone 1000. It should forward the call to the call to phone 1004 as instructed in the `usr_preferences` table.

Implementing Call Forward on Busy or Unanswered



This is the second part of this chapter. Now we will introduce two new important concepts. The first one is the `failure_route` and the second one is the `append_branch` used to fork the call. We will handle the following failure situations:

- 408 – Timeout
- 486 – Busy Here
- 487 – Request Cancelled

To implement call forward on busy and call forward when unanswered, we will use the concept of the failure route.

In the logic below, just before sending the INVITE to the standard processing we will call the function `t_on_failure("1")`. This allows us to handle the SIP failure messages (with reply codes higher than 299, also called negative replies) in `failure_route[1]`.

When receiving a call in this situation we will forward it to a voicemail system. Asterisk can make a good voicemail system. Let's see how to integrate Asterisk to record the voicemail messages. We will prefix the URI with **b** (busy) to inform the Asterisk server to play the busy message and **u** (unanswered) to play the unanswered message. Asterisk will process the voicemail requests using the application `voicemail(b${EXTEN})` for busy messages and `voicemail(u${EXTEN})` for unanswered messages.

Below is the complete script with the changes highlighted.

```
#
#
# $Id: openser.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $
#
# simple quick-start config script
# Please refer to the Core CookBook at http://www.openser.org/
dokuwiki/doku.php
# for a explanation of possible statements, functions and parameters.
#

# ----------- global configuration parameters ------------------------

debug=3            # debug level (cmd line: -dddddddddd)
fork=yes
log_stderror=no    # (cmd line: -E)
children=4
port=5060

# ----------------- module loading ----------------------------------
```

```
#set module path
mpath="//lib/openser/modules/"

# Uncomment this if you want to use SQL database
#loadmodule "mysql.so"

loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"
loadmodule "lcr.so"
loadmodule "avpops.so"
loadmodule "xlog.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"

# ----------------- setting module-specific parameters ---------------

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|usrloc","db_url","mysql://
openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/
openser")
modparam("avpops", "avp_table", "usr_preferences")
```

```
# ------------------------  request routing logic -------------------

# main routing logic

route{

   #
   # -- 1 -- Request Validation
   #
   if (!mf_process_maxfwd_header("10")) {
          sl_send_reply("483","Too Many Hops");
          exit;
   };

   if (msg:len >=  2048 ) {
          sl_send_reply("513", "Message too big");
          exit;
   };

   #
   # -- 2 -- Routing Preprocessing
   #
   ## Record-route all except Register
   if (!method=="REGISTER") record_route();

      ##Loose_route packets
      if (has_totag()) {
         #sequential request withing a dialog should
         # take the path determined by record-routing
         if (loose_route()) {
             #Check authentication of re-invites
             if(method=="INVITE" && (!allow_trusted())) {
                 if (!proxy_authorize("","subscriber")) {
                     proxy_challenge("","1");
                     exit;
                 } else if (!check_from()) {
                     sl_send_reply("403", "Forbidden, use From=ID");
                     exit;
                 };
             };
             route(1);
         } else {
             sl_send_reply("404","Not here");
         }
```

```
            exit;
        }

        #CANCEL processing
        if (is_method("CANCEL")) {
              if (t_check_trans()) t_relay();
              exit;
        };

        t_check_trans();

    #
    # -- 3 -- Determine Request Target
    #
    if (method=="REGISTER") {
           route(2);
    } else {
           route(3);
    };
}


route[1] {
    #
    # -- 4 -- Forward request to target
    #
    ## Forward statefully
    t_on_failure("1");
    if (!t_relay()) {
           sl_reply_error();
    };
    exit;
}

route[2] {
    ## Register request handler
    if (is_uri_host_local()) {
           if (!www_authorize("", "subscriber")) {
                  www_challenge("", "1");
                  exit;
           };

           if (!check_to()) {
                  sl_send_reply("401", "Unauthorized");
```

```
                exit;
        };

        save("location");
        exit;
    } else if {
        sl_send_reply("401", "Unauthorized");
    };
}

route[3] {
    ## Non-Register request handler
    if (is_from_local()){
        # From an internal domain -> check the credentials and FROM
        if(!allow_trusted()){
            if (!proxy_authorize("","subscriber")) {
                proxy_challenge("","1");
                exit;
            } else if (!check_from()) {
                sl_send_reply("403", "Forbidden, use From=ID");
                exit;
            };
        } else {
            log("Request bypassed the auth.using allow_trusted");
        };

        if(avp_db_load("$ru/username","$avp(s:callfwd)")) {
           avp_pushto("$ru", "$avp(s:callfwd)");
           route(1);
           exit;
        }

        consume_credentials();

        #Verify aliases, if found replace R-URI.
        lookup("aliases");

        if (is_uri_host_local()) {
                # -- Inbound to Inbound
                route(10);
        } else {
                # -- Inbound to outbound
                route(11);
        };
```

```
        } else {
                #From an external domain ->do not check credentials
                #Verify aliases, if found replace R-URI.
                lookup("aliases");
                if (is_uri_host_local()) {
                        #-- Outbound to inbound
                        route(12);
                } else {
                        # -- Outbound to outbound
                        route(13);
                };
        };
    }

    route[4] {
            # routing to the public network
         if (!load_gws()) {
              sl_send_reply("503", "Unable to load gateways");
              exit;
            }

            if(!next_gw()){
              sl_send_reply("503", "Unable to find a gateway");
              exit;
         }
         route(5);
         exit;
    }

    route[5] {
        #
        # -- 4 -- T_relay for gateways
        #
        ## Forward statefully, if failure load other gateways
        t_on_failure("2");
        if (!t_relay()) {
                sl_reply_error();
        };
        exit;
    }

    route[10] {
         #from an internal domain -> inbound
         #Native SIP destinations are handled using the location table
```

```
#Gateway destinations are handled by regular expressions
append_hf("P-hint: inbound->inbound \r\n");

if (uri=~"^sip:[2-9][0-9]{6}@") {
     if (is_user_in("credentials","local")) {
         prefix("+1");
       route(4);
         exit;
     } else {
         sl_send_reply("403", "No permissions for local calls");
         exit;
     };
};

if (uri=~"^sip:1[2-9][0-9]{9}@") {
    if (is_user_in("credentials","ld")) {
       strip(1);
       prefix("+1");
    route(4);
       exit;
    } else {
       sl_send_reply("403", "No permissions for long distance");
       exit;
     };
};

if (uri=~"^sip:011[0-9]*@") {
     if (is_user_in("credentials","int")) {
         strip(3);
       prefix("+");
       route(4);
         exit;
     } else {
         sl_send_reply("403", "No perm. for internat.calls");
     };
};

if (!lookup("location")) {
     if (does_uri_exist()) {
         ## User not registered at this time.
         ## Use the IP Address of your e-mail server
         revert_uri();
         prefix("u");
         rewritehostport("192.168.1.171"); #Use the voicemail IP
```

```
                route(1);
            } else {
                sl_send_reply("404", "Not Found");
             exit;
            }
            sl_send_reply("404", "Not Found");
            exit;
      };
      route(1);
}

route[11] {
      # from an internal domain -> outbound
      # Simply route the call outbound using DNS search
      append_hf("P-hint: inbound->outbound \r\n");
      route(1);
}

route[12] {
      # From an external domain -> inbound
      # Verify aliases, if found replace R-URI.
      lookup("aliases");
      if (!lookup("location")) {
           sl_send_reply("404", "Not Found");
           exit;
      };
      route(1);
}

route[13] {
      #From an external domain outbound
      #we are not accepting these calls
      append_hf("P-hint: outbound->inbound \r\n");
      sl_send_reply("403", "Forbidden");
      exit;
}

failure_route[1] {
        ##--
        ##-- If cancelled, exit.
        ##--
        if (t_check_status("487")) {
            exit;
        };
```

```
        ##--
        ##-- If busy send to the e-mail server, prefix the "b"
        ##-- character to indicate busy.
        ##--
        if (t_check_status("486")) {
            revert_uri();
            prefix("b");
            xlog("L_ERR","Stepped into the 486 ruri=<$ru>");
            rewritehostport("192.168.1.171");
            append_branch();
            route(1);
            exit;
        };
        ##--
        ##-- If timeout (408) or unavailable temporarily (480),
        ##-- prefix the uri with the "u"character to indicate
        ##-- unanswered and send to the e-mail
        ##-- sever
        ##--
        if (t_check_status("408") || t_check_status("480")) {
           revert_uri();
           prefix("u");
           xlog("L_ERR","Stepped into the 480 ruri=<$ru>");
           rewritehostport("192.168.1.171");
           append_branch();
           route(1);
           exit;
        };
}

failure_route[2] {
   if(!next_gw()) {
     t_reply("503", "Service not available, no more gateways");
         exit;
   }
   t_on_failure("1");
   t_relay();
}
```

Inspecting the Configuration File

Our script is becoming very hard to debug. Now let's introduce the XLOG module. It implements the `XLOG()` function. It is very similar to the `LOG()` function, but it allows you to use pseudo-variables such as the request URI (`$ru`) inside the message. Below, there is an example of the XLOG usage.

```
loadmodule "xlog.so"
xlog("L_ERR","Marker 480 ruri=<$ru>");
```

You can check the latest XLOG messages with the command:

```
tail /var/log/syslog
```

It is important to understand that on blind call forward, only the original INVITE message will be processed; we can safely change the request URI and the call to `append_branch` does not need to be invoked.

On the other hand, for call forward on busy and call forward for unanswered calls, after the original INVITE had failed, to fork the message at this point you will have to execute the `append_branch()` function.

```
t_on_failure("1");
```

The `t_on_failure()` function tells OpenSER to handle SIP failure conditions (negative/unsuccessful replies) if these occur. Failure conditions in this context are error messages prefixed by 4xx and 5xx. When you call `t_on_failure`, just before calling the `t_relay()` function, you tell OpenSER to transfer the control to the `failure_route[1]` when it receives a failure message.

```
failure_route[1] {
       ##--
       ##-- If cancelled, exit.
       ##--
       if (t_was_cancelled()) {
           exit;
       };
```

The first part of the `failure_route` section handles cancelled messages (487). The script simply terminates the processing for this kind of message. Following this we will process busy messages.

```
       ##--
       ##-- If busy send to the e-mail server, prefix the "b"
       ##-- character to indicate busy.
       ##--
       if (t_check_status("486")) {
```

```
        revert_uri();
        prefix("b");
        xlog("L_ERR","Stepped into the 486 ruri=<$ru>");
        rewritehostport("192.168.1.171");
        append_branch();
        route(1);
        exit;
    };
```

If the status is equal to 486 (Busy here) the action is to revert the URI (486 is a failure message in the inverse direction to the INVITE request), prefix the URI with "b" (indicating to the voicemail system to play the busy message) and rewrite the host to send the message to the voicemail. `append_branch()` is called to add the destination to the request. The same logic is applied to the messages 408 and 480.

```
##-- If timeout (408) or unavailable temporarily (480),
##-- prefix the uri with the "u"character to indicate
##-- unanswered and send to the e-mail
##-- sever
##--
if (t_check_status("408") || t_check_status("480")) {
   revert_uri();
   prefix("u");
   xlog("L_ERR","Stepped into the 480 ruri=<$ru>");
   rewritehostport("192.168.1.171");
   append_branch();
   route(1);
   exit;
};
```

On the Asterisk server the file `extensions.conf` should have the following instructions. The voicemail accounts should be created to match the account on OpenSER. You can integrate both databases to avoid keeping duplicate database entries using the tutorial found at `http://www.voip-info.org/wiki/view/Realtime+Integration+Of+Asterisk+With+OpenSER`.

```
#extensions.conf file
[default]
exten=>_9.,1,Dial(ZAP/g1/${EXTEN})
exten=>_9.,2,hangup()
exten=>_u.,1,Voicemail(u${EXTEN})
exten=>_u.,2,hangup()
exten=>_b.,1,Voicemail(b${EXTEN})
exten=>_b.,2,hangup()
```

Lab—Testing the Call Forward Feature

To create this lab, some experience with Asterisk is required for the voicemail integration. This lab is relatively hard to implement. Some IP phones hardly ever send the busy message, because they have more than a single line. It is important to use all the lines before to get the "486 busy" message. There is an IP phone from SNOM that has a wonderful busy button to indicate to the user that the phone is busy. We will reduce the INVITE timeout to make the tests easier and less cumbersome. On production environments remove these instructions.

```
modparam("tm", "fr_timer", 5)
modparam("tm", "fr_inv_timer", 20)
```

Step 1: Test the call forward for unanswered calls.

Call from extension 1000 to extension 1002. The call should go to the voicemail system, with the unavailable message.

Step 2: Test the call forward on busy.

Take extension 1003 off-hook. Call extension 1003 from extension 1000. It should go to the voicemail system with the busy message.

Summary

In this chapter, we have learned how to use AVPs to handle user preferences such as call forward. Using `failure_route` allows us to implement two common situations, call forward on busy and call forward on no answer. Finally we have learned how to send this kind of message to an external voicemail system such as an Asterisk server.

9
SIP NAT Traversal

NAT, also known as Network Address Translation, was the solution found to solve the shortage of IP addresses forecast in mid 90s. The solution consisted of using a small range of IP addresses (in most cases a single IP addresses) on the outside port of the firewall and a range of reserved addresses (non-registered addresses defined in RFC1918) on the inside port of the firewall.

Unfortunately, NAT breaks SIP communication. In this chapter, we will explain some ways to solve the NAT traversal challenge.

By the end of this chapter you will be able to:

- Explain why NAT breaks SIP communication
- Describe the different NAT types and their implications
- Describe the main mechanisms available for NAT traversal
- Implement a NAT traversal mechanism called TURN
- Install and configure the MediaProxy server and its related modules

NAT is usually implemented on routers and firewalls. The NAT router maps the internal address to an external address keeping an address mapping table. Sometimes NAT is also referred as PAT (Port Address Translation). PAT maintains a mapping table of "ip:port" pairs allowing a single external address to be used by several internal addresses. You can search for more information in RFC1631.

RFC1918 defines the address allocation for private networks. The private address space can be defined as the blocks below:

- 10.0.0.0 to 10.255.255.255 (10/8 prefix)
- 172.16.0.0 to 172.31.255.255 (172.16/12 prefix)
- 192.168.0.0 to 192.168.255.255 (192.168/16 prefix)

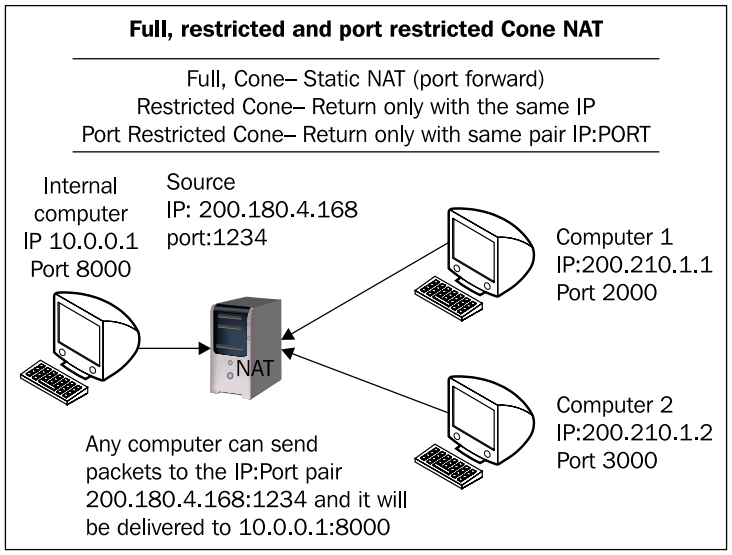
NAT Types

There are four kinds of NAT:

- Full Cone
- Restricted Cone
- Port Restricted Cone
- Symmetric

Full Cone

The first type of NAT, full cone, represents a static mapping from an external "ip:port" pair to an internal "ip:port" pair. Any external computer can connect to it using the external "ip:port" pair. This is the case in non-stateful firewalls implemented with the use of filters.



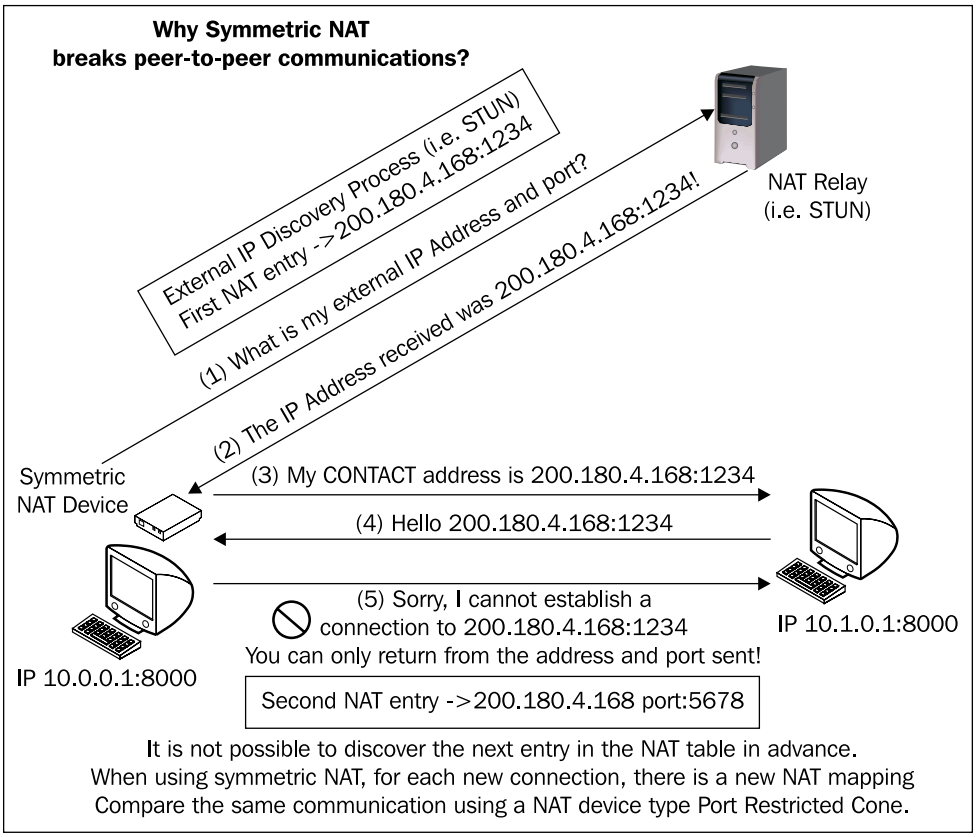
Restricted Cone

In the restricted cone scenario, the external "ip:port" pair is opened only when the internal computer sends data to an outside address. However, the restricted cone NAT blocks any incoming packets from a different address. In other words, the internal computer has to send data to an external computer before it can send data back.

Port Restricted Cone

The port restricted cone firewall is almost identical to the restricted cone. The only difference is that the incoming packet should come from exactly the same "ip:port" pair as the destination of the sent packet.

Symmetric



The last type of NAT is called symmetric. It is different from the first three in that a specific mapping is done to each external address. Only specific external addresses are allowed to come back by the NAT mapping. It is not possible to predict the external "ip:port" pair that will be used by the NAT device. With the other three types of NAT, it was possible to use an external server to discover the external IP address to use for communication. With symmetric NAT, even if you can connect to an external server, the discovered address cannot be used for any other device except for this server.

NAT Firewall Table

Below is a summary table of all four NAT types. This table is very useful to understand the differences between the various types of NAT.

NAT Type	Need to send data before receiving	It is possible to determine the ip:port pair for returning packets	It restricts the incoming packets to the destination ip:port
Full Cone	No	Yes	No
Restricted Cone	Yes	Yes	Only IP
Port Restricted Cone	Yes	Yes	Yes
Symmetric	Yes	No	Yes

Solving the SIP NAT Traversal Challenge

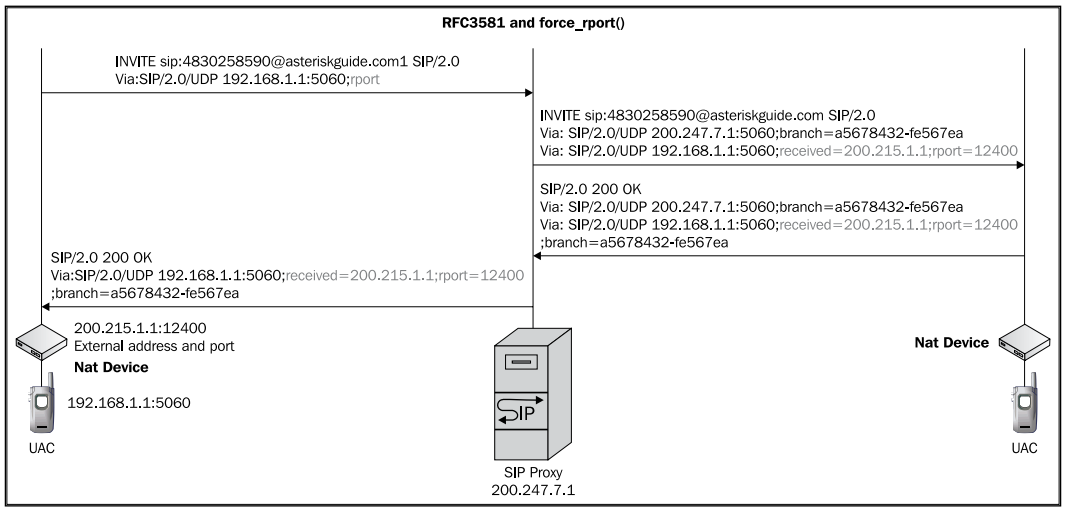
The solutions for NAT traversal can be classified as **near-end** for solutions implemented in the client side and **far-end** for solutions implemented on the server side. The far-end solutions are easier to manage and solve NAT traversal in all four types of NAT devices. However, these solutions impose a scalability penalty forcing the RTP media flow into media servers. Near-end solutions are harder to manage and can solve just the first three types of NAT devices but are far more scalable. The ideal solution is to use near-end NAT traversal solutions to every device that supports it (Symmetrical NAT devices don't) and use far-end NAT traversal for the SIP devices behind a symmetric firewall. We will start using a far-end NAT solution using the MediaProxy server.

Implementing a Far-End NAT Solution

Now let's examine a solution to implement NAT traversal on the servers without having to make any special configuration to the clients. The UAC has to be symmetric, in other words to send and receive on the same UDP port for both SIP (5060) and RTP (usually, in the range of 10000 to 20000). Most user agent clients as of today are symmetric, so it is rare to find an asymmetric client.

The SIP NAT traversal problem can be classified into two parts. The SIP protocol itself and the RTP protocol that carries the media. We will use a set of techniques to handle the SIP and the RTP protocols. We will use RFC3581 to traverse the SIP packets, together with some message treatment and the a Media Relay server known as "MediaProxy" developed by Adrian Georgescu (`www.ag-projects.com`) and released with a GPL license. This kind of solution is also known by the acronym TURN (traversal of UDP over relay NAT). There is another TURN solution for OpenSER known as RTPproxy.

