



From Technologies to Solutions

Building Telephony Systems with **OpenSIPS 1.6**

Build scalable and robust telephony systems using SIP

Flavio E. Goncalves

[PACKT]
PUBLISHING

Building Telephony Systems with OpenSIPS 1.6

Build scalable and robust telephony systems using SIP

Flavio E.Goncalves



BIRMINGHAM - MUMBAI

Building Telephony Systems with OpenSIPS 1.6

Copyright © 2010 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: January 2010

Production Reference: 1140110

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

ISBN 978-1-849510-74-5

www.packtpub.com

Cover Image by Vinayak Chittar (vinayak.chittar@gmail.com)

Credits

Author

Flavio E. Goncalves

Production Editorial Manager

Abhijeet Deobhakta

Reviewers

Bogdan-Andrei Iancu

Justin Thomas Zimmer

Editorial Team Leader

Aanchal Kumar

Development Editors

Dilip Venkatesh

Neha Patwari

Project Team Leader

Priya Mukherji

Project Coordinator

Prasad Rai

Technical Editors

Charumathi Sankaran

Smita Solanki

Tarun Singh

Graphics

Nilesh R. Mohite

Copy Editor

Sneha Kulkarni

Production Coordinators

Shantanu Zagade

Aparna Bhagat

Indexer

Monica Ajmera Mehta

Cover Work

Aparna Bhagat

Proofreader

Lesley Harrison

About the Author

Flavio E. Goncalves was born in 1966 in 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 a focus on computer-aided design and computer-aided manufacturing.

He is also the CEO of V.Office Networks in Brazil – a consulting company dedicated to the areas of Networks, Security, and Telecommunications and a training center since its foundation in 1996. Since 1993, he has participated in a series of certification programs and been 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 that the way certification programs were organized in the past was very good for helping learners. Some books today are written by strictly technical people who, sometimes, do not have a clear idea of how people learn. He tried to use his 15 years of experience as an instructor to help people learn about the open source telephony software. His experience with networks, protocol analyzers, and IP telephony combined with his teaching experience give him an edge to write this book. This is the third book written by him; the first one was "*Configuration Guide for Asterisk PBX*", BookSurge Publishing.

As the CEO of V.Office, Flavio E. Goncalves balances his time between family, work, and fun. He is a father of two children and lives in Florianopolis, Brazil, 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 processes of reviewing and editing the book. I would like to thank Bogdan Andrei Iancu for the countless tips on OpenSIPS and the book itself and Adrian Georgescu for his contribution for CDRTool and Media Proxy. I would also like to thank several students, who took courses in the OpenSIPS Bootcamp 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 entered the SIP world in 2001, right after graduating in Computer Science from the "Politehnica" University of Bucharest, Romania. He started in the early days of SIP as a researcher at the Fokus Fraunhofer Institute, Berlin, Germany. For almost four years, Bogdan-Andrei Iancu accumulated a quick understanding and experience of VoIP/SIP, being involved in research and industry project and following tight the evolution of the VoIP world.

In 2005, Bogdan-Andrei Iancu started his own company Voice System. The company entered the open source software market by launching the OpenSER/OpenSIPS project – a free GPL-SIP proxy implementation. As CEO of Voice System, Bogdan-Andrei Iancu pushes the company in two directions: developing and supporting the OpenSIPS public project (Voice System being the major contributor and sponsor of the project), and creating professional solutions and platforms (OpenSIPS based) for the industry. In other words, Bogdan's interest was to create knowledge (by the work with the project) and to provide the knowledge where needed (embedded in commercial products or in raw format as consultancy service).

In the effort of sharing the knowledge of the SIP/OpenSIPS project, together with Flavio E. Goncalves, the author of this book, he started to run OpenSIPS Bootcamp since 2008, an intensive training dedicated to people who want to learn and get hands-on experience on OpenSIPS from the most experienced people.

Bogdan-Andrei Iancu's main concern is to research and develop new technologies or new software for SIP-based VoIP (actually, this is the reason for his strong involvement with the OpenSIPS project), and to pack all these cutting-edge technologies as professional solutions to the industry.

SIP and OpenSIPS became a key factor in the VoIP world along the year – telephony providers, telcos, carrier grades started to adopt and use OpenSIPS as the core component of their VoIP network, because of its stability, performance, and security, but most importantly, because of its reliability as a project.