RFC3581 and the force_rport() Function

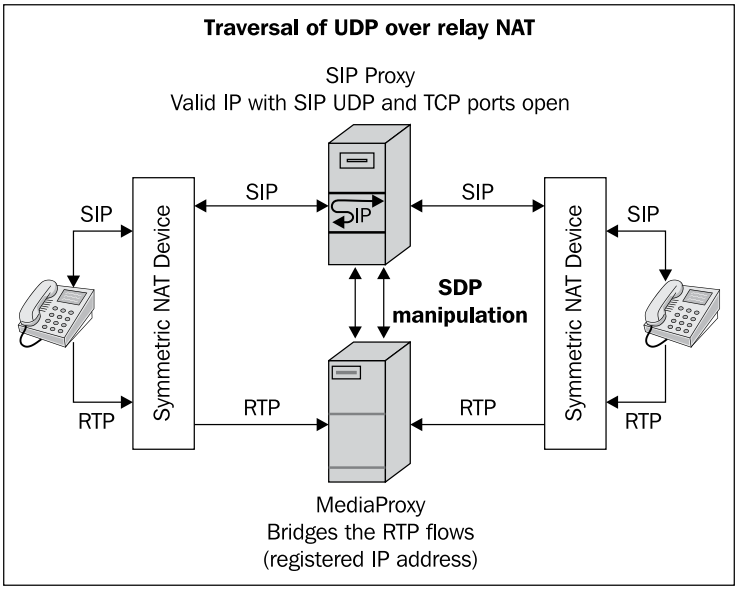


The traversal of the SIP signaling can be easily solved using the fields named "**rport**" and "**received**" as stated in RFC3581.

When a client is compatible with RFC3581, it inserts a parameter named **rport** in the VIA header field of the SIP request. The Proxy will verify the existence of the field, and will include the fields **received=** and **rport=** with the address from the interface where it received packet. With this, it is now very easy for the proxy to forward the responses to the SIP devices. In our case, we will force the "rport" even if the client does not support it. RFC3581 solves the problem with dialing, but not the reception of calls.

Solving the Traversal of RTP Packets

The SIP problem was solved with these solutions. This was possible because the SIP protocol uses a single and previously known port (UDP 5060). On the other hand RTP uses dynamic UDP ports described in the SDP (Session Description Protocol). When using symmetric NAT devices, a new mapping will be created to each destination. So it is not possible to inform the UAC ahead of the correct UDP port from which it received the RTP packet in order to send the response. The solution is to send both RTP sessions directly to device called MediaProxy with a known IP address and port and bridge the RTP flow in this box.



When you use MediaProxy, which is also known as a Media Relay server, it bridges the media flows (RTP) coming from the UACs. For now it is the only way available to traverse a symmetric NAT device. The script will change the SDP addresses to force the RTP packets over MediaProxy using the function `use_media_proxy()` from the MediaProxy module.

MediaProxy, developed by AG Projects (`www.ag-projects.com`) is not the only solution for the Media Relay server component. You can also use, another open-source solution developed by PortaOne (`www.portaone.com`) called RTPproxy. RTPproxy is being actively maintained by Sippy Software Inc. (`www.sippysoft.com`), where you can find the software for download. There are some devices called Session Border Controllers (SBC) that can also be used for this purpose. Some are very sophisticated and scalable such as the ones available from Acme Packets, Newport Networks, and Voice System, the last one based on OpenSER (VNT-10).

They can bridge thousands of calls, but not without a price tag. This price might achieve a few hundreds of thousands of dollars. If you are planning to be a huge service provider, I recommend you to check SBC solutions carefully.

I have chosen the MediaProxy software for this material because it is able to distribute load and thus be scalable to thousands of users. It can be integrated with the accounting module to produce more precise results. The only drawback is the fact that it is written in Python and supports a limited number of simultaneous sessions. RTPproxy may handle, in some cases, more than 900 calls in a single computer. In several situations, RTPproxy can be a better choice. We will not cover RTPproxy in this material, but you can find a compatible example file at `www.sermyadmin.org/openser.rtpproxy`.

The INSTALL file of MediaProxy states that it can handle 80 sessions per GHz of processing power.

Handling REGISTER Requests behind NAT

Now that we have learned how to use `force_rport()` to allow the return of the message to the right IP address and UDP port, let's see how to handle REGISTER requests and allow the UAC to receive calls behind a NAT device. When a UAC sends a REGISTER request we will do the following actions:

1. Test if the UAC is behind NAT (this can be done using `client_nat_test()`) and if the request is not to unregister all contacts (using * in the contact header field).
2. Force the use of the rport field to reply to the right IP address and UDP port back to the UAC.
3. If the UAC is behind NAT we will fix the REGISTER using `fix_nated_register()` to request the insertion of the received IP address and UDP port into the location table.

4. Keep the NAT mappings open using the `natping` parameter of the MediaProxy module after the first registration. This parameter is used to set the interval between sending UDP packets, to keep the NAT mappings open. Ping is used to test the connectivity using the protocol ICMP (Internet Control Message Protocol). Natping uses the same concept of connectivity testing, but uses a UDP packet instead. As most of you know, NAT has a timeout, where it deletes an entry in the mapping table. Natping helps us to avoid the NAT timeout, keeping the NAT table always refreshed.
5. Set a specific flag (six in our example) to be saved in the location table. This will help later to determine if a destination is behind NAT.

Sample code to handle REGISTER requests:

```
if (!search("^Contact:[ ]*\*") && client_nat_test("7")) {
        setflag(6);
        fix_nated_register();
        force_rport();
};
```

Determining if the Client is behind NAT

There are two functions that could be used to test if a client is behind a NAT device. One is exported by the `nathelper` module (part of the RTPproxy solution developed by PortaOne) and the other exported by the `mediaproxy` module (part of the MediaProxy solution developed by AG Projects). Note that the first three tests from both modules are very similar.

- Function `client_nat_test()`
    - Exported by the MEDIAPROXY module
    - Parameters:
        - "1" tests if the client has an RFC1918 address in the "Contact" header field.
        - "2" tests if the client has contacted OpenSER from an address that is different from the one in the VIA field.
        - "4" tests if the client has an RFC1918 address in the topmost VIA header.

- Function `nat_uac_test()`
  - Exported by the module NATHELPER
  - Parameters:
    - "1" tests if the client has an RFC1918 address in the "Contact" header field.
    - "2" tests if the client has contacted OpenSER from an address that is different from the one in the VIA field.
    - "4" tests if the client has an RFC1918 address in the topmost VIA header.
    - "8" SDP is searched for RFC1918 addresses.
    - "16" tests if the source port is different from the port in the VIA header field.

You should specify the sum of the tests to be done. It will return true if at least one of the tests succeeded. If you want to perform tests "1"," 2", and "4" specify "7" as the function's parameter.

Handling INVITE Messages behind NAT

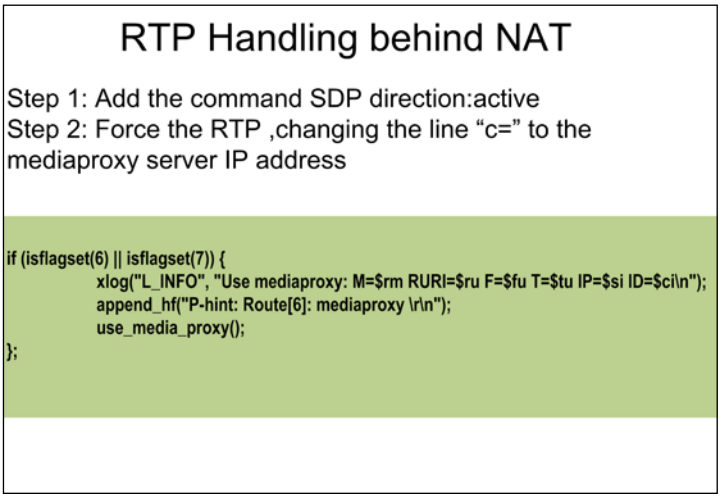
INVITE request handling behind NAT

Step 1: test if the client is behind NAT
Step 2: Fix the contact() header field
Step 3: Force the rport (received port) in the VIA header field
Step 4: Send to the route(6) for media handling

```
if (client_nat_test("3")) {
                    xlog("L_INFO", "Route[3] M=$rm RURI=$ru F=$fu T=$tu IP=$si ID=$ci\n");
                    append_hf("P-hint: Route(3)- setflag7,forcerport,fix_contact \r\n");
                    setflag(7);
                    force_rport();
                   fix_contact()
};
```

On REGISTER messages we had to handle just the SIP protocol. Now for the INVITE messages we will have to handle the SIP and the RTP protocols. To accomplish this we will have to make modifications to the SIP and the SDP headers.

Besides, the contact information is wrong; it points to the private (RFC1918) address. OpenSER should change the contact information from the private address to the public address. This is done by the function `fix_nated_contact()` exported by the `nathelper` module. Other messages, such as ACK, BYE, and CANCEL should have the CONTACT header field corrected also.



When an INVITE message is sent, it will contain an SDP payload. This SDP (Session Description Protocol) identifies the session content (audio, video, chat, named events). The SDP payload describes several things about the UAC, such as the kinds of session it supports, IP address, and UDP port where the other part can be found.

Example: A UAC describes the "ip:port" pair 192.168.0.1:23000 as the point where it wants to receive the RTP media flow. At the time of the INVITE, there is no matching "IP:port" pair in the NAT mapping table, because the RTP flow has not even started. The SDP lines describing the IP address and UDP port are shown below:

```
c=IN IP4 192.168.0.1.
m=audio 23767 RTP/AVP 0 101.
```

To handle AUDIO sessions, OpenSER will do one thing before forwarding the INVITE to the final user: Force RTP to pass over the Media Proxy changing the line c to **c=<ip-address-of-the-media-proxy> RTP/AVP 0 101.**

This option means that you need to configure a Media Relay server with a public IP address on which both users can send the RTP traffic. Thus, you can add an additional hop to the RTP connection. The additional hop will have implications on the delay and possibly jitter when calling from one phone to another. However, it is the only way to traverse symmetrical NAT available at this moment. For a VoIP provider this is not a big issue, because most calls will go to the gateway probably inside the provider. The Media Relay server will bridge the RTP sessions coming from both clients. There are two Media Relay servers available, MediaProxy from AG Projects and RTPproxy from PortaOne, both released with GPL license.

Handling the Responses

The "200 OK" message returned from the UAC will need to be manipulated too as above. Thus a NAT handling code must be included in the section `on_reply_route[]`. A flag was set (7) on the INVITE message to indicate that this transaction is to or from a client behind a NAT device. In the reply route we will check this flag and if it is set we will have to fix the contact, and have the RTP forced to Media Relay server. It is important to emphasize that the "200 OK" message contains an SDP header describing the session parameters agreed.

MediaProxy Installation and Configuration

The MediaProxy server will allow a specialized processing of clients behind NAT. The MediaProxy server from AG Projects (`www.ag-projects.com`) has the following characteristics:

- Use of the DNS SRV records to load balance the requests
- Can be executed in a separate server offloading the SIP Proxy
- Web monitoring

The MediaProxy server is not included with OpenSER. The OpenSER distribution has only the `mediaproxy.so` module that integrates the MediaProxy server with the OpenSER server. To work, the MediaProxy server needs a public IP address. In most production environments, the MediaProxy server won't be run in the same CPU as the SIP proxy.

Installing MediaProxy

Step 1: Download the MediaProxy server from:

```
cd /usr/local
wget http://mediaproxy.ag-projects.com/mediaproxy-1.9.1.tar.gz
tar -xzvf mediaproxy-1.8.2.tar.gz
```

Newer versions are released very often, please check the current version number

Step 2: Copy the init file to `/etc/init.d` to start MediaProxy server at boot time:

```
cd /usr/local/mediaproxy/boot
cp mediaproxy.debian /etc/init.d/mediaproxy
update-rc.d mediaproxy defaults 20 90
```

Step 3: MediaProxy is configured using the `mediaproxy.ini` file:

```
cd /usr/local/mediaproxy
cp mediaproxy.ini.sample mediaproxy.ini
vi mediaproxy.ini
```

Step 4: Mediaproxy is developed in Python, so we will have to install it before running the mediaproxy.

Step 5 Edit the `mediaproxy.ini` file and remove the highlighted remarks:

```
; Configuration file for MediaProxy

[Dispatcher]
;Section for configuring the proxy dispatcher
;
;The following options are available here:
;;start    Boolean value that specifies if to start the dispatcher.
;          Default value: Yes
;
; socket   Path to the UNIX socket where the dispatcher receives
commands
;          from SER. This should match the value for mediaproxy_socket
in
;          openser.cfg. Use the keyword None to disable listening on a
;          local socket.
;          Default value: /var/run/proxydispatcher.sock
;
```

```
; listen   Network address where the dispatcher receives commands from
;          a remote Mediaproxy to close sessions for which media did
;          timeout.
;          Valid values for this are:
;            - Default
;              when using this keyword it will listen on 0.0.0.0:25061
;            - address[:port]
;              listen on the specified address and port
;              address can be an IP a hostname or the keyword Any
;              (in which case it will listen on 0.0.0.0). If address
;              is a hostname, that should map in DNS to an IP address
;              present on the machine, through an A record.
;              If port is missing assume 25061.
;
;          Default value: Default
;
; group    Put the socket in this group and make it group writable.
;          Default value: openser
;
; defaultProxy  Default mediaproxy to use in case the From/To domains
;               involved in the call don't define any.
;               Valid values for this are:
;
;               -None
;                  don't use any default proxies. domains without
;                  mediaproxy SRV records won't work
;               -/path/to/unix/socket
;                  use a single MediaProxy server identified by the
;                  given UNIX socket path
;               -IP_or_hostname[:port]
;                use a single MediaProxy server identified by its
;                network address. The network address consists of an
;                IP address or a hostname and an optional port number
;                separated by a double colon. If port is missing 25060
;                will be assumed.
;                    Examples:
;                       10.0.0.1   (connect to 10.0.0.1 on port 25060)
;                       10.0.0.1:90 (connect to 10.0.0.1 on port 90)
;                       mp1.mydomain.com
;                       mp1.mydomain.com:7000
;               -domain://domain_name
;                Use all MediaProxies defined by domain_name,
;                honoring their priority and weight to create a
;                cluster of proxies with fallback and load balancing
```

```
;                 capabilities.
;
;                 Default value: /var/run/mediaproxy.sock
;
;start = yes
;socket = /var/run/proxydispatcher.sock
;group = openser
;defaultProxy = /var/run/mediaproxy.sock

[MediaProxy]
;
; Section for configuring the MediaProxy server
;
; The following options are available here:
;
; start    Boolean value that specifies if to start the RTP proxy
           server.
;          Default value: Yes
;
; socket   Path to the UNIX socket where MediaProxy receives commands
;          from the dispatcher or SER. Use the keyword None to disable
;          listening on a local socket.
;          Default value: /var/run/mediaproxy.sock
;
; group    Put the socket in this group and make it group writable.
;          Default value: openser
;
; listen   Network address where MediaProxy receives commands from
;          a remote dispatcher.
;          Valid values for this are:
;
;          - None
;            don't listen for network connections at all
;          - address[:port]
;            listen on the specified address and port
;            address can be an IP a hostname or the keyword Any
;            (in which case it will listen on 0.0.0.0). If address is
;            a hostname, that should map in DNS to an IP address
;            present on the machine, through an A record.
;            If port is missing assume 25060.
;
;            Default value: None
;
; allow    List of addresses that are allowed to connect to this
```

```
;          MediaProxy server and send commands.
;          They are specified as a comma separated list of entries,
;          with each entry being specified in the CIDR network/mask
           notation
;          (ex. 10.0.0.0/8)
;
;          In addition simple IP addresses or hostnames are allowed, in
;          which case the mask is considered to be 32.
;
;          In addition to network ranges/addresses 2 keywords can be
;          used for this option:
;            None   to specify that none is allowed to connect (not
;                   very useful but this is the default for security
                    reasons)
;                   Any   to specify that anyone is allowed to connect
;                  (dangerous!)
;
;          Example: allow = 10.0.0.0/24, home-pc.mydomain.com, 1.2.3.4
;
;          Default value: None
;
; proxyIP  IP address to use to talk to the phones. If not specified,
;          the first found will be used. However first found usually
;          means first defined in /etc/hosts which may not be what you
           want.
;          If you find that the address that's automatically selected
;          is not the one you want, you can specify the right one
;          using this option. The address must be one that's present
;          on one of the host's interfaces.
;
; portRange The range of ports to use for proxying the rtp streams.
;           This option is specified as minport:maxport with minport
;           and maxport being even numbers in the range 1024-65536
;           Default value: 60000:65000
;
; TOS       Mark all forwarded RTP packets with this specific TOS
            value.
;           Unless you know what TOS means, leave this option alone.
;           The TOS value can be specified either as a decimal number
;           or as a hex number in the 0xnn format.
;           Default value: 0xb8
;
; idleTimeout   Expire idle sessions after this much time.
;               Default 60 seconds
;
```

```
; holdTimeout   Expire calls on hold after this much time.
;               Default value is 3600 seconds
;
; forceClose    Forcibly close a RTP session after this many seconds
;               even if it's still active. If forceClose is 0, then a
;               session is never closed no matter how long it lasts.
;               Default value: 0
;
start = yes
socket = /var/run/mediaproxy.sock
group = openser
;listen = None
;allow = None
proxyIP = 10.0.0.1
;portRange = 60000:65000
;TOS = 0xb8
;idleTimeout = 60
;holdTimeout = 3600
;forceClose = 0

[Accounting]
; one of none, radius or database
accounting = none

[Database]
user = dbuser
password = dbpass
host = dbhost
database = radius
table = radacct

[Radius]
secret = secret
server = localhost
authport = 1812
acctport = 1813
dictionaries = /etc/radiusclient-ng/dictionary, /etc/openser/radius/
dictionary, /usr/local/mediaproxy/dictionary
retries = 2
timeout = 3
```

The proxy dispatcher section is an advanced section that you will use when you want to load balance MediaProxy servers. In our case we will only enable MediaProxy with `/var/run/mediaproxy`.

openser.cfg Analysis

Copy the file `openser.nat` (`http://www.asteriskguide.com/openser/openser.nat`) to `/etc/openser.cfg`. Let's analyze the required changes to the `openser.cfg` file.

Modules Loading

NatHelper and MediaProxy are responsible for handling the NAT traversal. Some functions such as `fix_nated_contact()`, `fix_nated_register()`, `fix_nated_sdp()`, and `nat_uac_test()` are made available by the NatHelper module. The MediaProxy module exports the functions `client_nat_test`, `fix_contact()`, `use_media_proxy()`, and `end_media_session()`.

```
loadmodule "/usr/lib/openser/modules/nathelper.so"
loadmodule "/usr/lib/openser/modules/mediaproxy.so"
```

Modules' Parameters

The modules NatHelper and MediaProxy comes from two different solutions (RTPproxy and MediaProxy). There are some redundancies between them. So we will disable some functions from NatHelper such as `natping`. We don't need two `natping` processes to be active simultaneously.

```
modparam("nathelper", "rtpproxy_disable", 1)
modparam("nathelper", "natping_interval", 0)
```

The parameter below controls the natping interval. OpenSER will send a dummy 4-byte UDP package to the phone each 30 seconds.

```
modparam("mediaproxy","natping_interval", 30)
```

OpenSER and the MediaProxy server will communicate with each other using a Unix Socket such as the specified below:

```
modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")
```

The MediaProxy server needs to know if a SIP UA is asymmetric. A UAC is asymmetric when it transmits and receives UDP packets on different ports. They are not very common these days. The MediaProxy server has special support for SIP and RTP asymmetric clients, but you will need to configure static mappings in the NAT device. Check the MediaProxy module documentation for more information regarding asymmetric clients.

It is not possible to solve the NAT problem with asymmetric UACs using the Media Relay Service.

```
modparam("mediaproxy","sip_asymmetrics","/etc/openser/sip-clients")
modparam("mediaproxy","rtp_asymmetrics","/etc/openser/rtp-clients")
```

We will use flag 6 as a NAT marker. This will allow the REGISTRAR module to store this flag in the `usrloc` table. The flag will be restored by the `lookup()` function and will indicate that the client is behind NAT.

```
modparam("registrar", "nat_flag", 6)
```

Register Message Processing

```
route[2] {
        ##--
        # Register message handling
        ##--
        sl_send_reply("100", "Trying");
        if (!search("^Contact:[ ]*\*") && client_nat_test("7")) {
          setflag(6);
          fix_nated_register();
          force_rport();
        };
```

Above, we will test if the UAC is behind NAT using `client_nat_test()`. We will check, using a regular expression, if exists a header field **CONTACT** with an * (asterisk). Some clients use **CONTACT** with an * to deregister any old registration for this client. If the client is behind NAT then we will need to set flag 6, fix the **CONTACT** header field to the external address, and force the **rport** for the client to receive the external port. The public address of the client will be saved to the user location table in memory (`usrloc`) together with flag 6 to indicate that this UAC is behind NAT.

Invite Message Processing

```
        if (client_nat_test("3")) {
                setflag(7);
                force_rport();
                fix_nated_contact();
        };
```

The INVITE messages will have a handling slightly different from the REGISTER messages. Now we will check if the UAC is behind a NAT device. If true, we will set flag 7 to mark this transaction; it will be helpful later when handling the responses.

Now we already know if the caller is behind a NAT device or not. However, we still don't know if the callee is behind NAT. Using the `lookup()` function, we will find the user in the user location tables. If the user found in USRLOC is behind NAT it has been marked with the flag 6.

```
        if (!lookup("location")) {
                sl_send_reply("404", "User Not Found");
                exit;
        };
        route(6);
        route(1);
```

If the caller or the callee is behind NAT, marked with flag 6 or flag 7 respectively, let's instruct them to use MediaProxy using the function `use_media_proxy()`

```
route[6] {
        ##--
        ## Nat traversal section
        ##--
        if (isflagset(6) || isflagset(7)) {
          use_media_proxy();
        };
}
```

Route[6] is the routing block responsible for activating the MediaProxy, whenever the caller or callee is behind NAT (flags 6 or 7 respectively).

BYE and CANCEL Message Processing

```
##--
## BYE and CANCEL message handling
##--
if (method=="BYE" || method=="CANCEL") {
     end_media_session();
};
```

At any time we can receive a BYE or a CANCEL message. We should assume that the calls have used MediaProxy. Then we need to close the related MediaProxy session, even for calls that have not used it. BYE processing occurs at the loose route section.

RE-INVITE Message Handling

If we don't handle the re-invite messages, our RTP stream could drop off during the re-invite. Now, OpenSER will embed a NAT indicator in the original Record-Route header field of the INVITE request. This header will persist across the dialog as a Route header and the system will be able to identify it later, when processing re-invites of clients behind NAT.

```
if(!is_method("REGISTER")){
   if(nat_uac_test("19")){
      record_route(";nat=yes");
   } else {
      record_route();
   };
};
```

Now let's check the `loose_route()` section where the re-invites are handled. In a re-invite message usually the route header fields are defined and the `loose_route()` function will return true. To avoid other users using `loose_route()` to establish calls, we will also check if the TO header field has a **tag=** entry indicating that this message belongs to a established call.