Justin Thomas Zimmer has worked in the contact-center technology field for twelve years. During that time, he has performed extensive software and computer telephony integrations using both PSTN and IP telephony. His current projects include system designs utilizing open source soft switches over more traditional proprietary hardware-based telephony and the integration of these technologies into market-specific CRM products.

As the Technical Partner of Unicore Technologies out of Phoenix, Arizona, Justin is developing custom business solutions utilizing open source software. Unicore's solutions present businesses with low startup costs in a turbulent economy.

He has worked on *The Hopewell Blogs* — a science fiction adventure novel that will be released online chapter-by-chapter, and available in print once the final chapter has been released.

I'd like to thank the countless community contributors who have provided enough online documentation to make this book as accurate and helpful as possible. And I'd like to thank my wife Nicole for putting up with the extra hours spent reviewing this book, as well as my boys Micah, Caden, and daughter Keira for giving up some of their daddy-time for this project.

Table of Contents

Preface	1
Chapter 1: Introduction to SIP	7
SIP basics	8
SIP operation theory	10
SIP registering process	11
Server operating as a SIP proxy	13
Server operating as a SIP redirect	13
Basic messages	14
SIP dialog flow	15
SIP transactions and dialogs	20
The RTP protocol	21
Codecs	22
DTMF relay	22
Real Time Control Protocol (RTCP)	22
Session Description Protocol (SDP)	22
The SIP protocol and the OSI model	24
VoIP provider, the big picture	24
SIP proxy	25
User administration and provisioning portal	25
PSTN gateway	25
Media server	26
Media Proxy or RTP Proxy for NAT traversal	26
Accounting and CDR generation	26
Monitoring tools	26
Where you can find more information	26
Summary	27

Chapter 2: Introduction to OpenSIPS	29
Where we are	30
What is OpenSIPS?	30
OpenSIPS history	31
Main characteristics	31
Speed	32
Flexibility	32
OpenSIPS is extendable	32
Portability	32
Small footprint	32
Usage scenarios	33
OpenSIPS configuration file	34
Core and modules	35
Sections of the opensips.cfg file	35
Sessions, dialogs, and transactions	36
Message processing in the opensips.cfg	36
SIP proxy—expected behavior	36
Stateful operation	37
Summary	39
Chapter 3: OpenSIPS Installation	41
Hardware requirements	41
Software requirements	42
Lab—installing Linux for OpenSIPS	42
Downloading and installing OpenSIPS v1.6.x	55
OpenSIPS console	56
Lab—running OpenSIPS at the Linux boot	56
OpenSIPS v1.6.x directory structure	57
Configuration files (etc/opensips)	57
Modules (/lib/opensips/modules)	58
Binaries (/sbin)	58
Log files	59
Redirecting OpenSIPS log files	59
Startup options	60
Summary	62
Chapter 4: Script and Routing Basics	63
Where we are	64
Scripting OpenSIPS	64
Global parameters	65
Listen interfaces	65
Logging	65
Number of processes	66

Daemon options	66
SIP identity	67
Miscellaneous	67
Standard script for global parameters	67
Modules and their parameters	68
Standard configuration for modules and parameters	69
Scripting basics	70
Core functions	71
Core values	71
Core keywords	71
Pseudo-variables	72
Script variables	72
Attribute-Value Pair (AVP) overview	74
Flags	76
The module GFLAGS	76
Statements	76
if-else	76
Switch	77
Subroutes	77
Routing basics	77
Routing requests and replies	78
Initial and sequential requests	79
Sample route script	80
Using the standard configuration	88
Common issues	89
Daemon does not start	89
Client unable to register	89
Too many connections	90
Summary	90
Chapter 5: Adding Authentication with MySQL	91
Where we are	92
The AUTH_DB module	92
The REGISTER authentication sequence	94
Register sequence	94
Register sequence code snippet	96
The INVITE authentication sequence	97
INVITE sequence packet capture	98
INVITE code snippet	100
Digest authentication	101
WWW-Authenticate response header	101
The Authorization request header	102

QOP—Quality Of Protection	102
Plaintext or hashed passwords	103
Installing MySQL support	103
Analysis of the opensips.cfg file	106
Register requests	107
Non-Register requests	108
The opensipsctl shell script	109
The resource file—opensipsctlrc	110
The opensipsctlrc file	110
Using OpenSIPS with authentication	113
Enhancing the script	114
Managing multiple domains	116
Using aliases	117
Handling CANCEL request and retransmissions	118
Full script with all the resources above	119
Lab—multi-domain support	125
Lab—using aliases	125
Summary	126
Chapter 6: Graphical User Interfaces for OpenSIPS	127
OpenSIPS Control Panel	128
Installation of opensips-cp	129
Installing Monit	131
Configuring OpenSIPS Control Panel	132
SerMyAdmin	133
Lab—installing SerMyAdmin	134
SerMyAdmin configuration	136
Basic tasks	137
Registering a new user	138
Approving a new user	139
User management	140
Domain management	142
Interface customization	142
Comparing OpenSIPS-CP and SerMyAdmin	143
Summary	144
Chapter 7: Connectivity to the PSTN	145
The big picture	146
Requests sent to the gateway	147
The GROUP module	147
Requests coming from the gateways	148
The module permissions	148
Example	151

Inspection of the opensips.cfg file	156
Using Asterisk as a PSTN gateway	159
Asterisk gateway (sip.conf)	160
Cisco 2601 gateway	161
Dynamic routing	162
Most relevant parameters	162
Sort order	163
Blacklist	163
Force_dns	163
Drouting tables	163
Case study for dynamic routing	165
DIALPLAN transformations	167
DIALPLAN example	168
Inspection of the file opensips.cfg	171
Blacklists and "473/Filtered Destination" messages	173
Summary	173
Chapter 8: Media Services Integration	175
Playing announcements	176
Example: playing demo-thanks	177
Voicemail	178
How to integrate Asterisk Real Time with OpenSIPS	178
Call forwarding	182
Implementing blind call forwarding	183
AVPOPS module loading and parameters	183
Lab—implementing blind call forwarding	184
Implementing call forward on busy or unanswered	186
Inspecting the configuration file	190
Lab—testing the call forward feature	192
Summary	192
Chapter 9: SIP NAT Traversal	193
Why NAT breaks SIP	194
Where NAT breaks SIP	194
NAT types	195
Full cone	195
Restricted cone	196
Port restricted cone	196
Symmetric	197
Why symmetric NAT is hard to traverse	197
NAT firewall table	198
Solving the SIP NAT traversal challenge	198
Implementing a near-end NAT solution	198

Why STUN does not work with symmetric NAT devices	200
Implementing a far-end NAT solution	201
The RFC3581 and the force_rport() function	201
Solving the traversal of the RTP packets	202
RTP Proxy installation and configuration	203
Analysis of the file opensips.cfg	203
Modules loading	203
Modules parameters	204
Determining if the client is behind NAT	204
Handling REGISTER requests behind NAT	206
Handling INVITE messages behind NAT	206
Handling the responses	207
Handling RE-INVITE messages	208
Routing script	209
Invite diagram	216
Packet sequence	217
Lab—using the RTP Proxy for NAT traversal	222
Comparing STUN with TURN (MRS)	223
Application layer gateways (ALGs)	223
Interactive Connectivity Establishment (ICE)	224
Summary	224
Chapter 10: OpenSIPS Accounting and Billing	225
Objectives	225
Where we are	226
VoIP provider architecture	226
Accounting configuration	227
Automatic accounting	227
Multi-leg accounting	228
Lab—accounting using MySQL	228
Analysis of the opensips.cfg file	229
Generating the CDRs	230
Lab—generating Call Detail Records	231
Accounting using RADIUS	232
Lab—accounting using a FreeRADIUS server	232
Package and dependencies	233
FreeRADIUS client and server configuration	233
Configure OpenSIPS server	234

Solving the problem with missing BYEs	236
Account in the gateway instead of the proxy	236
Use SIP session timers	237
Use RTP proxy timeout	237
Use Media Proxy timeout	237
Prepaid and postpaid billing	237
Summary	238
Chapter 11: Monitoring Tools	239
Where we are	240
Built-in tools	240
Trace tools	243
SIPTRACE	243
Configuring the SIPTRACE	243
Stress testing tools	244
SIPSAK	244
SIPp	245
Installing SIPp	245
Stress test—the SIP signaling	246
Stress test—the RTP signaling	249
Wireshark	250
Monitoring tools	254
Summary	254
Index	255

Preface

This book starts with the simplest configuration and evolves chapter by chapter, teaching you how to add new features and modules. It will first teach you the basic concepts of SIP and SIP routing. Then you will start applying the theory by installing OpenSIPS and creating the configuration file. You will learn about features such as authentication, PSTN connectivity, user portals, media server integration, billing, NAT traversal, and monitoring. The book uses a metaphor of a VoIP provider to explain OpenSIPS. The idea is to have a simple but complete running VoIP provider by the end of the book.