Now let's verify if the client is behind NAT using the function `nat_uac_test(19)` and the mark left in the previous step that the client is behind NAT using `search("^Route:.*; nat=yes")`. If true, we will set flag 6 to mark the packet for the replies, fix the CONTACT header field using `fix_contact()`, and use `media_proxy`.

```
# subsequent messages withing a dialog should take the
# path determined by record-routing
  if (loose_route()) {
    if(!has_totag()){
      sl_send_reply("403", "Initial Loose-Routing Rejected");
        exit;
    };
    if(nat_uac_test("19") || search("^Route:.*;nat=yes")){
      append_hf(P-hint: Loose-Route - fixcontact,setflag6,
                mediaproxy \r\n);
      fix_contact();
      setflag(6);
      use_media_proxy();
    };
    route(1);
  };
```

Reply Message Handling

Now we will have to handle the reply messages (200 OK, 180, 183). We will use the `onreply_route` blocks. The first thing to do is to indicate to the transactions what reply routing block you will use before calling `t_relay`. If the message could not be relayed and is an `INVITE` or `ACK` message, end the media session to free MediaProxy resources.

```
route[1] {
        t_on_reply("1");
        if (!t_relay()) {
           if (method=="INVITE" || method=="ACK") {
              end_media_session();
           };
           sl_reply_error();
        };
        exit;
}
```

Our `on_reply_route[1]` block will handle the replies generated by `route[1]`. Any messages here are a response to a previous message sent, part of a transaction. We will check the status of the reply using a regular expression and the flags indicating if the caller or the callee is behind NAT. The reply is part of a transaction, so the flags are kept until the end of the transaction. We search for the **Content-type:** and verify if it is **application/sdp** to check if the message has an SDP payload. If the message has an SDP payload we handle it using media_proxy.

```
onreply_route[1] {
  if ((isflagset(6) || isflagset(7)) && (status=~"(180)|(183)|2[0-
9][0-9]")) {
    if (search("^Content-Type:[ ]*application/sdp")) {
      use_media_proxy();
    };
  };
    if (client_nat_test("1")) {
      fix_nated_contact();
    };
}
```

Routing Script

```
#set module path
mpath="//lib/openser/modules/"

# Uncomment this if you want to use SQL database
#
# $Id: openser.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $
#
# simple quick-start config script
# Please refer to the Core CookBook at http://www.openser.org/
dokuwiki/doku.php
# for a explanation of possible statements, functions and parameters.
#

# ----------- global configuration parameters ------------------------

debug=3            # debug level (cmd line: -ddddddddddd)
fork=yes
log_stderror=no    # (cmd line: -E)
children=4
port=5060

# ------------------ module loading ----------------------------------
#set module path
mpath="//lib/openser/modules/"

# Uncomment this if you want to use SQL database
#loadmodule "mysql.so"

loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"
loadmodule "lcr.so"
loadmodule "avpops.so"
loadmodule "xlog.so"
loadmodule "nathelper.so"
loadmodule "mediaproxy.so"
```

```
# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"

# ----------------- setting module-specific parameters ---------------

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("registrar", "received_avp", "$avp(i:42)")
modparam("usrloc", "db_mode", 2)
modparam("usrloc", "nat_bflag", 4)
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|usrloc","db_url","mysql://
openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/
openser")
modparam("avpops", "avp_table", "usr_preferences")
modparam("nathelper","rtpproxy_disable", 1)
modparam("nathelper","natping_interval", 0)
modparam("nathelper","received_avp", "$avp(i:42)")
modparam("mediaproxy","natping_interval",20)
modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")
modparam("mediaproxy","sip_asymmetrics","/etc/openser/sip-clients")
modparam("mediaproxy","rtp_asymmetrics","/ect/openser/rtp-clients")

# ------------------------  request routing logic -------------------

# main routing logic

route{

    #
    # -- 1 -- Request Validation
    #
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
    };

    if (msg:len >=  2048 ) {
        sl_send_reply("513", "Message too big");
        exit;
    };

    #
    # -- 2 -- Routing Preprocessing
```

```
#
## Record-route all except Register
## Mark packets with nat=yes
## This mark will be used to identify the request in the loose
## route section
if(!is_method("REGISTER")){
    if(nat_uac_test("19")){
        record_route(";nat=yes");
    } else {
        record_route();
    };
};

##Loose_route packets
if (has_totag()) {
    #sequential request withing a dialog should
    # take the path determined by record-routing
    if (loose_route()) {
        #Check authentication of re-invites
        if(method=="INVITE" && (!allow_trusted())) {
            if (!proxy_authorize("","subscriber")) {
               proxy_challenge("","1");
               exit;
            } else if (!check_from()) {
               sl_send_reply("403", "Forbidden, use From=ID");
               exit;
            };
        };
        if(method=="BYE" || method=="CANCEL") {
            end_media_session();
        };
        ##Detect requests in the dialog behind NA, flag with 6
        if(nat_uac_test("19") || search("^Route:.*;nat=yes")){
            append_hf("P-hint: LR|fixcontact,setflag6\r\n");
            fix_contact();
            setbflag(6);
        };
        route(1);
    } else {
        sl_send_reply("404","Not here");
    }
    exit;
}

#CANCEL processing
if (is_method("CANCEL")) {
    if (t_check_trans()) {
        end_media_session();
        t_relay();
    };
```

```
        exit;
      }

    t_check_trans();
    #
    # -- 3 -- Determine Request Target
    #
    if (method=="REGISTER") {
        route(2);
    } else {
        route(3);
    };
}


route[1] {
    #
    # -- 4 -- Forward request to target
    #
    # Forward statefully
    t_on_reply("1");
    t_on_failure("1");
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

route[2] {
    ## Register request handler
    if (is_uri_host_local()) {
        if (!www_authorize("", "subscriber")) {
            www_challenge("", "1");
            exit;
        };

        if (!check_to()) {
            sl_send_reply("403", "Forbidden");
            exit;
        };

        if(!search("^Contact:[ ]*\*") && client_nat_test("7")) {
            setbflag(6);
            fix_nated_register();
            force_rport();
        };
        save("location");
        exit;

    } else if {
```

```
            sl_send_reply("403", "Forbidden");

        };
    }

    route[3] {
        ## Requests handler
        if (is_from_local()){
            # From an internal domain -> check the credentials and the
    FROM
            if(!allow_trusted()){
                if (!proxy_authorize("","subscriber")) {
                    proxy_challenge("","0");
                    exit;
                } else if(!check_from()) {
                sl_send_reply("403", "Forbidden, use From=ID");
                exit;
             };
            };

            if (client_nat_test("3")) {
                append_hf("P-hint: setflag7|forcerport|fix_contact\r\n");
                setbflag(7);
                force_rport();
                fix_contact();
            };

            #unconditional call forward
            if(avp_db_load("$ru/username","$avp(s:callfwd)")) {
                avp_pushto("$ru", "$avp(s:callfwd)");
                route(1);
                exit;
            }

            consume_credentials();

            #verify aliases, if found replace R-URI.
            lookup("aliases");

            if (is_uri_host_local()) {
                # -- Inbound to Inbound
                route(10);
            } else {
                # -- Inbound to outbound
                route(11);
            };

        } else {

            #From an external domain ->do not check credentials
```

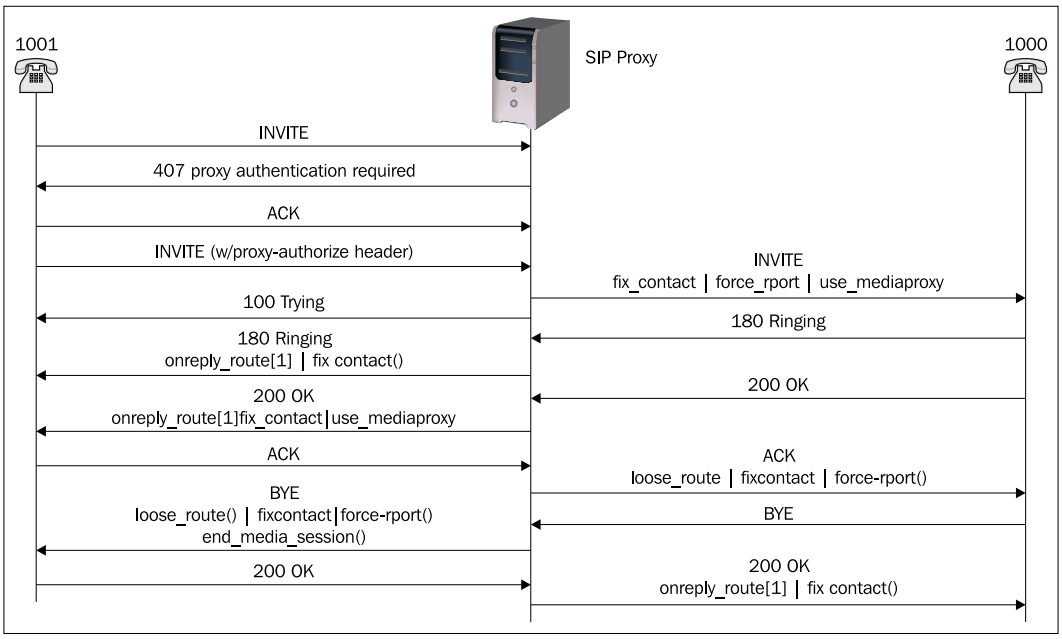
```
        #Verify aliases, if found replace R-URI.
        lookup("aliases");
            if (is_uri_host_local()) {
                #-- Outbound to inbound
                route(12);
            } else {
                # -- Outbound to outbound
                route(13);
            };
    };
}

route[4] {
    # routing to the public network
    if (!load_gws()) {
        sl_send_reply("503", "Unable to load gateways");
    exit;
    }

    if(!next_gw()){
        sl_send_reply("503", "Unable to find a gateway");
    exit;
    }
    t_on_failure("2");
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

route[6] {
    #
    # -- NAT handling --
    #
    if (isbflagset(6) || isbflagset(7)) {
        append_hf("P-hint: Route[6]: mediaproxy \r\n");
        use_media_proxy();
    };
}

route[10] {
    #from an internal domain -> inbound
    #Native SIP destinations are handled using the location table
    #Gateway destinations are handled by regular expressions
    append_hf("P-hint: inbound->inbound \r\n");

    if (uri=~"^sip:[2-9][0-9]{6}@") {
        if (is_user_in("credentials","local")) {
            prefix("+1305");
            route(6);
```

```
            route(4);
            exit;
        } else {
            sl_send_reply("403", "No permissions for local calls");
            exit;
        };
    };

    if (uri=~"^sip:1[2-9][0-9]{9}@") {
        if (is_user_in("credentials","ld")) {
            strip(1);
            prefix("+1");
            route(6);
            route(4);
            exit;
        } else {
            sl_send_reply("403", "No permissions for long distance");
            exit;
        };
    };

    if (uri=~"^sip:011[0-9]*@") {
        if (is_user_in("credentials","int")) {
            strip(3);
            prefix("+");
            route(6);
            route(4);
            exit;
        } else {
            sl_send_reply("403", "No permissions for
                          international calls");
        };
    };

    if (!lookup("location")) {
        if (does_uri_exist()) {
            ## User not registered at this time.
            ## Use the IP Address of your e-mail server
            revert_uri();
            prefix("u");
            rewritehostport("192.168.1.171"); #Use the IP address of
your voicemail server
            route(6);
         route(1);
        } else {
            sl_send_reply("404", "Not Found");
         exit;
        }
        sl_send_reply("404", "Not Found");
        exit;
```

```
    };
    route(6);
    route(1);
}

route[11] {
     # from an internal domain -> outbound
     # Simply route the call outbound using DNS search
     append_hf("P-hint: inbound->outbound \r\n");
     route(1);
}

route[12] {
     # From an external domain -> inbound
     # Verify aliases, if found replace R-URI.
     lookup("aliases");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
}

route[13] {
     #From an external domain outbound
     #we are not accepting these calls
     append_hf("P-hint: outbound->inbound \r\n");
     sl_send_reply("403", "Forbidden");
     exit;
}

failure_route[1] {
        ##--
        ##-- If cancelled, exit.
        ##--
        if (t_was_cancelled()) {
            exit;
        };
        ##--
        ##-- If busy send to the e-mail server, prefix the "b"
        ##-- character to indicate busy.
        ##--
        if (t_check_status("486")) {
            revert_uri();
            prefix("b");
            rewritehostport("192.168.1.171");
            append_branch();
            route(1);
            exit;
        };
```

```
        ##--
        ##-- If timeout (408) or unavailable temporarily (480),
        ##-- prefix the uri with the "u"character to indicate
        ##-- unanswered and send to the e-mail
        ##-- sever
        ##--
        if (t_check_status("408") || t_check_status("480")) {
           revert_uri();
           prefix("u");
           rewritehostport("192.168.1.171");
           append_branch();
           route(1);
           exit;
        };
}

failure_route[2] {
    if(!next_gw()) {
        t_reply("503", "Service not available, no more gateways");
        exit;
    };
    t_on_failure("2");
    t_relay();
}

onreply_route[1] {
#
#-- On-replay block routing --
#
    if (client_nat_test("1")) {
        append_hf("P-hint: Onreply-route - fixcontact \r\n");
        fix_contact();
    };

    if ((isbflagset(6) || isbflagset(7)) &&
                              (status=~"(180)|(183)|2[0-9][0-9]")) {
        if (search("^Content-Type:[ ]*application/sdp")) {
            append_hf("P-hint: onreply_route|usemediaproxy \r\n");
            use_media_proxy();
        };
    };
    exit;
}
```

Invite Diagram



Packet Sequence

```
U 8.8.3.80:62003 -> 8.8.3.48:5060
INVITE sip:1000@8.8.3.48 SIP/2.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;branch=z9hG4bK31390;rport.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 INVITE.
Contact: <sip:1001@192.168.0.111:5060>.
max-forwards: 70.
supported: 100rel.
```

```
user-agent: Voip Phone 1.0.
Allow: INVITE, ACK, OPTIONS, BYE, CANCEL, REFER, NOTIFY, SUBSCRIBE,
PRACK, UPDATE.
Content-Type: application/sdp.
Content-Length: 295  .
.
v=0.
o=sdp_admin 30472538 21739392 IN IP4 192.168.0.111.
s=A conversation.
c=IN IP4 192.168.0.111.
t=0 0.
m=audio 10050 RTP/AVP 0 4 18 8 101.
a=rtpmap:0 PCMU/8000.
a=rtpmap:4 G723/8000.
a=rtpmap:18 G729/8000.
a=rtpmap:8 PCMA/8000.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.
a=sendrecv.


U 8.8.3.48:5060 -> 8.8.3.80:62003
SIP/2.0 407 Proxy Authentication Required.
Via: SIP/2.0/UDP 192.168.0.111:5060;branch=z9hG4bK31390;rport=62003;re
ceived=8.8.3.80.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=50304af8547890328f8e4533797682df.822
e.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 INVITE.
Proxy-Authenticate: Digest realm=8.8.3.48, nonce=4682071833d18d2020246
78d258908fef07a049b, qop=auth.
Server: OpenSer (1.1.0-notls (i386/linux)).
Content-Length: 0.
Warning: 392 8.8.3.48:5060 Noisy feedback tells:  pid=3053 req_
src_ip=8.8.3.80 req_src_port=62003 in_uri=sip:1000@8.8.3.48 out_
uri=sip:1000@8.8.3.48 via_cnt==1.
.

U 8.8.3.80:62003 -> 8.8.3.48:5060
ACK sip:1000@8.8.3.48 SIP/2.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;branch=z9hG4bK31390;rport.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=50304af8547890328f8e4533797682df.822
e.
```

```
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 ACK.
max-forwards: 70.
Content-Length: 0.
.

U 8.8.3.80:62003 -> 8.8.3.48:5060
INVITE sip:1000@8.8.3.48 SIP/2.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;branch=z9hG4bK28696;rport.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Contact: <sip:1001@192.168.0.111:5060>.
Proxy-Authorization: Digest username=»1001, realm=8.8.3.48, nonce
=4682071833d18d202024678d258908fef07a049b, uri=sip:1000@8.8.3.48,
response=04e92e136fc8143af3c0992b01777688, algorithm=MD5,
cnonce=234abcc436e26670, qop=auth, nc=00000001.
max-forwards: 70.
supported: 100rel.
user-agent: Voip Phone 1.0.
Allow: INVITE, ACK, OPTIONS, BYE, CANCEL, REFER, NOTIFY, SUBSCRIBE,
PRACK, UPDATE.
Content-Type: application/sdp.
Content-Length: 295  .
.
v=0.
o=sdp_admin 30472538 21739392 IN IP4 192.168.0.111.
s=A conversation.
c=IN IP4 192.168.0.111.
t=0 0.
m=audio 10050 RTP/AVP 0 4 18 8 101.
a=rtpmap:0 PCMU/8000.
a=rtpmap:4 G723/8000.
a=rtpmap:18 G729/8000.
a=rtpmap:8 PCMA/8000.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.
a=sendrecv.

U 8.8.3.48:5060 -> 8.8.3.80:62003
SIP/2.0 100 trying -- your call is important to us.
Via: SIP/2.0/UDP 192.168.0.111:5060;branch=z9hG4bK28696;rport=62003;re
ceived=8.8.3.80.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
```

```
To: 1000 <sip:1000@8.8.3.48>.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Server: OpenSer (1.1.0-notls (i386/linux)).
Content-Length: 0.
Warning: 392 8.8.3.48:5060 Noisy feedback tells:  pid=3052 req_src_
ip=8.8.3.80 req_src_port=62003 in_uri=sip:1000@8.8.3.48 out_uri=sip:10
00@192.168.0.100:5060 via_cnt==1.
.

U 8.8.3.48:5060 -> 8.8.3.91:60166
INVITE sip:1000@192.168.0.100:5060 SIP/2.0.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK3094.efbe1187.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK28
696;rport=62003.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Contact: <sip:1001@8.8.3.80:62003>.
max-forwards: 69.
supported: 100rel.
user-agent: Voip Phone 1.0.
Allow: INVITE, ACK, OPTIONS, BYE, CANCEL, REFER, NOTIFY, SUBSCRIBE,
PRACK, UPDATE.
Content-Type: application/sdp.
Content-Length: 290.
P-hint: route[3] - setflag7,forcerport,fix_contact .
P-hint: route[5] - usemediaproxy .
.
v=0.
o=sdp_admin 30472538 21739392 IN IP4 192.168.0.111.
s=A conversation.
c=IN IP4 8.8.3.48.
t=0 0.
m=audio 60012 RTP/AVP 0 4 18 8 101.
a=rtpmap:0 PCMU/8000.
a=rtpmap:4 G723/8000.
a=rtpmap:18 G729/8000.
a=rtpmap:8 PCMA/8000.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.
a=sendrecv.
```

```
U 8.8.3.91:60166 -> 8.8.3.48:5060
SIP/2.0 100 Trying.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK3094.efbe1187.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK28
696;rport=62003.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Content-Length: 0.
.

U 8.8.3.91:60166 -> 8.8.3.48:5060
SIP/2.0 180 Ringing.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK3094.efbe1187.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK28
696;rport=62003.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=914020329.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Contact: <sip:1000@192.168.0.100:5060>.
Content-Length: 0.
.

U 8.8.3.48:5060 -> 8.8.3.80:62003
SIP/2.0 180 Ringing.
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK28
696;rport=62003.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=914020329.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Contact: <sip:1000@8.8.3.91:60166>.
Content-Length: 0.
P-hint: Onreply-route - fixcontact .
.

U 8.8.3.91:60166 -> 8.8.3.48:5060
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK3094.efbe1187.0.
```

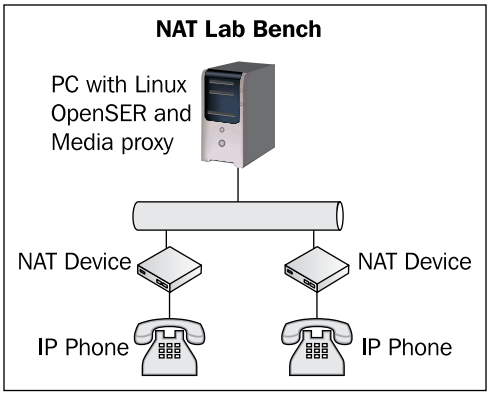
```
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK28
696;rport=62003.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=914020329.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Contact: <sip:1000@192.168.0.100:5060>.
supported: replaces.
Content-Type: application/sdp.
Content-Length: 239  .
.
v=0.
o=sdp_admin 30091196 10973278 IN IP4 192.168.0.100.
s=A conversation.
c=IN IP4 192.168.0.100.
t=0 0.
m=audio 10052 RTP/AVP 0 4 18 8.
a=rtpmap:0 PCMU/8000.
a=rtpmap:4 G723/8000.
a=rtpmap:18 G729/8000.
a=rtpmap:8 PCMA/8000.
a=sendrecv.

U 8.8.3.48:5060 -> 8.8.3.80:62003
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK28
696;rport=62003.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=914020329.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 INVITE.
Contact: <sip:1000@8.8.3.91:60166>.
supported: replaces.
Content-Type: application/sdp.
Content-Length: 234.
P-hint: Onreply-route - fixcontact .
P-hint: Onreply-route - usemediaproxy .
.
v=0.
o=sdp_admin 30091196 10973278 IN IP4 192.168.0.100.
s=A conversation.
c=IN IP4 8.8.3.48.
```

```
t=0 0.
m=audio 60012 RTP/AVP 0 4 18 8.
a=rtpmap:0 PCMU/8000.
a=rtpmap:4 G723/8000.
a=rtpmap:18 G729/8000.
a=rtpmap:8 PCMA/8000.
a=sendrecv.

U 8.8.3.80:62003 -> 8.8.3.48:5060
ACK sip:1000@8.8.3.91:60166 SIP/2.0.
Via: SIP/2.0/UDP 192.168.0.111:5060;branch=z9hG4bK17145.
Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=914020329.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 ACK.
max-forwards: 70.
user-agent: Voip Phone 1.0.
Content-Length: 0.
.

U 8.8.3.48:5060 -> 8.8.3.91:60166
ACK sip:1000@8.8.3.91:60166 SIP/2.0.
Record-Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK3094.efbe1187.2.
Via: SIP/2.0/UDP 192.168.0.111:5060;received=8.8.3.80;branch=z9hG4bK1
7145.
From: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
To: 1000 <sip:1000@8.8.3.48>;tag=914020329.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 2 ACK.
max-forwards: 69.
user-agent: Voip Phone 1.0.
Content-Length: 0.
P-hint: Loose-Route - fixcontact,setflag6 .
.

U 8.8.3.91:60166 -> 8.8.3.48:5060
BYE sip:1001@8.8.3.80:62003 SIP/2.0.
Via: SIP/2.0/UDP 192.168.0.100:5060;branch=z9hG4bK29815;rport.
Route: <sip:8.8.3.48;lr=on;ftag=2824524117;nat=yes>.
From: 1000 <sip:1000@8.8.3.48>;tag=914020329.
To: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 BYE.
```

```
max-forwards: 70.
user-agent: Voip Phone 1.0.
Content-Length: 0.
.

U 8.8.3.48:5060 -> 8.8.3.80:62003
BYE sip:1001@8.8.3.80:62003 SIP/2.0.
Record-Route: <sip:8.8.3.48;lr=on;ftag=914020329;nat=yes>.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK6094.da62cfc5.0.
Via: SIP/2.0/UDP 192.168.0.100:5060;received=8.8.3.91;branch=z9hG4bK29
815;rport=60166.
From: 1000 <sip:1000@8.8.3.48>;tag=914020329.
To: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 BYE.
max-forwards: 69.
user-agent: Voip Phone 1.0.
Content-Length: 0.
P-hint: Loose-Route - fixcontact,setflag6 .
.

U 8.8.3.80:62003 -> 8.8.3.48:5060
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 8.8.3.48;branch=z9hG4bK6094.da62cfc5.0.
Via: SIP/2.0/UDP 192.168.0.100:5060;received=8.8.3.91;branch=z9hG4bK29
815;rport=60166.
Record-Route: <sip:8.8.3.48;lr=on;ftag=914020329;nat=yes>.
From: 1000 <sip:1000@8.8.3.48>;tag=914020329.
To: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 BYE.
Contact: <sip:1001@192.168.0.111:5060>.
Content-Length: 0.
.

U 8.8.3.48:5060 -> 8.8.3.91:60166
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 192.168.0.100:5060;received=8.8.3.91;branch=z9hG4bK29
815;rport=60166.
Record-Route: <sip:8.8.3.48;lr=on;ftag=914020329;nat=yes>.
From: 1000 <sip:1000@8.8.3.48>;tag=914020329.
To: 1001 <sip:1001@8.8.3.48>;tag=2824524117.
Call-ID: 38476419-134726572@192.168.0.111.
CSeq: 1 BYE.
Contact: <sip:1001@8.8.3.80:62003>.
Content-Length: 0.
P-hint: Onreply-route - fixcontact
```

Lab Using MediaProxy for NAT Traversal

To test NAT traversal is not an easy task. You can test with some friends behind the Internet calling your server on a public IP address. For a test bench the easiest setup is to have two IP phones behind two NAT devices.