What this book covers

Chapter 1, *Introduction to SIP* teaches you the SIP protocol and its functionality along with SIP components, the SIP architecture and describes its main messages and processes.

Chapter 2, *Introduction to OpenSIPS* explains about OpenSIPS and its main characteristics and features. You will see the configuration file, its modules, the configuration blocks, and so forth.

Chapter 3, *OpenSIPS Installation* shows you how to install and prepare Linux for installing OpenSIPS with RADIUS and MySQL modules and getting started with OpenSIPS.

Chapter 4, *Scripting and Routing Basics* discusses the basics needed to construct a working routing script. It explains the global configuration parameters for scripting, the modules, and the routing statements available.

Chapter 5, *Adding Authentication with MySQL* teaches you how to integrate MySQL with OpenSIPS to authenticate users and handle inbound and outbound calls.

Chapter 6, *Graphical User Interfaces for OpenSIPS* explains the need for user and administration portals. It will teach you how to configure access, handle domains, and customize portals.

Chapter 7, *Connectivity to PSTN* teaches you how to connect SIP gateways with PSTN, build dynamic dialplans, and apply permissions.

Chapter 8, *Media Services Integration* teaches you how to connect OpenSIPS to external media servers for implementing user preferences like call forwarding, and integrating databases for simplified administration.

Chapter 9, *SIP NAT Traversal* describes various NAT types and devices. Here we will learn how to implement the Media Proxy solution to solve the NAT traversal problem.

Chapter 10, *OpenSIPS Accounting and Billing* teaches you how to implement the accounting feature with MySQL and RADIUS.

Chapter 11, *Monitoring Tools* discusses how to use built-in monitoring tools and implement testing techniques for OpenSIPS.

Who this book is for

This book targets readers who want to understand how to build a SIP provider from scratch using OpenSIPS. It is suitable for VoIP providers, large enterprises, and universities.

Our objective of writing this book is to take the user from the basics up to the level required to run an OpenSIPS server in a VoIP provider, in an enterprise. Some interesting topics have not been covered. This is because we consider them to be a bit advanced for an introductory book. We hope to cover them soon in another title to be announced.

Telephony and Linux experience will be helpful but is not essential. Readers need not have prior knowledge of OpenSIPS. This book will also help readers who were using OpenSER, but are now confused with OpenSIPS.

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.

Code words in text are shown as follows: "You have to use `www_authorize` when your server is the endpoint of the request." A block of code is set as follows:

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

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in 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/opensips_MySQL" type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/opensips" username="opensips"
password="opensipsrw"/>
</Context>
```

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

```
tar -xzvf sermyadmin-install-2.x.tar.gz
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Now, choose **Finish partitioning and write changes to disk**".

 Warnings or 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 send an e-mail to feedback@packtpub.com, and mention the book title via 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 e-mail 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 on, 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/0745_Code.zip to directly download 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 content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration, and help us to improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to any list of existing errata. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or web site name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any 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 **Request for Comment (RFC)**. RFC3261 is one of the most recent of the documents, 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 audio and video sessions, e-learning, chatting, or screen-sharing sessions. It is based on a text protocol similar to **Hypertext Transfer Protocol (HTTP)** and 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 the 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 the following five features for establishing and closing multimedia sessions:

1. **User location:** Determines the endpoint address used for communication.
2. **User parameters negotiation:** Determines the media and parameters to be used.
3. **User availability:** Determines if the user is available to establish a session.

4. **Call establishment:** Establishes the parameters for both the caller and callee, and informs both parties about the call progress (ringing, ringback, congestion).
5. **Call management:** Session transfer and closing.