Step 1: Copy the `openser.mediaproxy` script to `/etc/openser/openser.cfg`:

```
cd /etc/openser
wget http://www.asteriskguide.com/openser/openser.mediaproxy
cp openser.mediaproxy openser.cfg
```

Step 2: Dowload and decompress the MediaProxy server from AG Projects to the directory `/usr/src`.

```
cd /usr/arc
wget http://mediaproxy.ag-projects.com/mediaproxy-1.8.2.tar.gz
tar -xzvf mediaproxy-1.8.2.tar.gz
```

[ Please, check for newer versions.]

Step 3: Install python if it is not already installed:

```
apt-get install python
```

Step 4: Edit `mediaproxy.ini` according to step 4 of the earlier section *Installing MediaProxy*.

```
vi mediaproxy.ini
```

Step 5: Start the `mediaproxy` process.

```
./mediaproxy.py
```

Step 6: Assemble you lab bench and register phones 1000 and 1001 behind

NAT devices.

Step 7: Use `ngrep` to record all the packets.

```
ngrep -p -q -W byline port 5060
```

Step 8: Test the configuration dialing from phone 1000 to phone 1001.

Step 9: Verify the RTP flows using:

```
/usr/local/mediaproxy/sessions.py
```

Implementing a Near-End NAT Solution

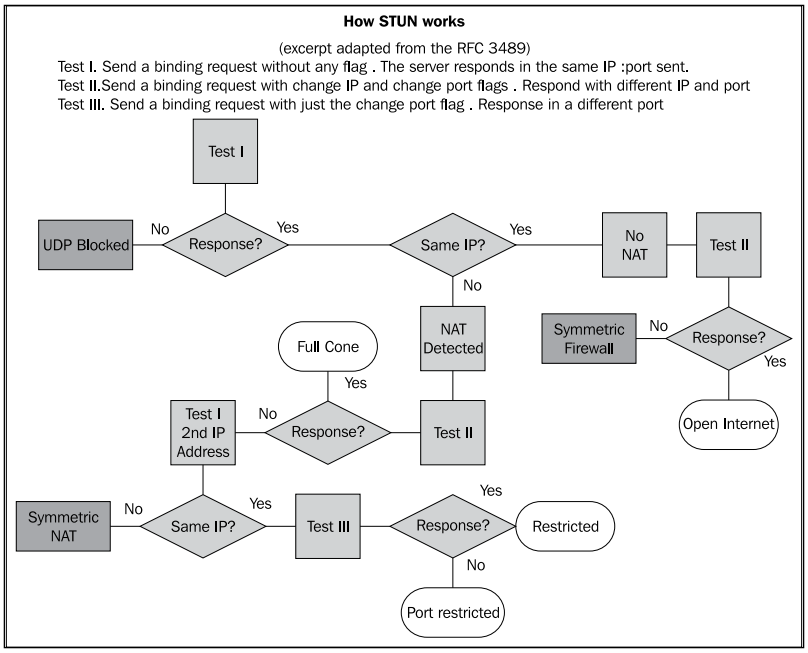
Simple Traversal of UDP over NAT or simply STUN is the most common method for near-end NAT traversal. STUN is based on RFC3489 and is considered a near-end NAT traversal solution. The biggest advantage of using STUN is that the client seems to the proxy to be in the public Internet. You don't require any configuration on the server for NAT traversal. The biggest disadvantage is that it does not work with symmetric NAT devices.

The STUN protocol allows IP endpoints behind NAT devices discover their external IP address and ports. With this information a device can inform another party at what address it can be contacted.

STUN is relatively complex, with several messages such as mapped-address, changed-address, source-address, response-address, and change-IP. With this information it can discover if the client is:

- In an open Internet
- Behind a firewall that blocks UDP
- Behind a NAT device, and if so what kind of device it is

In the diagram below all the results in green boxes can be handled by STUN. The other situations can only be traversed with the media relay solution (TURN).



You can implement STUN using a Linux server with Vovida software. You will need a server and two public IP addresses for this server. The STUN server from Vovida can be downloaded from `http://www.vovida.org/`.

It is possible to test STUN without installing a server; simply use a public STUN server. A short list of public STUN servers and a lot more information about STUN can be found at `http://www.voip-info.org/wiki-STUN`.

If you use STUN in the client, it won't be necessary to make any changes in the `openser.cfg` script. It will work as if the client was directly connected to the Internet.

Why STUN Does Not Work with Symmetric NAT Devices

The main characteristic of a symmetric NAT device is its creation of a new mapping for each external device contacted. So if you contact the STUN device it will inform you the "ip:port" pair from which it has been contacted. Unfortunately, this "ip:port" pair informed won't be the mapping created for any other device, so this information is useless.

In the first three kinds of NAT devices (Cone, Cone Restricted, and Port Restricted) the mapping created to one device would be exactly the same to other devices, since the internal "ip:port" is the same.

Comparing STUN with TURN (Media Relay Server)

STUN allows a better scalability and the endpoints can communicate directly. With Media Rely Server, if a UAC wants to communicate with another UAC, they will have to use your server to relay the RTP packets. This will consume your bandwidth and in consequence your money. Worse, the payload is twice of a normal PSTN call, because you have to relay the RTP session from two UACs. CPU resources are also spent to bridge the packets.

STUN is great, I love STUN, but it does not solve the problem completely. It is very hard to implement a VoIP provider without taking symmetric NAT devices into consideration. Symmetric NAT devices are very common. You can check and even add some devices to the NAT Survey at `http://www.voip-info.org/wiki/view/NAT+survey`.

Clients behind STUN are identified as clients with a public IP address. The SIP proxy does not need any special handling for these packets. Use STUN whenever possible (for any NAT device except symmetric NAT). Use media relay services for users behind a symmetric NAT device.

There are free implementations of STUN servers and clients. You can find a lot of information about STUN at `http://www.voip-info.org/wiki-STUN`.

ALG—Application Layer Gateways

Another very common solution for near-end NAT traversal is ALG (application layer gateway). Several NAT devices implement ALG. In this case the NAT device changes the SIP and SDP headers to make the packets look as if they had been originated in the external interface with a public address. My personal experience with ALG is not good. Some ADSL modem routers have broken implementations of ALG and freeze

when accessing a SIP provider. Another implementation changes the headers but not the MD5 digest, giving me an authentication error (I was using the IP address of the SIP server and not the host name; using the host name solved the problem).

It is important to be aware of NAT devices with ALG on your network. When something is not working this is an important place to check.

ICE (Interactive Connection Establishment)

ICE is the newest protocol for NAT traversal. It combines NAT and TURN to choose the best path available. ICE uses all the available methods of TURN or STUN to check all possible connectivity addresses. It uses the best possible solution available avoiding the reconfiguration of each client.

Summary

In this chapter you have been presented with the different NAT types and devices. You have seen the implications of symmetrical NAT, the use of STUN and TURN. At the end of this chapter you have learned how to implement the MediaProxy solution to solve the NAT traversal problem.

As a rule of thumb, use STUN always when possible, it uses less processing power in your voice provider. If your customer is behind a symmetrical NAT, have the option to use Mediaproxy or RTPproxy.

MediaProxy can be load balanced, but supports a limited number of users per box. RTPproxy, developed in C, is a lot faster and can support a lot more users in a single box. On the other hand, it does not allow load balancing in the same way MediaProxy does and it does not help you with accounting. MediaProxy can detect RTP timeouts and adjust the values in the accounting server (RADIUS). Choose wisely.

10
OpenSER Accounting and Billing

In the last chapter we learned how to implement NAT traversal; now it is time to focus on the most important thing for a VoIP provider, the revenue. The accounting feature will allow you to determine the exact duration of each call. We will show you two methods. The first one is using MySQL and later using a RADIUS server. RADIUS is a de facto standard for AAA (Authentication, Authorization, and Accounting). Duration in minutes is not enough to bill the customers. You will also need a rating tool. This tool is able to convert minutes to whatever currency you will use to bill your customer.

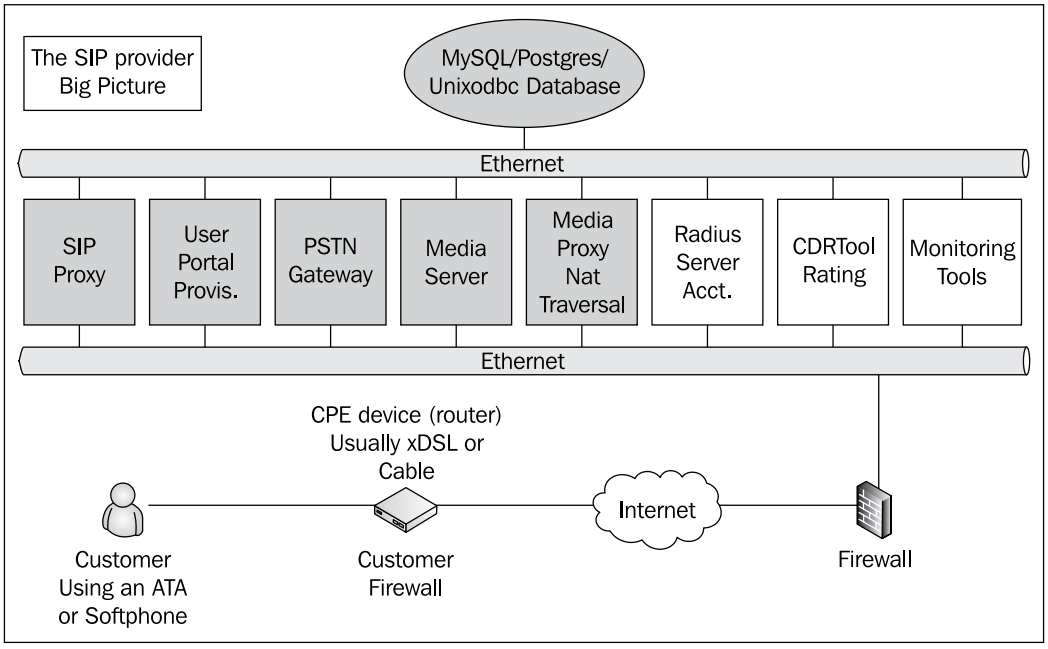
Objectives

By the end of this chapter you will be able to:

- Enable accounting on a MySQL server
- Enable accounting an a RADIUS server
- Rate calls using the CDRTool from AG Projects

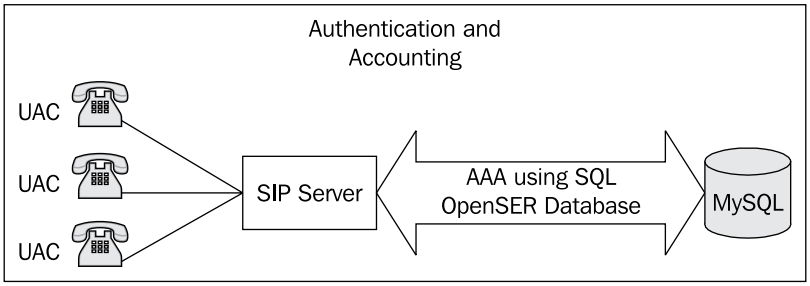
Where Are We?

We are going to work on the billing side of the solution. The proxy is working fine, completing calls between users and gateways. However, we are not billing the calls. Billing is a two-step process. In the first place, you have to determine the duration of the call. This is done using RADIUS or MySQL. The next step is to determine the price of that single call. We will use CDRTool, an open-source tool to help us in this task.



VoIP Provider Architecture

The VoIP server uses the concept of AAA (Authentication, Authorization, and Accounting). Until now, we have used only MySQL to authenticate and to authorize users. We can use MySQL or RADIUs to store the accounting data. It is easier to work with a RADIUS server because it uses an account-start packet for each INVITE transaction and an account-stop packet for each BYE transaction, writing a single record with the duration of the call. When you use MySQL you have to manually correlate INVITE and BYE transactions.



Accounting Configuration

Billing is an exceptional means of verifying the messages. It gives the status of the ended transactions. The billing process also gives the results of the INVITE and BYE transactions. To correlate the INVITE and BYE transactions is a function of the billing software. The best place to bill the calls is on the gateways, because a call can be left open after an INVITE without the correlated BYE. Another good reason is because a SIP proxy stays in the middle of the SIP signaling with very little control over the media. A proxy can be bypassed by the signaling after the call start, so the accounting info will be incomplete. In the gateways it is possible to set session timeouts to terminate unfinished SIP dialogs.

To enable the accounting feature we will use the ACC module. It will account to a MySQL database. We are going to use phpMyAdmin to check the database records. We need to set a flag in the transactions that we want to be accounted. The ACC module for version 1.2 is a bit different than the existing one in OpenSER 1.1. Now, the module logs by default just the following data:

ID	Method	from_tag	to_tag	callid	sip_code	sip_reason	Time
1	INVITE	5d09d45a	27095f70	ZTY5ND.	200	OK	2008-04-07 09:13:21
2	BYE	5d09d45a	27095f70	ZTY5ND.	200	OK	2008-04-07 09:13:30

So, to have something identifying caller and callee, you need to add some extra data.

LAB—Accounting using MySQL

To avoid a huge and complex script, let's implement the accounting over the script developed in Chapter 7 (about connecting to PSTN Gateways).

Step 1: Add the following fields in the ACC table:

```
mysql  -u root
   USE OPENSER;
   ALTER TABLE 'acc' ADD 'from_uri' VARCHAR( 64 ) NOT NULL ;
   ALTER TABLE 'acc' ADD 'to_uri' VARCHAR( 64 ) NOT NULL ;
```

Step 2: Use the script below:

```
   #
   # $Id: openser.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $
   #
   # simple quick-start config script
```

```
# Please refer to the Core CookBook at http://www.openser.org/
dokuwiki/doku.php
# for a explanation of possible statements, functions and parameters.
#

# ----------- global configuration parameters ------------------------

debug=3            # debug level (cmd line: -ddddddddddd)
fork=yes
log_stderror=no    # (cmd line: -E)
children=4
port=5060

# ------------------ module loading ----------------------------------
#set module path
mpath="//lib/openser/modules/"

# Uncomment this if you want to use SQL database
#loadmodule "mysql.so"

loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "acc.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"

# ----------------- setting module-specific parameters ---------------

modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
```

```
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|usrloc","db_url","mysql://
openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
modparam("acc", "db_flag", 2)
modparam("acc", "db_missed_flag", 3)
modparam("acc", "db_url", "mysql://openser:openserrw@localhost/
openser")
modparam("acc", "db_extra", "from_uri=$fu; to_uri=$tu")


# ------------------------  request routing logic -------------------

# main routing logic

route{

   #
   # -- 1 -- Request Validation
   #
   if (!mf_process_maxfwd_header("10")) {
       sl_send_reply("483","Too Many Hops");
       exit;
   };

   if (msg:len >=  2048 ) {
       sl_send_reply("513", "Message too big");
       exit;
   };

   #
   # -- 2 -- Routing Preprocessing
   #
   ## Record-route all except Register
   if (!method=="REGISTER") record_route();

   ##Loose_route packets
   if (has_totag()) {
       #sequential request withing a dialog should
       # take the path determined by record-routing
       if (loose_route()) {
           if(method=="BYE") {
```

```
                #Account BYE transactions
                setflag(2);
            };
            #Check authentication of re-invites
            if(method=="INVITE" && (!allow_trusted())) {
                if (!proxy_authorize("","subscriber")) {
                    proxy_challenge("","1");
                    exit;
                } else if (!check_from()) {
                    sl_send_reply("403", "Forbidden, use From=ID");
                    exit;
                };
            };
            route(1);
        } else {
            sl_send_reply("404","Not here");
        }
        exit;
    }

    #CANCEL processing
    if (is_method("CANCEL")) {
        if (t_check_trans()) t_relay();
            exit;
    };

    t_check_trans();

    #
    # -- 3 -- Determine Request Target
    #
    if (method=="REGISTER") {
        route(2);
    } else {
        route(3);
    };
}

route[1] {
    #
    # -- 4 -- Forward request to target
    #
    ## Forward statefully
    if (!t_relay()) {
```

```
        sl_reply_error();
    };
    exit;
}

route[2] {
    ## Register request handler
    if (is_uri_host_local()) {
        if (!www_authorize("", "subscriber")) {
            www_challenge("", "1");
            exit;
        };

        if (!check_to()) {
            sl_send_reply("403", "Forbidden");
            exit;
        };

        save("location");
            exit;
    } else if {
        sl_send_reply("401", "Forbidden");
    };
}

route[3] {
    ## Non-Register request handler
    if (method=="INVITE") {
        # Account INVITE packets
        setflag(2);
        # Account Missed calls
        setflag(3);
    };
    if (is_from_local()){
       # From an internal domain -> check the credentials and the FROM
         if(!allow_trusted()){
              if (!proxy_authorize("","subscriber")) {
                   proxy_challenge("","1");
                   exit;
              } else if (!check_from()) {
                   sl_send_reply("403", "Forbidden, use From=ID");
                   exit;
              };
         } else {
```

```
                log("Request bypassed the auth.using allow_trusted");
            };

           consume_credentials();

              #Verify aliases, if found replace R-URI.
              lookup("aliases");

              if (is_uri_host_local()) {
                    # -- Inbound to Inbound
                    route(10);
           } else {
                    # -- Inbound to outbound
                    route(11);
           };

      } else {
           #From an external domain ->do not check credentials

           #Verify aliases, if found replace R-URI.
           lookup("aliases");
           if (is_uri_host_local()) {
                       #-- Outbound to inbound
                       route(12);
           } else {
                       # -- Outbound to outbound
                       route(13);
           };
      };
}

route[4] {
           # routing to the public network
           rewritehostport("10.125.123.177");
           route(1);
}

route[10] {
     #from an internal domain -> inbound
     #Native SIP destinations are handled using the location table
     #Gateway destinations are handled by regular expressions
     append_hf("P-hint: inbound->inbound \r\n");

     if (uri=~"^sip:[2-9][0-9]{6}@") {
```

```
            if (is_user_in("credentials","local")) {
                prefix("+1305");
              route(4);
                exit;
            } else {
                sl_send_reply("403", "No permissions for local calls");
                exit;
            };
        };

        if (uri=~"^sip:[2-9][1-9]{9}@") {
            if (is_user_in("credentials","ld")) {
                prefix("+1");
            route(4);
                exit;
            } else {
                sl_send_reply("403", "No permissions for
                              long distance");
                exit;
            };
        };

        if (uri=~"^sip:011[0-9]*@") {
            if (is_user_in("credentials","int")) {
                strip(3);
                prefix("+");
                route(4);
                exit;
            } else {
                sl_send_reply("403", "No permissions for
                              international calls");
            };
        };

        if (!lookup("location")) {
            sl_send_reply("404", "Not Found");
            exit;
        };
        route(1);
    }

    route[11] {
        # from an internal domain -> outbound
        # Simply route the call outbound using DNS search
```

```
        append_hf("P-hint: inbound->outbound \r\n");
        route(1);
    }

    route[12] {
        # From an external domain -> inbound
        # Verify aliases, if found replace R-URI.
        lookup("aliases");
        if (!lookup("location")) {
            sl_send_reply("404", "Not Found");
            exit;
        };
        route(1);
    }

    route[13] {
        #From an external domain outbound
        #we are not accepting these calls
        append_hf("P-hint: outbound->inbound \r\n");
        sl_send_reply("403", "Forbidden");
        exit;
    }
```

Step 3: Make a call between two available SIP devices.

Step 4: Verify the accounting table using the MySQL command-line interface:

```
#mysql -u root
    mysql>use openser
    mysql>select * from acc;
```

openser.cfg Analysis

The accounting feature is very simple to implement. The first step is to load the accounting module.

```
    loadmodule "acc.so"
```

The second step is the configuration of the module's parameters. The first parameter, `db_flag`, tells OpenSER to account transactions marked with the flag number `2`. The parameter `db_missed_flag` tells OpenSER to account missed calls. The parameter `db_extra` allows you to include new data to your database. Use the name of the field (`from_uri`) you have previously created in the database and a value that can be taken from pseudo-variables, AVPs or headers.

```
modparam("acc", "db_flag", 2)
modparam("acc", "db_missed_flag", 3)
modparam("acc", "db_url",
         "mysql://openser:openserrw@localhost/openser")
modparam("acc", "db_extra", "from_uri=$fu; to_uri=$tu")
```

Let's modify our script to account INVITE and BYE transactions for a while.

```
if (loose_route()) {
            if(method=="BYE") {
                 #Account BYE transactions
                 setflag(2);
            };
```

BYEs are being flagged in the `loose_route` section, because we are using record-route. INVITEs are being flagged for accouting in the initial requests section. You don't need to flag re-INVITEs.

```
if (method=="INVITE") {
          # Account INVITE packets
          setflag(2);
          # Account Missed calls
          setflag(3);
   };
```

You can check the accounting tables using the phpMyAdmin web utility. Look for the ACC table in the OpenSER database and select **Browse**.

Accounting using RADIUS

RADIUS (Remote Authentication Dial-in User Service) is a kind of AAA (Authentication, Authorization, and Accounting) service. It is a de facto standard for Internet Access Providers. Over the last few years RADIUS has been becoming an important security protocol used in several network applications such as NAC (Network Access Control) and VoIP accounting. You can implement a RADIUS server using the open-source package called FreeRADIUS. There are other RADIUS packages, licensed open-source and commercial. A good list of RADIUS servers can be found at:

`http://en.wikipedia.org/wiki/List_of_RADIUS_Servers`.

RADIUS was defined primarily in two RFCs:

- RFC2865—Authentication
- RFC2866—Accounting

In this chapter we will use RADIUS only for accounting. MySQL will be held in the authentication function. Let's install FreeRADIUS according to the instructions for our rating tool (CDRTool).