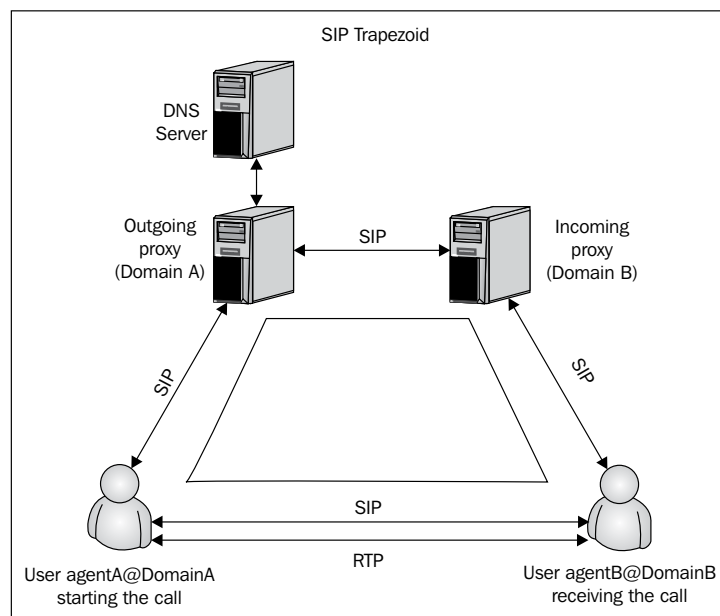
The SIP protocol was designed as part of a multimedia architecture containing other protocols such as RVSP, RTP, RTSP, SDP, and SAP. However, it does not depend on them for its operation.

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**, which enables you to 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 handle signaling and media. This is depicted in the next image:



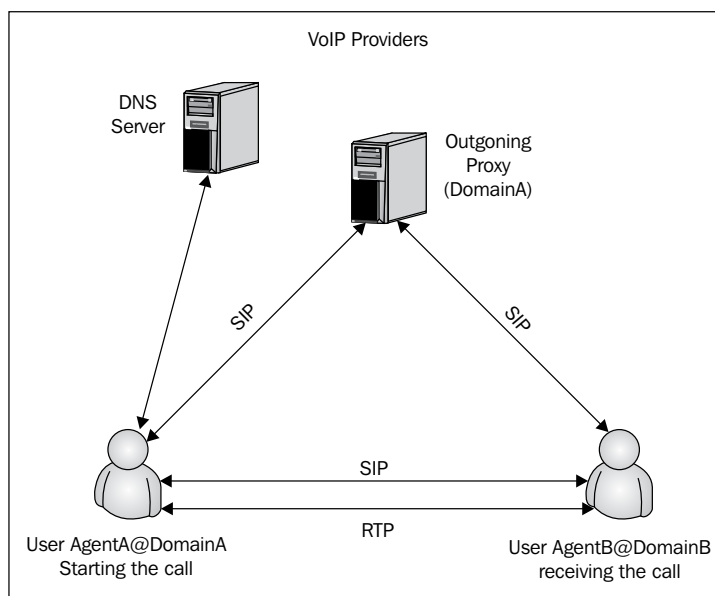
In the SIP model, a user agent – usually a SIP phone – will start communicating with its SIP proxy – seen here as the outgoing proxy (or its home proxy) – to send the call using a message known as INVITE.

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 the **DomainB**.

The incoming proxy will verify, on its location table for the IP address of the **agentB**, if this address was inserted in the location table by a previous registration process. If the incoming proxy can locate the address, it will forward the call to the **agentB**.

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

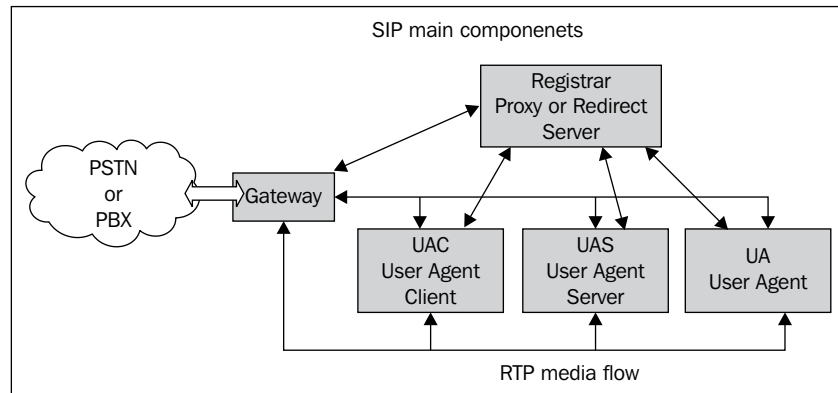
An example of a SIP message is shown in the next image:



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 something that is closer to a SIP triangle.