Installation of FreeRADIUS and CDRTool

The installation of the FreeRADIUS server is unquestionably a challenge. Several steps have to be strictly followed to have a working configuration. To do this we will divide the installation in five steps:

1. Package and dependencies installation
2. Database configuration
3. FreeRADIUS configuration
4. Radiusclient-ng installation
5. OpenSER configuration

Packages and Dependencies

For FreeRADIUS, freeradius-mysql and CDRTool install the packages below:

```
apt-get install freeradius freeradius-mysql
```

Create and Configure the Database for the Radius server

Step 1: Create the database for the FreeRADIUS server.

```
mysqladmin -u root –p create radius
```

Type the root password whenever required.

Step 2: Create the tables below to support the RADIUS server.

```
cd /usr/share/doc/freeradius/examples
gunzip mysql.sql.gz
mysql -u root radius <mysql.sql
```

Step 3: Apply the RADIUS and radacct patches that comes together with CDRTool. The FREERADIUS patch fixes a problem with accounting type 15. The other patch, radacct-patch, modifies the insert and update queries to send data to the RADIUS SQL database.

Download the FREERADIUS patch:

```
wget http://download.dns-hosting.info/CDRTool/freeradius/freeradius.patch
```

Apply the patch.

```
apt-get build-dep freeradius
apt-get source freeradius
apt-get install devscripts
cd freeradius-1.1.3
patch -p1 -s < ../freeradius.patch
debuild
cd ../
dpkg -i freeradius*.deb
```

Step 4: Install CDRTool to use its supporting files for RADIUS:

```
wget http://download.dns-hosting.info/CDRTool/cdrtool_6.4.1_all.deb
dpkg –i cdrtool_6.4.1_all.deb
apt-get –f install
```

Please, check the current version. It often changes.

Step 5: Apply the RADIUS accounting patch that comes together with CDRTool.

```
cd /var/www/CDRTool/setup/radius/OpenSER
./radacct-patch.sh
```

Step 6: Edit the database connection parameters in the SQL.CONF file.

```
vi /var/www/CDRTool/setup/radius/sql.conf
```

```
#
#  Configuration for the Freeradius SQL module using MySQL and a
#  central radacct table. To use monthly tables with auto-rotation
#. see radius_accounting.conf and radius_accounting.proc
#
sql {
    driver                      = "rlm_sql_mysql"
```

```
server                          = "127.0.0.1"
login                           = "root"
password                        = ""
radius_db                       = "radius"
acct_table                      = "radacct"
sqltrace                        = no
sqltracefile                    = ${logdir}/sqltrace-%Y%m%d.log
num_sql_socks                   = 25
connect_failure_retry_delay = 60
```

On a production environment don't use root without a password.

Step 7: Copy the file `sql.conf` to the directory `/etc/freeradius/sql.conf`:

```
cp sql.conf /etc/freeradius/sql.conf
```

Configuration of the FreeRADIUS Server

Now we are going to configure the FreeRADIUS servers and clients. OpenSER server is the client of the RADIUS server. It uses the library libradiusclient-ng to connect to the RADIUS server.

Step 1: Add OpenSER as a FreeRADIUS client.

In the RADIUS protocol architecture you have to define the devices that will send the authentication and accounting packets to the RADIUS server. Usually these devices are remote access gateways, 802.1X switches, and access points. In our case the RADIUS client is the SIP proxy server that will be sending the account requests.

Edit the `clients.conf` file in the FreeRADIUS configuration directory.

```
vi /etc/freeradius/clients.conf
```

Example:

```
client 127.0.0.1 {
        secret=openser
        shortname=OpenSER
        nastype=other
}
```

Step 2: Enable the MySQL accounting in FreeRADIUS. Edit the `radiusd.conf` file and uncomment or add the required lines. (This is the place where you often will make a mistake. The accounting section is deep in the file. I have copied a `radiusd.conf` to the same place as the book scripts to help you.)

```
Accounting {
    acct_unique
    detail
    sql
    unix
    radutmp
    }
```

Step 3: Copy the OpenSER dictionary to the `/etc/freeradius`.

```
cp /var/www/CDRTool/setup/radius/OpenSER/dictionary.ser /etc/freeradius
```

Step 4: Include the OpenSER dictionary in the RADIUS server configuration file.

```
cd /etc/freeradius
vi dictionary
```

```
#       The filename given here should be an absolute path.
#
$INCLUDE        /usr/share/freeradius/dictionary
$INCLUDE        /etc/freeradius/dictionary.ser
```

Be sure to include in the correct order.

Step 5: Restart the freeradius server.

```
/etc/init.d/freeradius restart
```

Configure the RADIUS Client (radiusclient-ng)

Step 1: Copy the file `dictionary.radius` to the RADIUS client configuration directory.

```
#cp /etc/openser/dictionary.radius /etc/radiusclient-ng/
```

Step 2: Edit the file `/etc/radiusclient-ng/dictionary`.

```
vi /etc/radiusclient-ng/dictionary
```

Add the line below as the *last line* of the file:

```
$INCLUDE                /etc/freeradius/dictionary.ser
```

Step 3: Edit the file `/etc/radiusclient-ng/servers`:

Configure the RADIUS server with the related key.

```
#Server Name or Client/Server pair              Key
#----------------                               ---------------
#portmaster.elemental.net                       hardlyasecret
#portmaster2.elemental.net                      donttellanyone
127.0.0.1                                       openser
```

Step 4: Edit the file `/etc/radiusclient-ng/radiusclient.conf`:

Add the IP address of your RADIUS server.

```
# RADIUS server to use for accouting requests. All that I
# said for authserver applies, too.
Acctserver   127.0.0.1
```

Configure OpenSER

The accounting module has support for RADIUS, but, by default it is not enabled. So let's enable the RADIUS support for OpenSER. We will have to recompile OpenSER to do this.

Step 1: edit the file Makefile of the module acc.

```
vi /usr/src/openser-1.2.2-tls/modules/acc/Makefile
```

Remove the comments (#) from the highlighted lines.

```
# uncomment the next two lines if you wish to enable RADIUS accounting
DEFS+=-DRAD_ACC -I$(LOCALBASE)/include
LIBS=-L$(LOCALBASE)/lib $(RADIUS_LIB)
```

Step 2: Recompile OpenSER:

```
cd /usr/src/openser-1.2.2-tls
make prefix=/ clean
make prefix=/ all
make prefix=/install
```

Step 3: Add the following entries in the `openser.cfg` file.

```
modparam("acc", "radius_config",      "/etc/radiusclient-ng/
radiusclient.conf")
modparam("acc", "radius_flag",        2)
modparam("acc", "radius_missed_flag", 3)
modparam("acc", "radius_extra",       "User-Name=$Au; \
```

```
Calling-Station-Id=$from; \
Called-Station-Id=$to; \
Sip-Translated-Request-URI=$ruri; \
Sip-RPid=$avp(s:rpid); \
Source-IP=$si; \
Source-Port=$sp; \
Canonical-URI=$avp(s:can_uri); \
Billing-Party=$avp(s:billing_party); \
Divert-Reason=$avp(s:divert_reason); \
X-RTP-Stat=$hdr(X-RTP-Stat); \
Contact=$hdr(contact); \
Event=$hdr(event); \
SIP-Proxy-IP=$avp(s:sip_proxy_ip); \
ENUM-TLD=$avp(s:enum_tld)")
```

Step 4: Restart the OpenSER Server.

Test the Configuration after Making a Call

Check the RADIUS accounting tables (`radacct` table in the RADIUS database). You can use the phpMyAdmin utility to do it. The SIP accounting is composed of two records, start and stop. The start event is triggered by an INVITE request and the stop event by the BYE request. In the initial event the attribute `Calling-Station-Id` identifies the caller and the `Called-Station-Id` the callee. The duration of the call is established by the difference between the timestamps of these two events. You can also check the RADIUS log files at `/var/log/freeradius/radacct`.

Example:

Sun Mar 12 17:29:21 2006

Acct-Status-Type = Start
Service-Type = Sip-Session
Sip-Response-Code = 200
Sip-Method = INVITE
User-Name = "101@openser.org"
Calling-Station-Id = "sip:101@openser.org"
Called-Station-Id = "sip:102@openser.org"
Sip-Translated-Request-URI = "sip:102@192.168.0.12:5066"
Acct-Session-Id = "1dbe198c82543fa2@192.168.0.11"
Sip-To-Tag = "00D0E90101B8_T9513"
Sip-From-Tag = "111aa0fda452c726"
Sip-Cseq = "4435"

Sip-Src-IP = "192.168.0.11"
Sip-Src-Port = "5068"
NAS-IP-Address = 127.0.0.1
NAS-Port = 5060
Acct-Delay-Time = 0
Client-IP-Address = 10.10.10.10
Acct-Unique-Session-Id = "37fb00358437ff4d"
Timestamp = 1142177361
Sun Mar 12 17:29:28 2006
Acct-Status-Type = Stop
Service-Type = Sip-Session
Sip-Response-Code = 200
Sip-Method = BYE
User-Name = "102@openser.org"
Calling-Station-Id = "sip:102@openser.org"
Called-Station-Id = "sip:101@openser.org"
Sip-Translated-Request-URI = "sip:101@192.168.0.11:5068"
Acct-Session-Id = "1dbe198c82543fa2@192.168.0.11"
Sip-To-Tag = "111aa0fda452c726"
Sip-From-Tag = "00D0E90101B8_T9513"
Sip-Cseq = "3305"
Sip-Src-IP = "192.168.0.12"
Sip-Src-Port = "5066"
NAS-IP-Address = 127.0.0.1
NAS-Port = 5060
Acct-Delay-Time = 0
Client-IP-Address = 10.10.10.10
Acct-Unique-Session-Id = "597f048f3aa62ca0"
Timestamp = 1142177368

Using CDRTool for Rating

One of the most valuable aspects of a VoIP provider is billing. Let's examine an open-source tool called CDRTool to do this. CDRTool has been developed by AG Project (`www.ag-projects.com`) and is licensed according to GPL.

CDRTool is a web application able to rate calls accounted in the FreeRADIUS tables. It is able to rate calls based in its internal tables. Beyond rating calls, the system is able to trace calls using the **siptrace** feature of OpenSER and apply **anti-fraud** mechanisms based on quotas. The component rating engine is a daemon that can receive TCP requests, making possible the implementation of a pre-paid mechanism.

In this material we will focus just on rating calls. Tracing calls will be seen in Chapter 11.

MediaProxy can be used in conjunction with CDRTool to provide 100% accurate accounting, regardless of the availability of BYE messages.

Having access to information from both the SIP signaling and the RTP media (MediaProxy), CDRTool can be used to implement several billing modes including traffic variables, or a combination of destination, application type, and duration.

LAB—CDRTool Installation

The CDRTool installation is very difficult. To install it, strictly follow the instructions below:

Step 1: The installation of the CDRTool package should be done as explained in the section *Installation of FreeRADIUS and CDRTool*.

Step 2: Create the CDRTool database.

Go to the `/var/www/CDRTool/setup/mysql` directory:

```
cd /var/www/CDRtool/setup/mysql
```

Edit the file `create_users.mysql`:

```
vi create_users.mysql
```

```
GRANT ALL ON cdrtool.* TO cdradmin@'localhost' IDENTIFIED by
'password';
GRANT ALL ON cdrtool.* TO cdradmin@'192.168.1.%' IDENTIFIED by
'password';
GRANT ALL ON cdrtool.* TO locker@'localhost' IDENTIFIED by 'password';
GRANT ALL ON cdrtool.* TO locker@'192.168.1.%' IDENTIFIED by
'password';
```

Execute the installation script. This step will create the CDRTool database. It also creates the initial account for login use (`admin` user with `admin` password).

```
./setup_mysql.sh "" localhost
```

Press *Enter* when prompted for the password or the root password, if you have set one.

Step 3: Create the configuration file `global.inc`.

```
cd /var/www/CDRTool
cp setup/global.inc.new.installation global.inc
```

Edit the file `global.inc` and configure the variable to match your system. For each different data source you should create a new instance.

```
<?

#
# 1. Change all hostnames and passwords according to the installation
# 2. Copy this file to /var/www/CDRTool/global.inc
#

##########################################
# System and web paths

$CDRTool['tld']    = "/CDRTool";
$CDRTool['Path']   = "/var/www/CDRTool";
$_PHPLIB['libdir'] = $CDRTool['Path']. "/phplib/";
include($_PHPLIB["libdir"] . "prepend.php3");
$global_local    = $CDRTool['Path']."/global.inc.local";

##########################################
# PHP Error reporting

$errorReporting = (E_ALL & ~E_NOTICE);
$errorReporting = 1;    // comment this out to enable PHP warnings
error_reporting($errorReporting);

##########################################
# Service provider information

$CDRTool['provider']['name']                  = "VOFFICETel";
$CDRTool['provider']['service']               = "SIP service";
$CDRTool['provider']['timezone']              = "Sao Paulo";
$CDRTool['provider']['fromEmail']             = "flavio@voffice.com.
br";
$CDRTool['provider']['toEmail']               = "support@voffice.com.
br";
```

```
$CDRTool['provider']['sampleLoginSubscriber'] = "number@voffice.com.
br";
$CDRTool['provider']['sampleLoginDomain']     = "voffice.com.br";

##########################################
# Where the rating engine listens for network requests:

$RatingEngine=array("socketIP"   => "192.168.1.170",
                    "socketPort" => "9024",
                    "CDRS_class" => "ser_radius");

$memcache_server  = "127.0.0.1:11212";

##########################################
# Normalize engine settings

$CDRTool['normalize']['defaultCountryCode']      = "55";
$CDRTool['normalize']['CountryNumberLength']     = "10";

##########################################
# Anti-fraud settings
# create group quota in SER and deny calls to users in this group
$UserQuota["default"]["traffic"]  = 5000;              // MBytes
$UserQuota["default"]["cost"]     = 1000;              // Euro


##########################################
# CDRTool datasources

class DB_CDRTool extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "cdrtool";
  var $User     = "cdradmin";
  var $Password = "senha";
  var $Halt_On_Error ="no";
}

class DB_Locker extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "cdrtool";
  var $User     = "locker";
  var $Password = "senha";
  var $Halt_On_Error ="no";
}
```

```
class DB_radius extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "radius";
  var $User     = "radius";
  var $Password = "senha";
  var $Halt_On_Error ="no";
}

class DB_ser extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "openser";
  var $User     = "openser";
  var $Password = "openserrw";
  var $Halt_On_Error ="no";
}


class DomainAuthLocal extends DomainAuth {      // defined in
                                                   phplib/local.inc
}

class PageLayoutLocal extends PageLayout {      // defined in
                                                   phplib/local.inc
}

$DATASOURCES=array(
"unknown"=>array(
                    "class"         => "CDRS_unknown" // leave it here
                    ),
"ser_radius"=>array(
                    "name"               => "OpenSER",
                    "class"              => "CDRS_ser_radius",
                    "table"              => "radacct",
                    "db_class"           => "DB_radius",
                    "db_class_readonly"  => "DB_radius",
                    "db_class_siponline" => "DB_ser",
                    "rateField"          => "Rate",
                    "rating"             => "1",
                    "priceField"         => "Price",
                    "normalize0SecCalls" => "1",
                    "DestinationIdField" => "DestinationId",
                    "normalizedField"    => "Normalized",
                    "BillingPartyIdField"=> "UserName",
```

```
                    "AccountsDBClass"    => "DB_ser",
                    "intAccessCode"      => "00",
                    "sipTraceDataSource" => "sip_trace",
                    "traceOutURL"        => array(
                                        "sipvm.example.com"=>"asterisk",
                                        "pstn.example.com"=>"cisco"
                                             ),
                    "UserQuotaClass"     => "SERQuota",
                    "UserQuotaTable"     => "user_quota",
                    "UserQuotaVerbose"   => "",
                    "UserQuotaNotify"    => "0",
                    "MinPstnNumLen"      => "9",
                    "EnableSIPOnline"    => "0",
                    "EnableNetworkRating"=> "1",
                    "domainTranslation"  => array(
                         "gw02.example.com"      => "pstn.example.com"
                                             ),
                    "EnableSIPOnline"    => 1,
                    "RotateTables"       => "Ym",
                    "RotateThisMonth"    => 0,
                    "purgeCDRsAfter"     => 120 // how many days to
                                                   keep the CDRs
                    ),
  "asterisk_vm"  =>array("name"            => "Voicemail server",
                    "class"              => "CDRS_asterisk",
                    "table"              => "asterisk_cdr",
                    "db_class"           => "DB_radius",
                    "rateField"          => "Rate",
                    "rating"             => "1",
                    "priceField"         => "Price",
                    "DestinationIdField" => "DestinationId",
                    "normalizedField"    => "Normalized",
                    "normalize0SecCalls" => "1",
                    "contexts"           => array(
                                   "SIP"=>array("WEBName"=>"OpenSER"),
                                             ),
                    "traceInURL"         => array(
                                             "SIP"=>"ser_radius"
                                                      ),
                    "traceOutURL"        => array(),
                    "purgeCDRsAfter"    => 180 // how many days to
                                                   keep the CDRs
                    ),
  "sip_trace"   =>array(
```

```
                        "name"               => "SIP trace",
                        "db_class"           => "DB_ser",
                        "table"              => "sip_trace",
                        "purgeRecordsAfter"  => "7"
                        )
    );

    // load CDRTool libraries
    $CDRToolModules=array("ser","asterisk","rating");

    if ($REMOTE_ADDR=="192.168.1.209") {
        //$verbose=1;
    } else {
        // prevent set of verbose via post/get
        unset($verbose);
    }
    ?>
```

Step 4: Enable the rating server.

Edit the file `/etc/default/cdrtool` and configure:

```
RUN_ratingEngine=yes
```

To start the rating server:

```
/etc/init.d/cdrtool restart
```

Step 5: Configure Apache to support CDRTool

In the end of the `/etc/apache2/apache2.conf`, add the following statements:

```
DirectoryIndex index.phtml index.php index.html index.htm
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
```

Restart Apache:

```
/etc/init.d/apache2 restart
```

Step 6: Access your CDRTool using your browser at the following address:

`http://ip_address_of_your_server/CDRTool` or

`http://name_of_your_server/CDRTool`

Step 7: Log in using the user `admin` password `admin`. Change the password as soon as possible to avoid security problems.

Installation parameters frequently change. Please check the INSTALL file of CDRTOOL. It can be found as `INSTALL.txt` in the `doc` directory of CDRTool (`/var/www/CDRTool/doc/INSTALL.txt`).

LAB—Using CDRTool

Change your script to the final script including NAT traversal and RADIUS Billing.

```
#
# $Id: openser.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $
#
# simple quick-start config script
# Please refer to the Core CookBook at http://www.openser.org/
dokuwiki/doku.php
# for a explanation of possible statements, functions and parameters.
#

# ----------- global configuration parameters ------------------------

debug=3            # debug level (cmd line: -dddddddddd)
fork=yes
log_stderror=no    # (cmd line: -E)
children=4
port=5060

# ------------------ module loading ----------------------------------
#set module path
mpath="//lib/openser/modules/"

# Uncomment this if you want to use SQL database
#loadmodule "mysql.so"

loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
```

```
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"
loadmodule "lcr.so"
loadmodule "avpops.so"
loadmodule "xlog.so"
loadmodule "nathelper.so"
loadmodule "mediaproxy.so"
loadmodule "acc.so"


# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"

# ----------------- setting module-specific parameters ---------------

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("registrar", "received_avp", "$avp(i:42)")
modparam("usrloc", "db_mode", 2)
modparam("usrloc", "nat_bflag", 4)
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|usrloc","db_url","mysql://
openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/
openser")
modparam("avpops", "avp_table", "usr_preferences")
modparam("nathelper","rtpproxy_disable", 1)
modparam("nathelper","natping_interval", 0)
modparam("nathelper","received_avp", "$avp(i:42)")
modparam("mediaproxy","natping_interval",20)
modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")
modparam("mediaproxy","sip_asymmetrics","/etc/openser/sip-clients")
modparam("mediaproxy","rtp_asymmetrics","/ect/openser/rtp-clients")
modparam("acc", "radius_config",      "/etc/radiusclient-ng/
radiusclient.conf")
```

```
modparam("acc", "radius_flag",        2)
modparam("acc", "radius_missed_flag", 3)
modparam("acc", "radius_extra", "User-Name=$Au; \
                                 Calling-Station-Id=$from; \
                                 Called-Station-Id=$to; \
                                 Sip-Translated-Request-URI=$ruri; \
                                 Sip-RPid=$avp(s:rpid); \
                                 Source-IP=$si; \
                                 Source-Port=$sp; \
                                 Canonical-URI=$avp(s:can_uri); \
                                 Billing-Party=$avp(s:billing_party); \
                                 Divert-Reason=$avp(s:divert_reason); \
                                 X-RTP-Stat=$hdr(X-RTP-Stat); \
                                 Contact=$hdr(contact); \
                                 Event=$hdr(event); \
                                 SIP-Proxy-IP=$avp(s:sip_proxy_ip); \
                                 ENUM-TLD=$avp(s:enum_tld)")

# ------------------------  request routing logic -------------------

# main routing logic

route{

    #
    # -- 1 -- Request Validation
    #
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
    };

    if (msg:len >=  2048 ) {
        sl_send_reply("513", "Message too big");
        exit;
    };

    #
    # -- 2 -- Routing Preprocessing
    #
    ## Record-route all except Register
    ## Mark packets with nat=yes
    ## This mark will be used to identify the request in the loose
    ## route section
```

```
if(!is_method("REGISTER")){
    if(nat_uac_test("19")){
        record_route(";nat=yes");
    } else {
        record_route();
    };
};

##Loose_route packets
if (has_totag()) {
 #sequential request withing a dialog should
 # take the path determined by record-routing
 if (loose_route()) {
     if(method=="BYE") {
         #Account BYE transactions
         setflag(2);
     };
     #Check authentication of re-invites
     if(method=="INVITE" && (!allow_trusted())) {
           if (!proxy_authorize("","subscriber")) {
            proxy_challenge("","1");
            exit;
         } else if (!check_from()) {
            sl_send_reply("403", "Forbidden, use From=ID");
            exit;
         };
       };
       if(method=="BYE" || method=="CANCEL") {
           end_media_session();
       };
       ##Detect requests in the dialog behind NAT,flag with 6
       if(nat_uac_test("19") || search("^Route:.*;nat=yes")){
           append_hf("P-hint: LR|fixcontact,setflag6\r\n");
           fix_contact();
           setbflag(6);
       };
       route(1);
    } else {
        sl_send_reply("404","Not here");
    }
    exit;
}

#CANCEL processing
```

```
        if (is_method("CANCEL")) {
          if (t_check_trans()) {
                  end_media_session();
                  t_relay();
               };
               exit;
        }

    t_check_trans();
    #
    # -- 3 -- Determine Request Target
    #
    if (method=="REGISTER") {
        route(2);
    } else {
        route(3);
    };
}


route[1] {
    #
    # -- 4 -- Forward request to target
    #
    # Forward statefully
    t_on_reply("1");
    t_on_failure("1");
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

route[2] {
    ## Register request handler
    if (is_uri_host_local()) {
        if (!www_authorize("", "subscriber")) {
            www_challenge("", "1");
            exit;
        };

        if (!check_to()) {
            sl_send_reply("403", "Forbidden");
            exit;
```

```
            };

            if(!search("^Contact:[ ]*\*") && client_nat_test("7")) {
                setbflag(6);
                fix_nated_register();
                force_rport();
            };
            save("location");
            exit;

        } else if {

            sl_send_reply("403", "Forbidden");

        };
    }

    route[3] {
        ## Requests handler
        if (method=="INVITE") {
         # Account INVITE packets
             setflag(2);
             # Account Missed calls
             setflag(3);
             # Radius Extra
             $avp(s:sip_proxy_ip)="127.0.0.1";
        };
        if (is_from_local()){
            # From an internal domain -> check the credentials and the FROM
            if(!allow_trusted()){
                if (!proxy_authorize("","subscriber")) {
                    proxy_challenge("","0");
                    exit;
                } else if(!check_from()) {
                   sl_send_reply("403", "Forbidden, use From=ID");
                   exit;
             };
            };

            if (client_nat_test("3")) {
                append_hf("P-hint: route(3)|setflag7,forcerport,
                          fix_contact\r\n");
                setbflag(7);
```

```
            force_rport();
            fix_contact();
        };

        #unconditional call forward
        if(avp_db_load("$ru/username","$avp(s:callfwd)")) {
            avp_pushto("$ru", "$avp(s:callfwd)");
            route(1);
            exit;
        }

        consume_credentials();

        #verify aliases, if found replace R-URI.
        lookup("aliases");

        if (is_uri_host_local()) {
            # -- Inbound to Inbound
            route(10);
        } else {
            # -- Inbound to outbound
            route(11);
        };

    } else {

        #From an external domain ->do not check credentials
        #Verify aliases, if found replace R-URI.
        lookup("aliases");
            if (is_uri_host_local()) {
                #-- Outbound to inbound
                route(12);
            } else {
                # -- Outbound to outbound
                route(13);
            };
    };
}

route[4] {
    # routing to the public network
    if (!load_gws()) {
        sl_send_reply("503", "Unable to load gateways");
    exit;
```

```
        }

        if(!next_gw()){
            sl_send_reply("503", "Unable to find a gateway");
        exit;
        }
        t_on_failure("2");
        if (!t_relay()) {
            sl_reply_error();
        };
        exit;
    }

    route[6] {
        #
        # -- NAT handling --
        #
        if (isbflagset(6) || isbflagset(7)) {
            append_hf("P-hint: Route[6]: mediaproxy \r\n");
            use_media_proxy();
        };
    }

    route[10] {
        #from an internal domain -> inbound
        #Native SIP destinations are handled using the location table
        #Gateway destinations are handled by regular expressions
        append_hf("P-hint: inbound->inbound \r\n");

        if (uri=~"^sip:[2-9][0-9]{6}@") {
            if (is_user_in("credentials","local")) {
                prefix("+1305");
                route(6);
                route(4);
                exit;
            } else {
                sl_send_reply("403", "No permissions for local calls");
                exit;
            };
        };

        if (uri=~"^sip:1[2-9][0-9]{9}@") {
            if (is_user_in("credentials","ld")) {
```

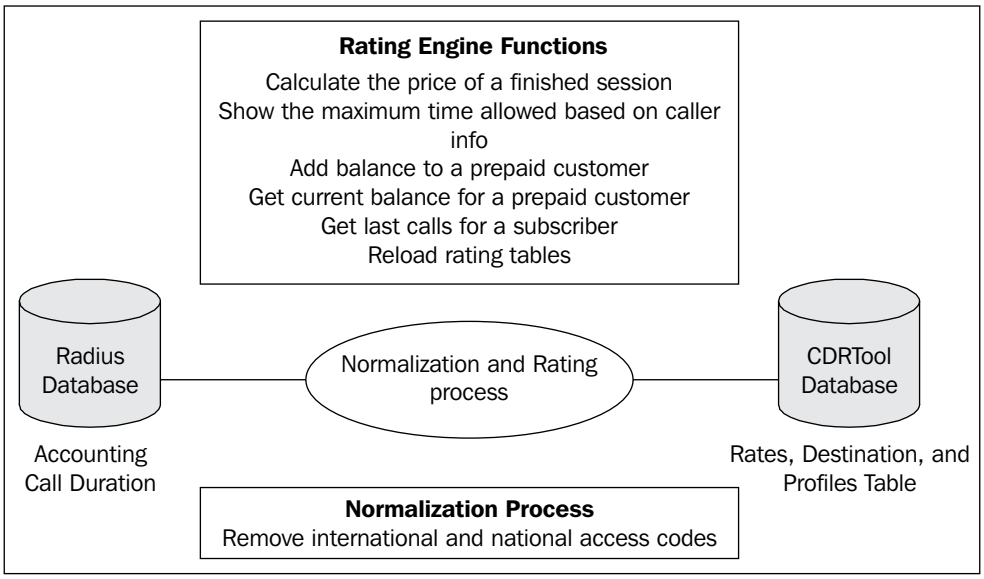
```
            strip(1);
            prefix("+1");
            route(6);
            route(4);
            exit;
        } else {
            sl_send_reply("403", "No permissions for long distance");
            exit;
        };
    };

    if (uri=~"^sip:011[0-9]*@") {
        if (is_user_in("credentials","int")) {
            strip(3);
            prefix("+");
            route(6);
            route(4);
            exit;
        } else {
            sl_send_reply("403", "No permissions for
                        international calls");
        };
    };

    if (!lookup("location")) {
        if (does_uri_exist()) {
            ## User not registered at this time.
            ## Use the IP Address of your e-mail server
            revert_uri();
            prefix("u");
            rewritehostport("192.168.1.171"); #Use the IP address of
                                              your voicemail server
            route(6);
        route(1);
        } else {
            sl_send_reply("404", "Not Found");
        exit;
        }
        sl_send_reply("404", "Not Found");
        exit;
    };
    route(6);
    route(1);
}
```

```
route[11] {
     # from an internal domain -> outbound
     # Simply route the call outbound using DNS search
     append_hf("P-hint: inbound->outbound \r\n");
     route(1);
}

route[12] {
     # From an external domain -> inbound
     # Verify aliases, if found replace R-URI.
     lookup("aliases");
     if (!lookup("location")) {
          sl_send_reply("404", "Not Found");
          exit;
     };
     route(1);
}

route[13] {
     #From an external domain outbound
     #we are not accepting these calls
     append_hf("P-hint: outbound->inbound \r\n");
     sl_send_reply("403", "Forbidden");
     exit;
}

failure_route[1] {
        ##--
        ##-- If cancelled, exit.
        ##--
        if (t_was_cancelled()) {
            exit;
        };
        ##--
        ##-- If busy send to the e-mail server, prefix the "b"
        ##-- character to indicate busy.
        ##--
        if (t_check_status("486")) {
            revert_uri();
            prefix("b");
            rewritehostport("192.168.1.171");
            append_branch();
            route(1);
            exit;
```

```
        };
        ##--
        ##-- If timeout (408) or unavailable temporarily (480),
        ##-- prefix the uri with the "u"character to indicate
        ##-- unanswered and send to the e-mail
        ##-- sever
        ##--
        if (t_check_status("408") || t_check_status("480")) {
           revert_uri();
           prefix("u");
           rewritehostport("192.168.1.171");
           append_branch();
           route(1);
           exit;
        };
}

failure_route[2] {
    if(!next_gw()) {
        t_reply("503", "Service not available, no more gateways");
        exit;
    };
    t_on_failure("2");
    t_relay();
}


onreply_route[1] {
   #
   #-- On-replay block routing --
   #
   if (client_nat_test("1")) {
              append_hf("P-hint: Onreply-route - fixcontact \r\n");
              fix_contact();
   };

   if ((isbflagset(6) || isbflagset(7)) &&
                              (status=~"(180)|(183)|2[0-9][0-9]")) {
              if (search("^Content-Type:[ ]*application/sdp")) {
             append_hf("P-hint: onreply_route|usemediaproxy \r\n");
          use_media_proxy();
           };
       };
       exit;
}
```

CDRTool Architecture

The tool uses two databases. The RADIUS database, more specifically, the table `radacct` and its own database named CDRTOOL. The main process for CDRTool is the rating engine (`/var/www/CDRTool/scripts/ratingEngine.php`) This software is responsible for taking the duration of the call and, according to a series of parameters, attributing a price to this particular call. The software exposes some functions allowing you to have online access to data such as Current Balance, essential for pre-paid billing. Let's explain in this section how the calls are rated.



How CDRTool Rates a Call

The software rates the call instantaneously, based on a rating plan, from multiple data sources such as Asterisk, Cisco, and OpenSER. CDRTool use the RADIUS database, more specifically the `radacct` table that contains duration, calling and called party, and media information (if you are using MediaProxy). The tool calculates the price of the session in real time and saves it to the `radacct` table.

Rates are directly related to profiles linked to the time of the day, day of week, and holidays. A call can span multiple profiles and be correctly rated. Each customer can be assigned a dedicated rating plan.

The CDRTool rating follows this path:

Step 1: Determination of the billing party

CDRTool identifies the rating plans based on the following order:

- SIP account user@domain
- SIP domain of the SIP account
- Source IP of the session
- Default

To do this it uses the field `radacct.username`.

Step 2: Determination of the destination

CDRTool identifies the destination used in the rating process in the order:

- Canonical-URI
- SIPTranslated RequestURI
- CalledStationID

Step 3: PSTN Rating

Look up in the billing profile in `cdrtool.billing_customers` table in the order below and use the profile or profiles matching the week day and hour.

- Subscriber
- Domain
- Gateway

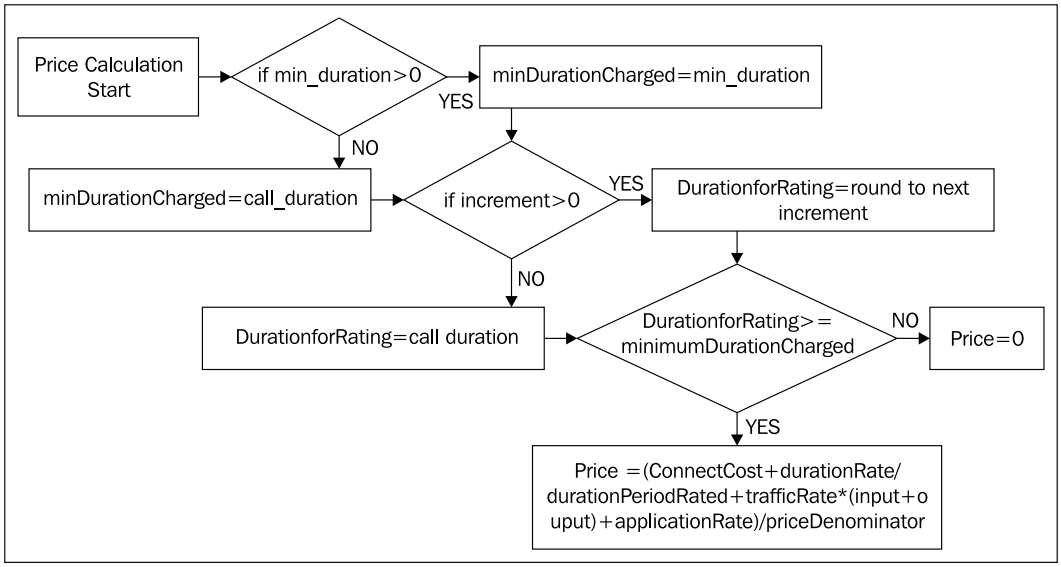
Using the profile determine the Rate ID(s) used for a specific destination and application (only audio is supported).

Calculate the price based on duration or the duration array based on the following settings. These settings can be changed defining this array inside the `global.inc` file:.

- "priceDenominator" => 10000, // e.g. 1 Eur = 10000 units
- "priceDecimalDigits" => 4, // how many digits to round the prices to
- "minimumDurationCharged" => 0, // Only calls greater than this duration will be charged
- "durationPeriodRated" => 60, // the prices from the rating table are calculated per this period

- "trafficSizeRated" => 1024, // same as above but for data traffic
- "reportMissingRates" => 0, // send email notifications if rates are missing from the ratingEngine
- "minimumDuration" => 0 // minimum duration to rate, if call duration is shorter the price is zero

Price denominator is used to calculate the cents. A rate of 200 with a price denominator of 10000 is equivalent to 0.02 (2 cents). Minimum duration charged is used to calculate the minimum price charged. Sometimes, VoIP providers call this 60/6 billing. The Minimum duration charged is 60 seconds, but the increment rated (durationPeriodRated) is per 6 seconds. You can apply connect and traffic fees as well. The system is very complete for post-paid billing.



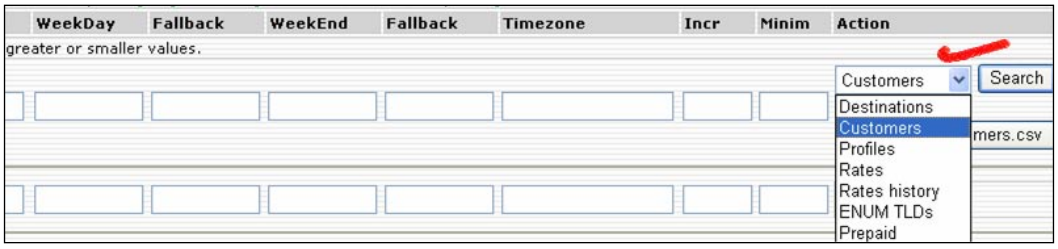
Step 4: Save the calculation in the CDR (`radacct` table). All the calculated prices are saved in the `radacct` table. Check using phpMyAdmin.

More details can be found at the rating documentation in the `doc` directory (`/var/www/CDRTool/doc/RATING.txt`).

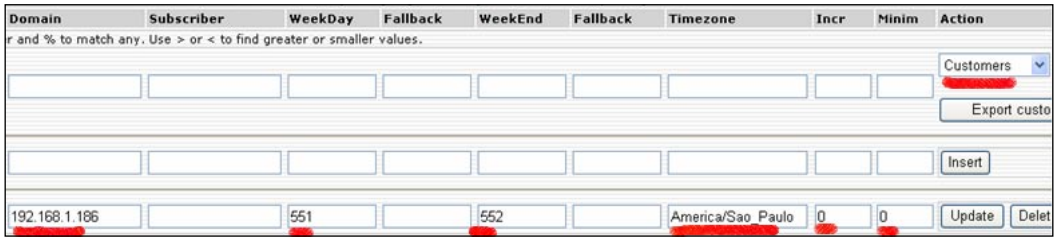
Lab—Creating and Applying a Rating Plan

Step 1: Log in to the CDRTool and go to the rating tables menu.

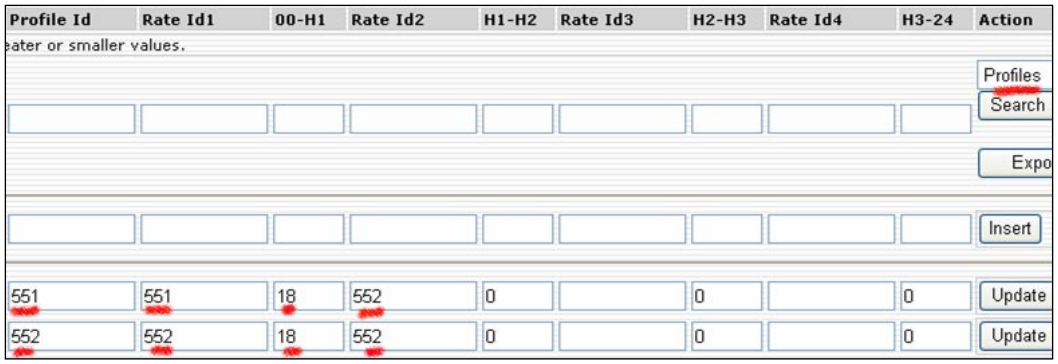
Step 2: Choose the **Customers** database.



Step 3: Insert your domain as **Customers**. All calls coming from this domain will be rated according to the profiles inserted. The first profile (551) will be used to rate calls on week days and the second (552) at the week-ends. Don't forget to set your timezone and to reload the tables.

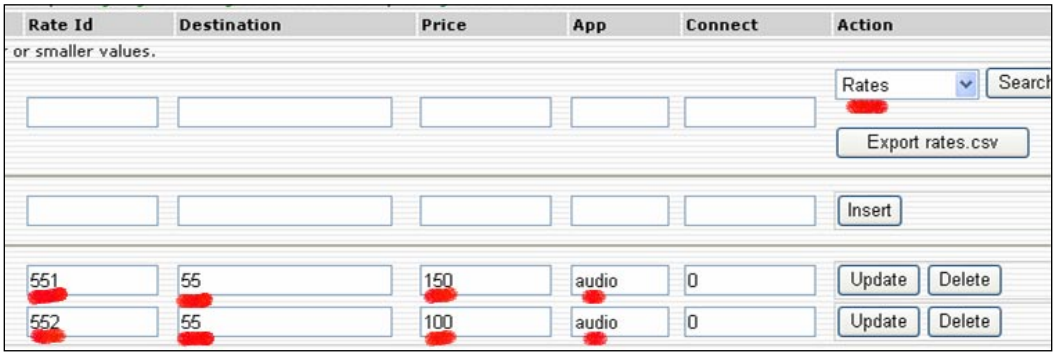


Step 4: Insert a profile to rate the calls based on different hours of the day.



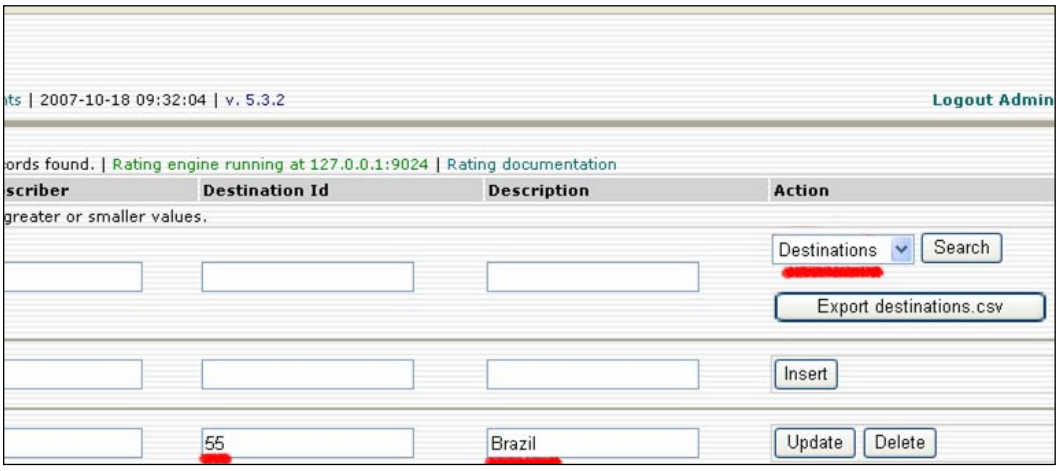
The first profile (551) is for weekdays and the second (552) is for week-ends. In this table you can assign different rates for different hours of the day. Up to four intervals can be assigned. In the preceding example you have assigned the **RateID** 551 from 00:00-18:00 on weekdays, and **Rate ID** 552 after 18:00. For week-ends **Rate ID** 552 from 00:00 to 18:00 and **Rate ID** 552 from 18:00-23:59.

Step 5: Assign prices to rate IDs.



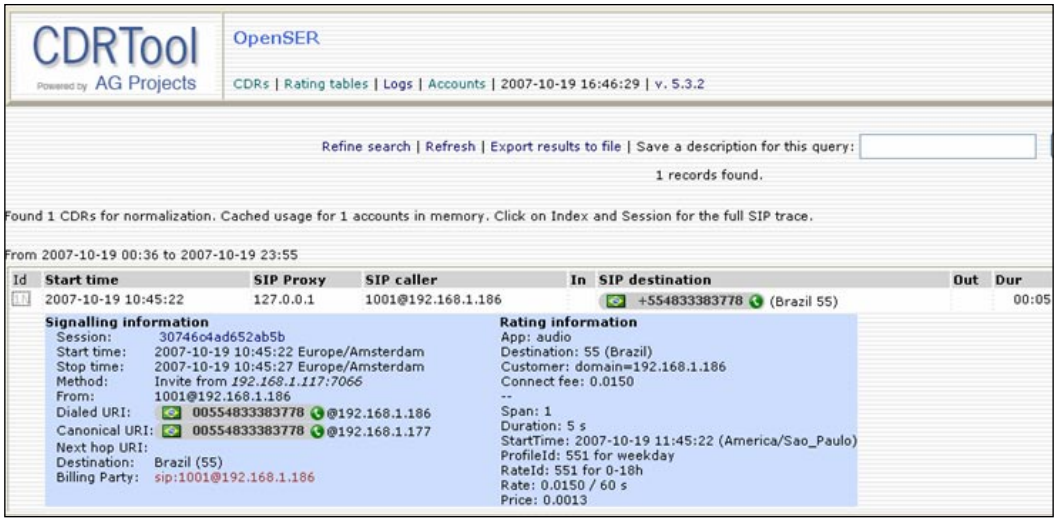
Create two new **Rate ID**s (551) and (552). **Rate ID**s are independent of profile IDs; in this case both were set with the same numbers to make it easier. In the example above, you are assigning a rate of 150 (1.5 cents) to each call during normal hours and 100 (1 cent) to calls after 18:00. The application is always audio and the connect parameter allows you to apply connect fees.

Step 6: Assign names for the destination IDs.



You can also use the sample tables using the utility `importRatingTables.php` located in `/var/www/CDRTool/setup/scripts`. See instructions in the `RATING.txt` file in the `doc` directory.

Step 7: Make a call to a destination 00554834567890 and check the CDRs.



Summary

In this chapter we have learned how to implement one of the most sensitive components of a VoIP provider, the accounting. Accounting can be done in MySQL, RADIUS, and Diameter. We have installed and tested accounting in MySQL and RADIUS. Besides we have learned how to install and use a rating tool called CDRTool. This tool has a very important role in the VoIP provider, rating the calls and calculating the price.

11
Troubleshooting Tools

After installing the whole system, now is time to start a stress test to check your configuration. We will use the SIPp utility to test our system. Before we use SIPp it is important to recognize helpful tools such as `openserctl moni`, `openserctl online`, and the SipTrace module. Packet capturing tools are used on a daily basis in a voice provider, so we will need to learn how to use Wireshark, Tshark, and `ngrep`. Later we will check `sipsak`, which calls itself a Swiss army knife for SIP. We can use it along with Nagios and Monit to monitor OpenSER.