SIP operation theory

You can see the main components of the SIP architecture in the following image:



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 following sequence:

1. **UAC (User Agent Client):** The client or terminal that starts the SIP signaling
2. **UAS (User Agent Server):** The server that responds to the SIP signaling coming from an UAC
3. **UA (User Agent):** The SIP terminal (IP phones, ATAs, softphones, and so on)
4. **Proxy server:** Receives requests from a UA and transfers to another SIP proxy if this specific terminal is not under its domain
5. **Redirect server:** Receives requests and responds to the caller with a message containing data about the destination ("302", Moved Temporarily)
6. **Location server:** Provides the callee's contact addresses to proxy and redirect servers

OpenSIPS can be configured to provide proxy, redirect, and location services on a single platform.

SIP registering process

The following image depicts the SIP registering process:



The SIP protocol employs a component called **REGISTRAR**. It is a server which accepts **REGISTER** requests and saves the information received within 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 the target destination. This is stored in a location database maintained by the Location server per domain. The register server may accept other types of information, not only the client's IP addresses. It can receive other information such as **Call Processing Language (CPL)** scripts on the server.

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

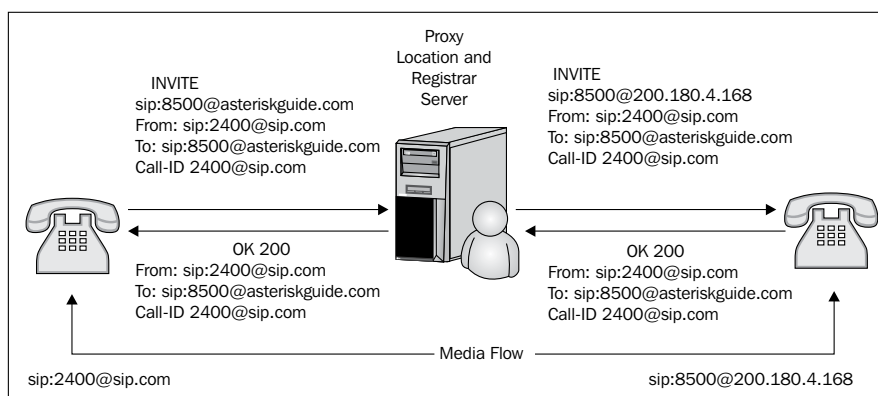
RFC3665 defines best practices to implement a minimum set of functionality for a SIP IP communications network.

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

Diagram	Description
New registration <pre> sequenceDiagram participant UA as User Agent participant SP as Sip Proxy Note over UA, SP: New registration UA->>SP: REGISTER SP-->>UA: 401 Unauthorized UA->>SP: REGISTER SP-->>UA: 200 OK </pre>	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 Chapter 5, <i>Adding Authentication with MySQL</i>, which is dedicated to authentication.
Update of Contact List <pre> sequenceDiagram participant UA as User Agent participant SP as Sip Proxy Note over UA, SP: Update of Contact List UA->>SP: REGISTER SP-->>UA: 200 OK </pre>	Update of Contact list: As 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.
Request for current Contact list <pre> sequenceDiagram participant UA as User Agent participant SP as Sip Proxy Note over UA, SP: Request for current Contact list UA->>SP: REGISTER SP-->>UA: 200 OK </pre>	Request for current Contact list: In this case, the User Agent will send the CONTACT header field empty, indicating that 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.
Cancellation of the registration <pre> sequenceDiagram participant UA as User Agent participant SP as Sip Proxy Note over UA, SP: Cancellation of the registration UA->>SP: REGISTER SP-->>UA: 200 OK </pre>	Cancellation of the 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 the existing contacts.
Unsuccessful Registration <pre> sequenceDiagram participant UA as User Agent participant SP as Sip Proxy Note over UA, SP: Unsuccessful Registration UA->>SP: REGISTER SP-->>UA: 401 Unauthorized UA->>SP: REGISTER SP-->>UA: 401 Unauthorized </pre>	Unsuccessful Registration: The UAC sends a REGISTER request and receives a 401 Unauthorized message in exactly the same way as successful registration. In the sequence, it produces a hash and tries to authenticate. The server, detecting an invalid password, sends a 401 Unauthorized message again. The process will be repeated for the number of retries configured in the UAC.