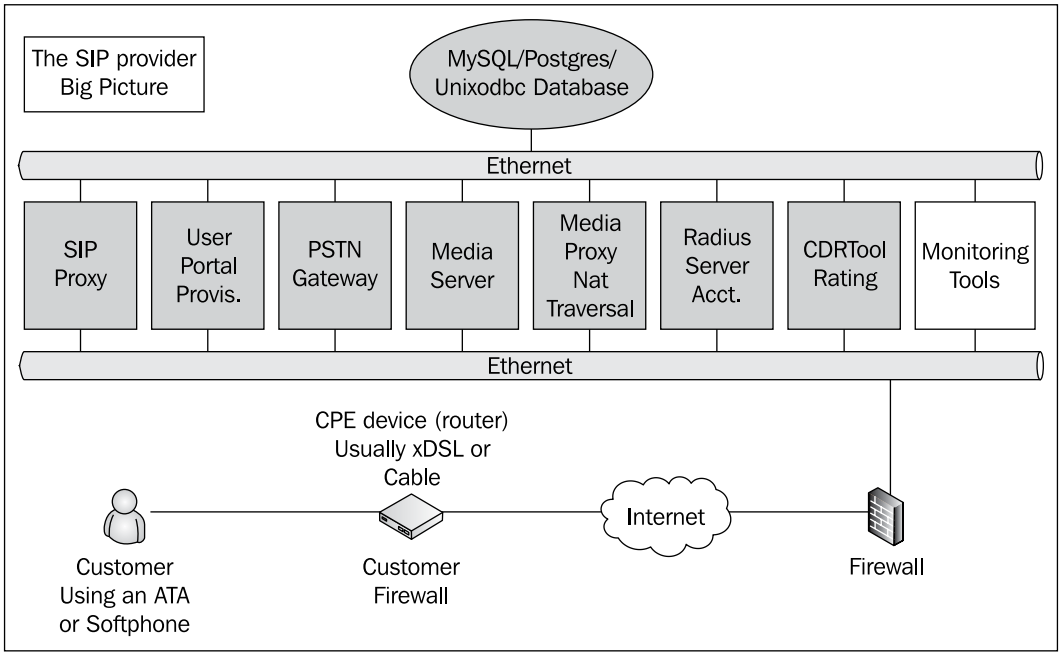
Objectives

By the end of this chapter you will be able to:

- Recognize the main tools used to manage OpenSER
- Understand how to use built-in tools such as `openserctl`
- Capture and analyze packets using Ethereal and `ngrep`
- Troubleshoot customer signaling using SipTrace
- Stress test OpenSer using SIPp
- Test OpenSer using `sipsak`

Where Are We?

In the last chapter, we finished the installation of the VoIP provider. Now it is time to start production and operation. On a daily basis, you will need some tools to deal with customers complaining about connectivity and voice quality issues. In this chapter we will show some of the best tools to help you with this task.



Built-in Tools

The built-in tools are openserctl and SIPTRACE.

Openserctl was created to monitor online users using openserctl online. You can also use the command openserctl ping to ping a registered user and finally, check the status of the server using openserctl moni.

Be sure that your `mi_fifo` module is correctly configured or openserctl won't work for OpenSER statistics. Please, check your `openserctlrc` file to see if the FIFO is pointing to the file `/tmp/openser_fifo`.

Issuing `openserctl moni` will bring the output below:

```
Server:: OpenSER (1.2.2-notls (i386/linux))
200 OK
Now:: Sat Nov  3 03:58:25 2007
Up since:: Sat Nov  3 03:55:41 2007
Up time:: 164 [sec]

Transaction Statistics:
200 OK
tm:UAS_transactions = 0
```

```
tm:UAC_transactions = 0
tm:inuse_transactions = 0

Stateless Server Statistics:
200 OK
sl:sent_replies = 0
sl:sent_err_replies = 0
sl:received_ACKs = 0

UsrLoc Stats:
200 OK
usrloc:registered_users = 0
usrloc:location-users = 0
usrloc:location-contacts = 0
usrloc:location-expires = 2
usrloc:aliases-users = 0
usrloc:aliases-contacts = 0
usrloc:aliases-expires = 0
```

This built-in tool prints statistics of the TM, SL, and USRLOC modules. You can spot how many transactions are in use and how many were done. Messages sent bring you some information about errors occurring. Finally, the `usrloc` statistics allow you the check the health of the REGISTER processes.

Packet Capture and Trace Tools

There are several packet capture and trace tools for OpenSER. One of the simplest is `ngrep` used throughout this book. TShark, former Tethereal, is a nice tool if you have a server without a GUI. With Tshark you can export the captured packets to analyze on Wireshark. Trace tools such as the SIPTRACE module are cool too. However they can impact the performance of your system when enabled. The SIPTRACE module logs the inbound and outbound traffic passing through the proxy for a marked transaction to a database.

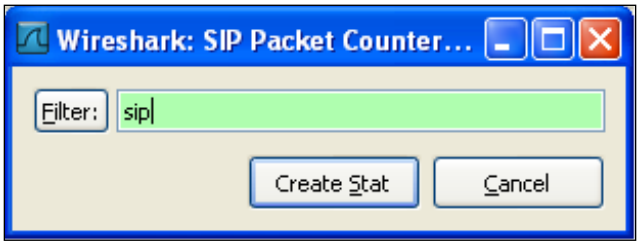
TShark, Wireshark

Wireshark (former Ethereal) is the most used protocol analyzer available in the market and it is GPL licensed. Usually you don't have a GUI (graphical user interface) in your server, but you can still capture the packets using the text version of Wireshark known as TShark. We often use `ngrep`, because it is very simple and light. To teach you exactly how to use a protocol analyzer is beyond the scope of this material; however, we will give you some tips on analyzing SIP and RTP packets.

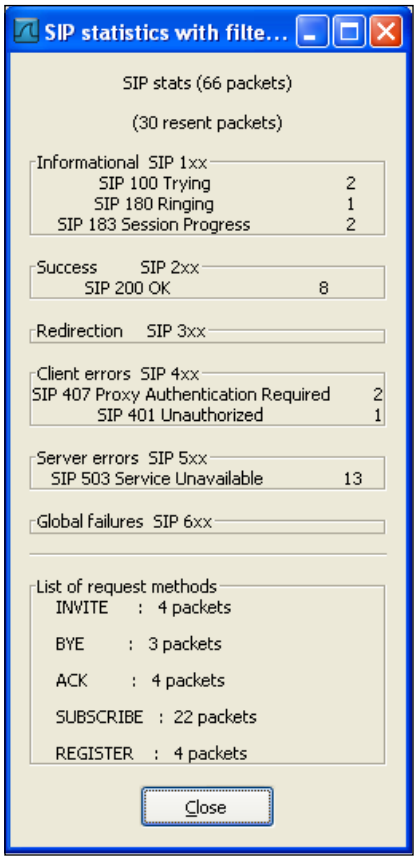
Wireshark has some special statistics for SIP and RTP. After loading the captured packets, you can start analyzing statistics for the SIP Protocol. Let's try; in the Wireshark menu select:

Statistics | sip

It will ask you for a **Filter**; use **sip**.



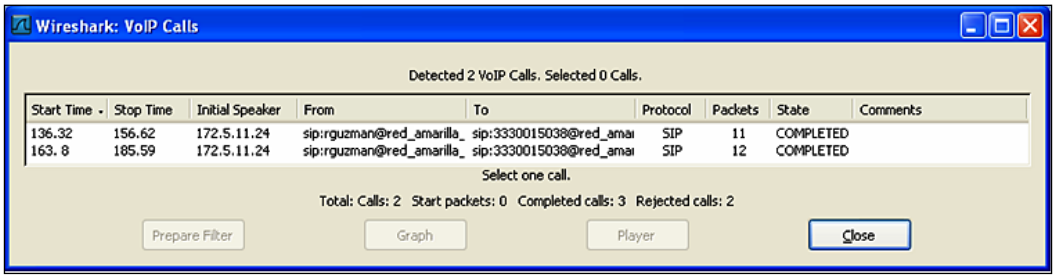
Press the **Create Stat** button.



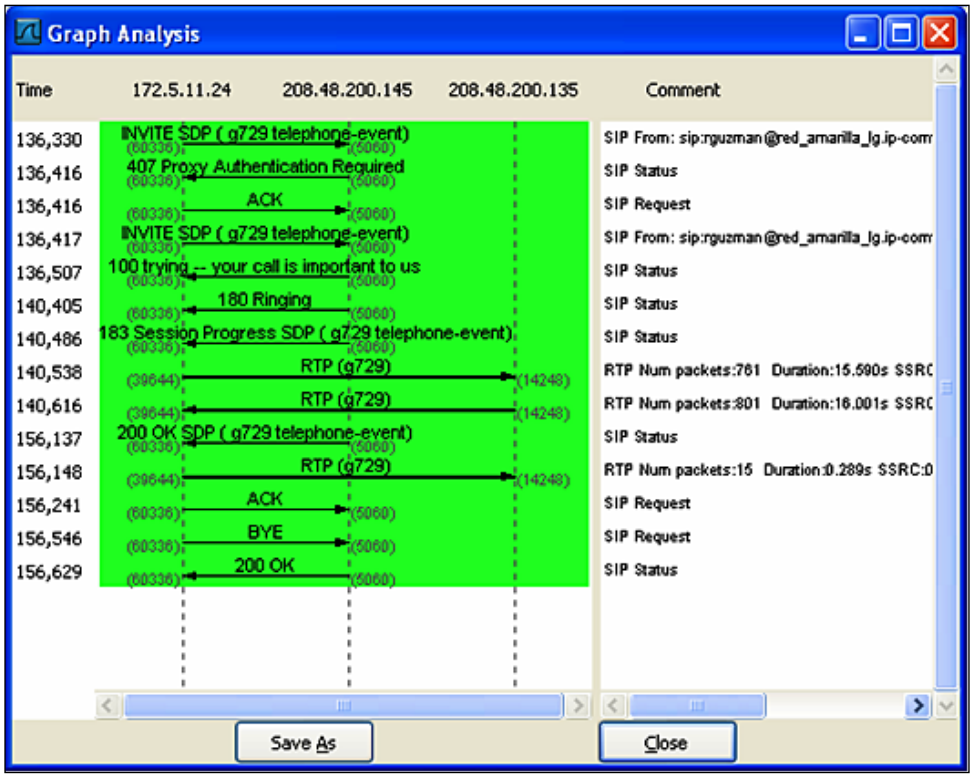
It is a nice trick. You can now check general statistics about your SIP messages. Well, this is not our best trick, but it can help to spot an abnormal behavior.

Let's go to the second trick, to graph the SIP dialog. In the Wireshark menu select **Statistics | voip calls**

You will see the following screen:

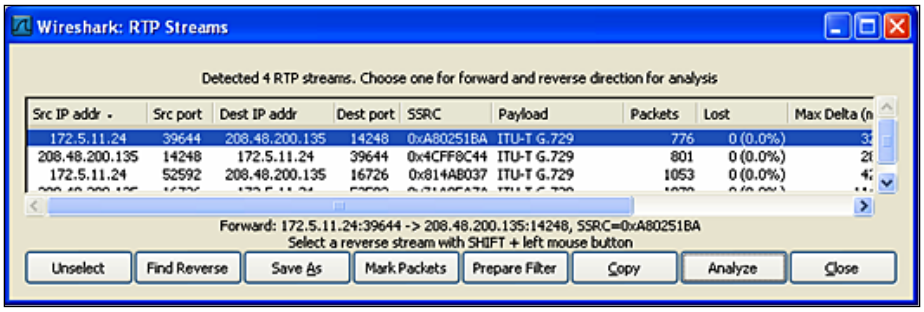


In this screen you can select the call you want to graph. After selecting the call press the **Graph** button.

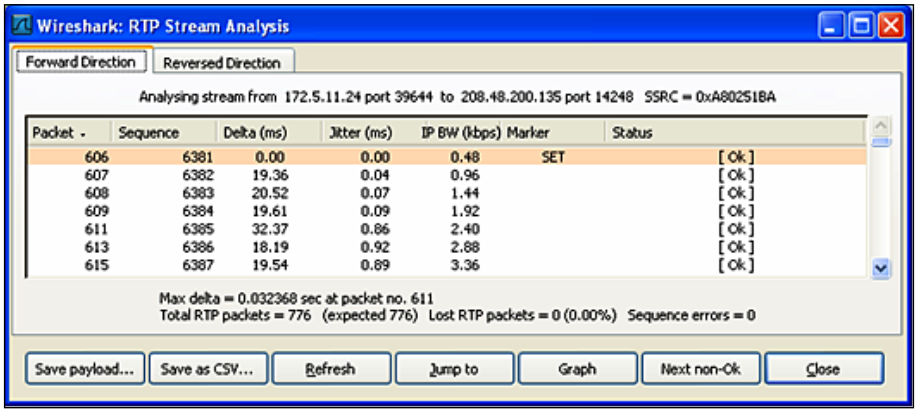


You will receive this amazing graph with the SIP dialog. Now you can spot specific problems in a single dialog. You can even play a call in the previous menu if it is coded using G.711 Alaw or Ulaw. Very nice indeed, don't you think? We still have some nice tricks in our bag, so please hold on and check the next.

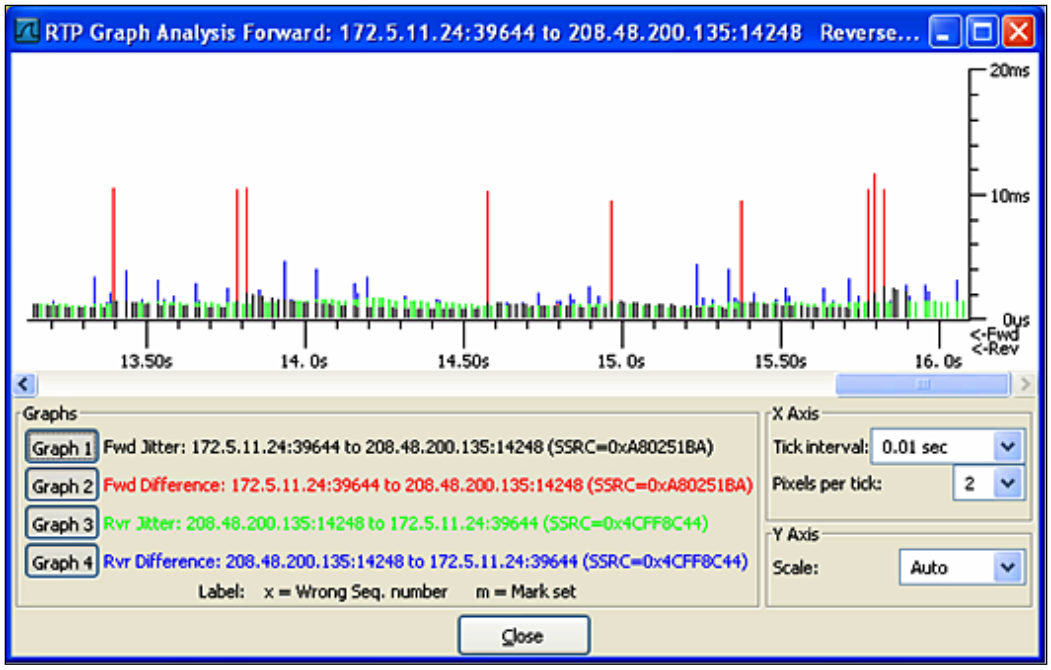
Well, now let's check the RTP packets. After all, RTP packets will determine the voice quality. There is no single recommendation; we consider having a good voice quality to mean the latency is below 150 ms (the one-way equivalent to a round trip time of 300 ms), jitter below 20 ms, and packet loss below 3%. You can have a good voice quality with higher latencies. However, the interactivity of the conversation deteriorates after 150 ms. Sure you can have voice over IP in satellite environments where the latency is typically 300 ms. The quality of the voice is affected by jitter much more than latency. Jitter usually causes distortion in the audio. Jitter buffers in UAs reduces this problem, but sacrificing some latency. However the interactivity is not as good as when you have a lower value. Check to see what works for you and use Wireshark to keep the voice quality within in your own standards. To help you in this task let's use the following statistics. In the Wireshark menu select **Statistics | RTP | Stream Analysis**.



Select a stream to analyze. Use *Shift-left* to select a reverse stream.



Now you can analyze packet by packet the jitter, latency (delta), IP bandwidth, and packet loss of your RTP streams. You can even graph the RTP stream.

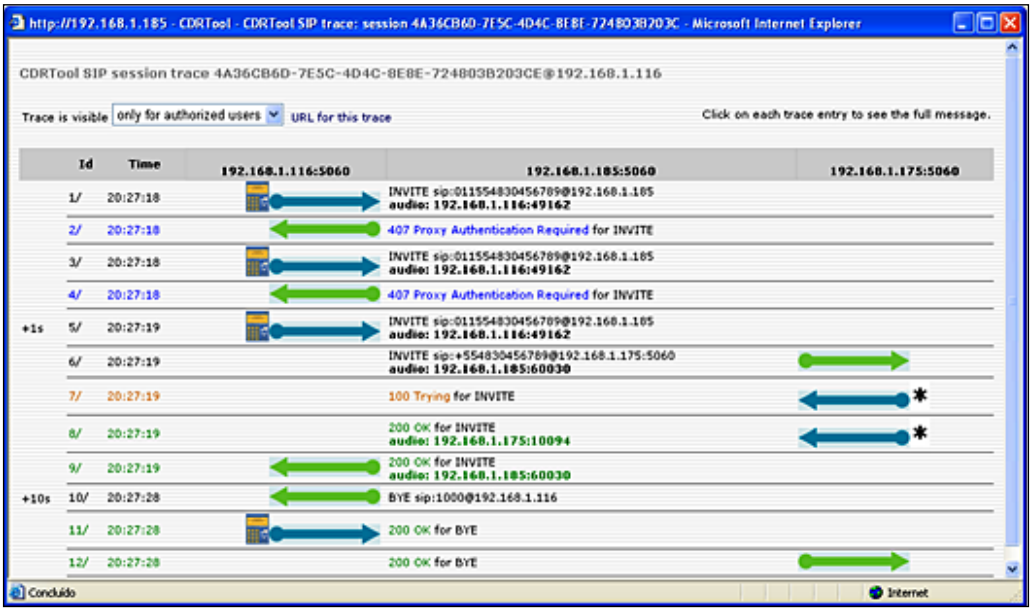


In our case we can see by the graph that our jitter is below 5 ms in both directions. The difference is the inter-arrival time between packets.

SipTrace

OpenSER has a module called SipTrace, which allows you to store SIP messages in the database. The CDRTool web interface is used as the user interface to trace the calls directly from the billing data. The module is very simple to use; just load the module and set the transactions you want to record using a specific flag defined in the module parameter `trace_flag`. You will probably not want to record all messages to the database because of the overhead.

You can set the module parameter `trace_on=0` and manually start and stop tracing the messages using `openserctl fifo sip_trace on` and `openserctl fifo sip_trace off`. Don't forget to set the `db_url` parameter of the `sip_trace` module. The software CDRTool has a nice feature. It displays graphically the data collected by the SIPTRACE module in the database. In the CDRTool you will see a screen similar to that below:



Stress Testing Tools

Now we will present some tools to stress test you OpenSER server before going to production. The first tool is `sipsak` (`www.sipsak.org`) and the second is SIPp (`sip.sourceforge.net`).

Sipsak

Sipsak is a command-line tool used by SIP administrators. It is used to run simple tests against the SIP server. It is good too for checking the security of the server, because you can create the SIP request exactly the way you want. Details can be found at `www.sipsak.org`. Let's show an example on how to use it. Install it using:

```
apt-get install sipsak
```

Example of use:

```
sipsak -U -s sip:1000@192.168.1.185 -a 1000 -W 1 -vvvvv
```

This command tests a REGISTER packet againt the SIP proxy and returns the Nagios code number 1. You can use Nagios (a utility to monitor servers) to monitor OpenSER using an effective transaction instead of simply pinging it.

SIPp

To explain each detail of SIPp is beyond the scope of this material. The idea here is to give you an overview of SIPp and teach you how to start. Give enough time to test your platform; you will need a lot of time to build a test lab with several UACs and UASes and interpret the results.

SIPp is a tool for traffic generation and stress testing for SIP. It is a good tool to submit traffic to your SIP server and test it before going to the production phase. It establishes and releases multiple calls with methods such as INVITE and BYE. The call rate can be adjusted dynamically. More information can be found at its web site: `sipp.sourceforge.net/doc/reference.html`.

Let's see some examples with real-world scenarios of what we can do with this tool.

Installing SIPp

Install the dependencies:

```
apt-get install g++
apt-get install ncurses-dev
apt-get install openssl-devel
apt-get install libssl-dev
apt-get install libnet1-dev
apt-get install libpcap0.8-dev
```

Download and decompress the `sipp` source file:

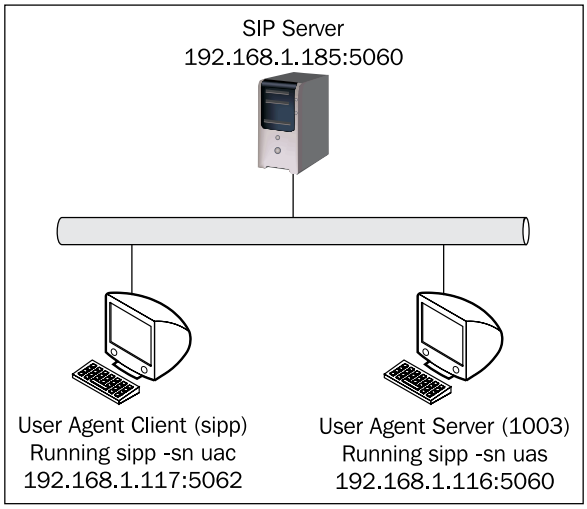
```
wget http://downloads.sourceforge.net/sipp/sipp-2.0.1.src.tar.gz
tar -xzvf sipp-2.0.1.src.tar.gz
```

Compile and run:

```
make
./sipp
```

Stress Test—The SIP Signaling

To stress test OpenSER you will need to have a user agent client, a user agent server, and the SIP proxy itself. See the diagram below:



You will need to register the user agent server manually adding a static mapping in the user location table. In the example below we are saying that user 1003 is at address 192.168.1.117 (where we started the UAS).

```
opensercctl ul add 1003 sip:1003@192.168.1.117:5060
```

To start the user agent server use:

```
./sipp -sn uas
```

It will show you a screen like that below:

```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
Port   Total-time  Total-calls  Transport
5060     139.31 s         4738  UDP

0 new calls during 1.000 s period      1 ms scheduler resolution
105 calls                              Peak was 662 calls, after 104 s
0 Running, 105 Paused, 0 Woken up
1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
----------> INVITE                 4512      1839                226

<---------- 180                    4512      1839
<---------- 200                    4512      2615      17
----------> ACK            E-RTD1  4272      0                   0

----------> BYE                    4390      1870                0
<---------- 200                    4390      1870
[   4000ms] Pause                  4390                          280
------------------------------ Sipp Server Mode -------------------------------
```

In the sample XML files of SIPp, record-routing is not supported. Please change the script accordingly. I have created an example named `openser.chapter11`, which I have used for these tests. You will have to manually handle the ACKs and BYEs.

To start the user agent client use:

```
./sipp -sn uac 192.168.1.185:5060 -s 1003 -ap 1000 -p 5062 -d 1000
```

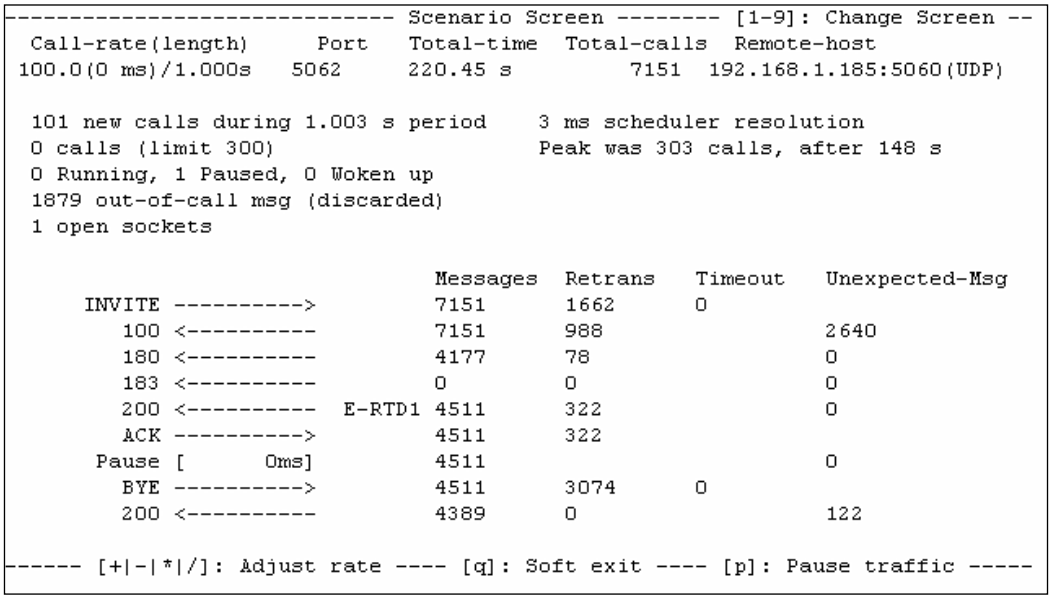
```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
 Call-rate(length)     Port   Total-time  Total-calls  Remote-host
100.0(0 ms)/1.000s   5062     220.45 s         7151  192.168.1.185:5060(UDP)

 101 new calls during 1.003 s period    3 ms scheduler resolution
 0 calls (limit 300)                    Peak was 303 calls, after 148 s
 0 Running, 1 Paused, 0 Woken up
 1879 out-of-call msg (discarded)
 1 open sockets

                                Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->         7151      1662      0
        100 <----------         7151      988                 2640
        180 <----------         4177      78                  0
        183 <----------         0         0                   0
        200 <----------  E-RTD1 4511      322                 0
        ACK ---------->         4511      322
      Pause [      0ms]         4511                          0
        BYE ---------->         4511      3074      0
        200 <----------         4389      0                   122

------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Increase the call rate using the + key until you start seeing retransmissions. In the case above, 100 simultaneous calls with MediaProxy support was enough. The screen above refers to my virtual machine. It handles from 75 to 125 simultaneous calls depending on the current load of the laptop running the VM.

Take care when testing with accounting turned on, you can fill up your hard disk easily.

Stress Test—The RTP Signaling

It is possible to test the RTP signaling using a combination of a UAS with the `rtp_echo` function combined with a UAC with the `pcap` function. See the detail in the SIP documentation.



```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
 Call-rate(length)     Port   Total-time  Total-calls  Remote-host
100.0(0 ms)/1.000s   5062     220.45 s         7151  192.168.1.185:5060(UDP)

 101 new calls during 1.003 s period    3 ms scheduler resolution
 0 calls (limit 300)                    Peak was 303 calls, after 148 s
 0 Running, 1 Paused, 0 Woken up
 1879 out-of-call msg (discarded)
 1 open sockets

                                Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->         7151      1662      0
        100 <----------         7151      988                 2640
        180 <----------         4177      78                  0
        183 <----------         0         0                   0
        200 <----------  E-RTD1 4511      322                 0
        ACK ---------->         4511      322
      Pause [      0ms]         4511                          0
        BYE ---------->         4511      3074      0
        200 <----------         4389      0                   122

------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

To start the UAS with RTP echo use a command similar to that below. Please adapt the scenario to your own situation before testing.

Example:

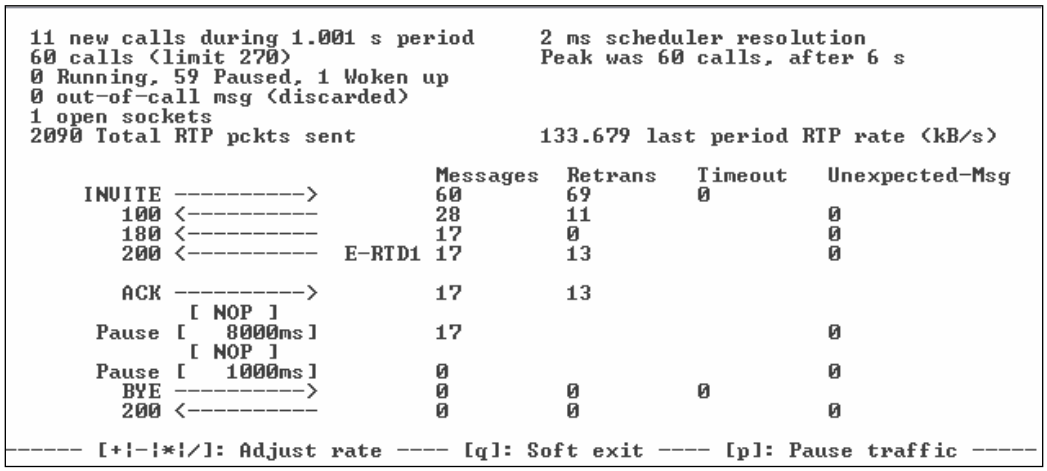
```
sipp -sf uas.xml -rtp_echo -mi 192.168.1.116 -mp 1000
```

See the data related to RTP packets echoed by the system in the screenshot.

To start the UAC with `pcap` use a command similar to :

```
sipp -sf uac_pcap.xml -s 1003 192.168.1.185:5060
```

You will see a screen like that below:



Testing MediaProxy

To test only MediaProxy using RTP you can use the RTP generator provided by AG Projects. You can check MediaProxy performance using:

```
./rtpgenerator  --g711 -count=30
```

To use several machines for testing use the following command in several machines.

```
./rtpgenerator -ip=ipof the mediaproxy  --g711 -count=50
```

Check the CPU using the Linux OS built-in application `top`. Use `./sessions.py` to see the sessions available.

Monitoring Tools

To monitor OpenSER you can use a set of utilities along with network monitoring tools. You can use Nagios along with `sipsak` to monitor real transactions such as REGISTER and INVITE. `monit` (`www.tildeslash.com`) is another tool you can use to monitor OpenSer from within. Using `monit` you can generate alerts about the status of the system and the OpenSER daemon. A good tutorial on how to set up `monit` with OpenSER can be found at `www.voip-info.org/wiki/view/OpenSER+And+Monit`.

Summary

In this appendix we have learned about the main tools to test and monitor OpenSER. It is wise to stress test OpenSER before starting the production phase. Packet capture tools such as Wireshark and `ngrep` are very important and will be used on a daily basis; be familiar, you will certainly need to use them. Finally `monit` can be used to monitor the processes and help you to keep OpenSER up and running.

After Words

The idea of the afterword came during the OpenSER summit. I was talking with a friend who told me about the new configuration file for OpenSER 1.3. It is a bit different than the one available for 1.2. Verifying the information, I found that the 1.2.3 was different too. So I decided to present here some of the best improvements seen in the new default files; basically the CANCEL handling (already integrated in the current script) and the method filtering in the `lookup()` function. RTPproxy is another missing part in the book. I have chosen MediaProxy, but now I have heard from some people that RTPproxy is in some cases 10 times faster than MediaProxy. Finally, I will describe some areas for improvement and further investigation.

What's New in Version 1.2.3

As I said, the newer version of OpenSER has a slightly different script than the one encountered in version 1.2.2; let's comment some of the modifications.

Cancel Handling

According to RFC3261, CANCEL requests have to be forwarded in the same way as INVITE requests. The code below is a shortcut. If an existing transaction exists, the CANCEL will be relayed automatically with the existing information about the transaction. The proxy needs to be operating in the stateful mode. In other words, using `t_relay()` instead of `forward()` to route the calls.

```
#CANCEL processing
if (is_method("CANCEL")) {
            if (t_check_trans()) t_relay();
            exit;
}
```

The `t_check_trans()` will take care of non-CANCEL and non-ACK requests belonging to a transaction, taking care of the retransmissions, if required. According to the documentation found at `www.openser.org`: non-CANCEL/non-ACK requests: if the request belongs to a transaction (it's a retransmision), the function will process the retransmission and will break or stop the script. The function returns false if the request is not a retransmission.

```
t_check_trans();
```

Blacklist is Disabled by Default

```
/* uncomment the next line to enable the auto temporary blacklisting
  of not available destinations (default disabled) */
#disable_dns_blacklist=no
```

Now in version 1.2.3 the DNS blacklist is disabled by default. I believe this will avoid a lot of confusion with messages "473 Filtered Destination".

Method Filtering

The new script uses a concept called method filtering. The `lookup()` function will return different values in different situations. The values are shown below:

According to the documentation the return codes are:

- 1—contacts found and returned
- -1—no contact found
- -2—contacts found, but method not supported
- -3—internal error during processing

The code below shows how to implement method filtering:

```
modparam("registrar", "method_filtering", 1)
if (!lookup("location")) {
                switch ($retcode) {
                        case -1:
                        case -3:
                                t_newtran();
                                t_reply("404", "Not Found");
                                exit;
                        case -2:
                                sl_send_reply("405", "Method Not
                                                Allowed");
                                exit;
                }
        }
```

The system is very clever. If you are trying to contact a user and this user does not support the method you are using (that is, for example SUBSCRIBE, PRACK), the system will return a message "405 Method Not Supported". Even if an internal error occurred, the system will create a new transaction and return a "404 Not Found". In the case of no contact found, the system will proceed as always with the message "404 Not Found".

Alias_DB

The lines below, even commented, suggest that now the best way to use Alias is directly from the database. There is a small performance penalty, but you can insert new aliases in real time. Aliases are often used for DID (Direct Inward Dial) relaying.

```
# apply DB based aliases (uncomment to enable)
 ##alias_db_lookup("dbaliases");
```

Branch_route

This is not brand new, but now it appears in the default configuration. It is used only with forking. If a message is forked to five destinations, the branch_route section will be processed five times. Sometimes this feature is used to filter some numbers before relaying to the final destination.

Migration from 1.2.2 to 1.2.3 and 1.3.1

For 1.2.3, you don't need to change anything. To use this book with version 1.3.1 you can find some instructions on this web page:

`http://www.openser.org/dokuwiki/doku.php/install:1.2.2-to-1.3.0`

Basically, you will need to implement two steps:

- The first one is to create the database using `openserctldb` instead of the old `openser_mysql.sh` in Chapter 5.
- The second one is to use the migrated script 1.3.1 (the script for Chapter 10 migrated to OpenSER 1.3.1).

Version 1.3.x is brand new at the time I'm writing this section. I recommend that you wait a little more time before using 1.3.x in a production environment. The OpenSER development team is releasing a newer version almost every 9 months. It is really hard to cope with their speed. So please check migration instructions if the ones above were insufficient.

Migrating the Script from Chapter 10 to openser 1.3.1

I had to change the following instructions to migrate the script openser.chapter10-2 to the version 1.3.1. After the changes I tested a call from two phones and two phones behind NAT. It worked fine, but I did not test the script thoroughly, so it is possible that you find a few issues.

Step 1: Migrate the `gw` table in the database (according to the web page cited above). The old table will be dropped.

```
mysql
use openser
drop table gw;
delete from version where table_name='gw'; (formatted as code in text)
insert into version values('gw', 5);
CREATE TABLE 'gw' ( 'id' int(10) unsigned NOT NULL auto_increment,
'gw_name' varchar(128) NOT NULL, 'grp_id' int(10) unsigned NOT NULL,
'ip_addr' varchar(15) NOT NULL, 'port' smallint(5) unsigned default
NULL, 'uri_scheme' tinyint(3) unsigned default NULL, 'transport'
tinyint(3) unsigned default NULL, 'strip' tinyint(3) unsigned default
NULL, 'prefix' varchar(16) default NULL, 'dm' tinyint(3) unsigned
NOT NULL default '1', PRIMARY KEY ('id'), UNIQUE KEY 'gw_name_idx'
('gw_name'), KEY 'grp_id_idx' ('grp_id') ) ENGINE=MyISAM DEFAULT
CHARSET=latin1;
```

Step2: Add the following lines to the script:

```
modparam("lcr", "dm_flag", 25)
modparam("lcr", "fr_inv_timer_avp", "$avp(i:704)")
modparam("lcr", "gw_uri_avp", "$avp(i:709)")
modparam("lcr", "ruri_user_avp", "$avp(i:500)")
modparam("lcr", "contact_avp", "$avp(i:711)")
modparam("^auth$|lcr", "rpid_avp", "$avp(i:302)")
```

Step 3: remove the following lines from the script:

```
modparam("nathelper","rtpproxy_disable", 1)
```

RTPProxy

RTPproxy cannot be ignored. I made a mistake to focus only on MediaProxy for this book. Some people claim that RTPproxy, developed in C, is in some cases ten times faster than MediaProxy, developed in Python. This is a huge difference and can't be ignored. It can be easily load balanced to achieve even greater scalability. However, MediaProxy has a resource able to fix the CDRs for dialogs with a missing BYE. RTPproxy does not have this same feature at the time I'm writing (I will present some ways to work around this). RTPproxy was developed by Maxim Sobolev and is now being actively maintained by Sippy Software Inc (`http://www.rtpproxy.org`).

Lab—Installing RTPProxy

Now let's quickly install the RTPproxy server.

Step 1: Donwload and compile the RTPproxy.

```
cd /usr/src
wget http://b2bua.org/chrome/site/rtpproxy-1.0.2.tar.gz
tar -xzvf rtpproxy-1.0.2.tar.gz
cd rtpproxy-1.0.2
./configure
make
make install
```

Step 2: Copy the file `openser.rtpproxy`, provided in the support area of the packtpub website, and copy it to `openser.cfg`. Restart OpenSER and test the new script.

Step 3: Start RTPproxy.

```
./rtpproxy -l 8.8.1.20 -s udp:127.0.0.1:7890
```

Remember to change the value 8.8.1.20 to the external address of your RTPproxy server.

The RTPproxy server *must* have a valid IP address (non-RFC1918)!!!

Areas for Further Investigation

There are some new modules for OpenSER 1.2 and 1.3. I didn't have enough time to research and write about them, but they are "on the radar". I will present an introduction to some of them below.

Carrier Route

Carrier Route first appeared in version 1.3.x. It is like a super LCR, allowing you to connect to a provider with a lot more features and speed compared to LCR. According to the documentation, it scales to several million users and is able to handle many thousand routes. It is a must-read for VoIP providers.

Dialog

The dialog module introduces dialog awareness to SIP proxy. The first practical use is to discover the number of active dialogs (calls). The module does not export any functions but makes available several statistics. Another use for this module is as a base for dialog information. Load the Dialog module in your script, make some calls and see the difference in the `openserctl moni` command output.

SIP Session Timers

SIP Session timers enhance the SIP protocol adding the capability to refresh SIP sessions resending repeated re-invites. The objective of this behavior is to establish a keep-alive mechanism. SIP proxies do not have control over the media. If a user does not send a BYE message (that is, disconnected from the network), the proxy do not have a mechanism to close this call and to generate the CDR (Call Detail Record) precisely.

There are, basically, three solutions for the problem of missing BYEs and the generation of the CDRs:

1. Generate CDRs only in the gateways and B2BUA (back to back user agents). They have an RTPTimeout mechanism able to finish the call, even without the BYE message. B2BUAs impose a performance penalty and require all the media to traverse your provider.
2. Use MediaProxy to fix the missing BYE session directly in the RADIUS Server. The software has a feature, allowing direct access to the RADIUS MySQL database. To activate this, check the configuration file `mediaproxy.ini`. Again, all the media has to traverse your provider.

3. Implement Session Timers on all your clients or in all your gateways. Check to see if your wholesale providers support it. Another way is to provide your phones or ATA (analog telephony adapters) with this configuration.

The last solution is the most scalable and does not break the philosophy of having peer-to-peer communications. It can save a lot of money on data access fees. The support for session timers is described in RFC4028. Recently, Asterisk announced support for RFC4028 integrated into Asterisk 1.6. SIP Session Timers need to be installed on UA or gateways, not necessarily on both.

SIP Peering

In Chapter 7, you saw how to terminate calls in a PSTN gateway. These days, most of time, you will terminate your calls in a VoIP Provider. To protect your gateways, you probably used a firewall preventing any other person from accessing the gateway's SIP channel directly. When you receive a call from the gateway, a trusted table is used to authorize the calls and your gateway is controlled inside your network. There are at least four ways to connect to a VoIP provider using OpenSER; let's see the pros and cons of the solutions:

1. Your VoIP provider authorizes your IP, and will bill according to the source IP. This is very common and I have seen several times. It is very simple, but it is definitely not the safest method. Some VoIP providers will require authentication.
2. If your VoIP provider requires user authentication, the standard way to do this is to use a B2BUA such as Asterisk. You configure Asterisk as an ordinary gateway, but instead of terminating calls in the PSTN it terminates the calls in the VoIP provider. Calls coming from a user will be bridged to the VoIP provider in the B2BUA server. This solution forces the traffic through your network and requires several additional servers. Check the UAC module of version 1.3; it has started to support authentication using "qop-auth". Maybe it is worth a try.
3. You can have an agreement with your VoIP provider to use a VPN to encrypt SIP traffic. You send the calls to the other domain, without the need of authentication. Authentication is performed at the network level using a VPN. RTP packets won't be encrypted, because there is a lot of overhead in this process and the traffic is peer-to-peer out of your control. You will have to force the SIP traffic through your proxy, using the outbound proxy setting in your phones.

4. You can also use TLS. You will receive traffic from your users using UDP or TCP, but you will forward this traffic to the provider using TLS. A pair of public/private keys will handle the authentication, and the SIP traffic is secure. Again RTP packets are untouched. The peer-to-peer nature of SIP is preserved in this solution too. You will have to force the traffic to your proxy, using the outbound proxy setting in your phones.

TLS Transport Layer Security

TLS has the potential to protect communications between parties in a SIP call. I believe it is one of the standards that will grow in the near future, mainly because of the adoption of NGNs (next generation networks) in the Telco companies.
It is possible to protect the communication from users and to VoIP providers. I recommend reading about this. TLS does not encrypt RTP packets, but it can be used to exchange keys for SRTP (secure RTP). SRTP can be used to encrypt SIP traffic.

Development

For those willing to develop and who don't want to use C, it is now possible to use Perl or Java. These programming languages are easier to learn and to debug than C.

PERL

Perl is a new module present in OpenSER 1.2.x. It may be used to develop applications interfacing directly to OpenSER. Check the documentation at `www.openser.org. http://www.openser.org/docs/modules/1.2.x/perl.html`.

WeSIP

WeSIP will allow you to develop SIP applications using Java and OpenSER. According to its web page "WeSIP is a SIP and HTTP Converged Application Server built on top of OpenSER".

Common Mistakes

I will present now a list of several common mistakes made when using this material. I observed this in the classroom when teaching this material in Portuguese. For anyone interested in training please check the calendar at `www.sermyadmin.org`.

Daemon Does Not Start

This is very common. What you have to do is:

1. First, run `openser -c` to check for syntax errors in the configuration file.
2. Check `/var/log/syslog` for errors in the loading of modules.

This usually solves most problems.

Another common mistake is to start using the init script and try to stop using `openserctl` and vice versa. Please don't mess with the starting commands. Rule of thumb:

- If you start using `/etc/init.d/openser start`, stop using `/etc/init.d/openser stop`.
- In the other hand, if you start using `openserctl start`, stop using `openserctl stop`.

Client Unable to Register

This is by far the most common problem. Check the following things:

1. **Is your domain inserted in the domain table** of the database? If you are using an IP address, please insert the IP addres to the database too.
2. **Plaintext or encrypted passwords?** You can't mix plaintext with encrypted passwords. There are two places to check:

Use the code below for plaintext passwords in the file `openser.cfg`:

```
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
```

In the file `openserctlrc`, be sure to leave commented:

```
#STORE_PLAINTEXT=0
```

On the other hand, if you want to use encrypted passwords use in `openser.cfg`:

```
modparam("auth_db", "calculate_ha1", 0)
modparam("auth_db", "password_column", "ha1")
```

In the file `openserctlrc`, be sure to leave uncommented:

```
STORE_PLAINTEXT=0
```

If you mix these things, you will end up not authenticating. The file `openserctlrc` regulates the creation of the users using `openserctl`. So, if you change this setting, the new settings will be valid only for new users. SerMyAdmin does not support encrypted passwords at this time.

Sending a Call to a Provider with Authentication

I have seen this several times. Some people think of OpenSER as if it was similar to an Asterisk Server. OpenSER is a SIP proxy, while Asterisk is a B2BUA. OpenSER is not able to authenticate ahead.

The standards for inter-domain communication are different than those for the communication between clients and servers. You cannot use your SIP proxy to authenticate to other SIP Proxies ahead. However there is a small hack. There is a module called UAC (User Agent Client) that allows you to mangle from and to fields and authentication. Until version 1.2.x, the module didn't have support for qop-auth (authentication using quality of protection), which makes it almost useless. Now in version 1.3.x it has. So it is time to give it a try again. Anyway, the best method is still using TLS for SIP Peering.

Typos in the Configuration File

It is funny that almost nobody gets to complete the CDRTool lab in the first time. It is very common to make a mistake in the `global.inc` file. I copied mine to the files available with this book. I hope it helps.

- It is very common too, to invert the load order of dictionaries in the RADIUS configuration. Please check or you will finish with missing fields in your CDRs.
- Take care of the indentation in the files. It is very easy to get lost in the `if` clauses.

The Last Tip

Be sure to understand the difference between initial and sequential requests. Sequential requests in the same dialog such as BYE and ACK are handled in the `loose_route` section, if you are using "record route". If you can't differentiate what is a transaction and what is a dialog, OpenSER can potentially make your life miserable. Be sure to understand these concepts thoroughly; if you don't, go back to routing basics in Chapter 4.

Forum and Training

By the time you start reading this material, I will have `www.sermayadmin.org` configured. I'm creating a forum to exchange ideas for the book and corresponding training. The participation in the forum will be free of charge. I hope to post answers to the most common questions there. The calendar for training in Portuguese and English will be posted there too. Anyone interested in promoting training, please contact `flavio@asteriskguide.com`. Training will be held in English and Portuguese. Check for available dates and locations.

Summary

In this chapter, I tried to present some of the things that I think will help you. I would like to write a lot more, but one day the book has to finish. OpenSER versions change very often and most of times the scripts are not totally compatible with the previous version. This is a nightmare for anyone trying to write about a topic. I spent a lot of time changing this material from 1.0 to 1.1 and to 1.2 and by the time I wrote this, 1.3.1 is available. I decided to stop running like a "dog chasing its tail" and publish the material with version 1.2.x. I'm making available the script adapted to 1.3.1 in the packtpub support website. Please check this area for newer scripts when new releases are made available. I hope this book helps you. Countless hours were spent testing the labs and migrating from one version to another. I hope all this effort benefits you and enables you to avoid all the time I have spent debugging.

Index

H

I

L

M

N

O

P

R

S

T

V

W