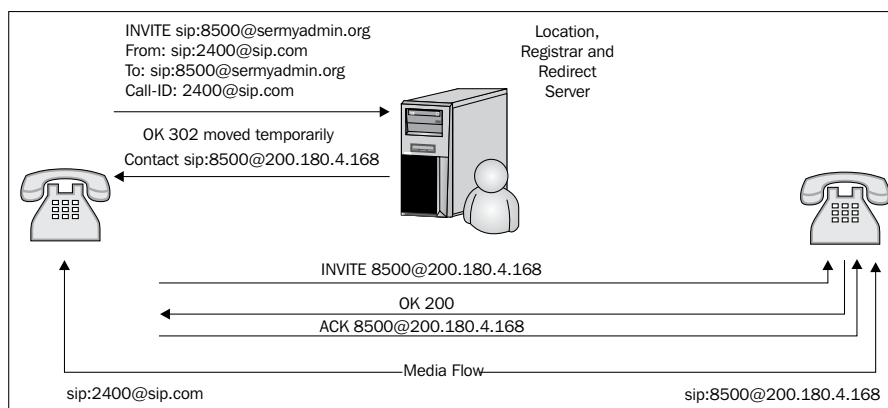
Server operating as a SIP proxy

In SIP proxy mode, all SIP signaling goes through the SIP proxy. This behavior will help in processes such as billing and 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's establishment. Remember, RTP packets will always go directly from one endpoint to another, even if the server is working as a SIP proxy. This is shown in the following screenshot:



Server operating as a SIP redirect

The SIP proxy can operate in 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" message and is removed from 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:

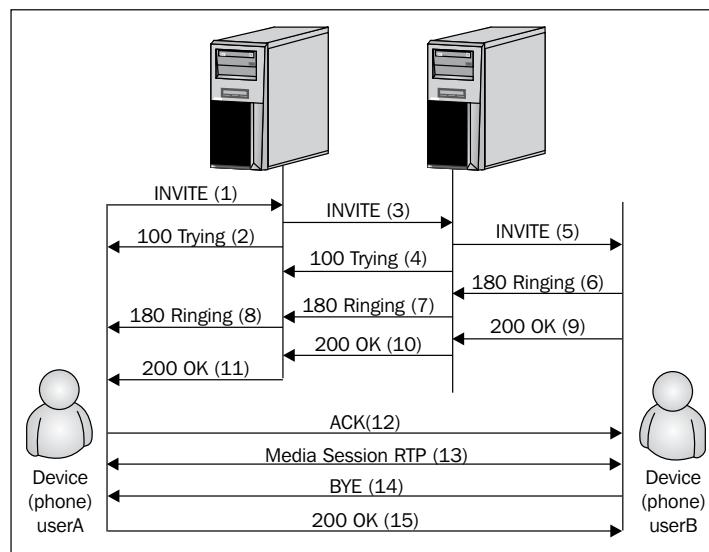
Message	Description	RFC
ACK	Acknowledge an INVITE	RFC3261
BYE	Terminate an existing session	RFC3261
CANCEL	Cancel a pending registration	RFC3261
INFO	Mid-call signaling information	RFC2976
INVITE	Session establishment	RFC3261
MESSAGE	Instant message transport	RFC3428
NOTIFY	Send information after subscribe	RFC3265
PRACK	Acknowledge a provisional response	RFC3262
PUBLISH	Upload status information to the server	RFC3903
REFER	Ask another UA to act upon URI	RFC3515
REGISTER	Register the user and update the location table	RFC3261
SUBSCRIBE	Establish a session to receive future updates	RFC3265
UPDATE	Update a session's 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 responses are shown in the following image:

Description	Code	Examples	
Informational or provisional response	1XX	100 Trying 180 Ringing 181 Call Is Being Forwarded	182 Queued 183 Early Media
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 RequestURI Too Large 415 Unsupported Media Type 420 Bad Extension 480 Temporarily not available 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 Timeout 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 in the next image. You can see several other flows associated with session establishment in the 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.

userA calls **userB** using its SIP identity, called the SIP URI. The URI is similar to an e-mail 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:` (secure SIP) 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 as shown here:

INVITE from A->B

```
INVITE sip:userB@sipB.com SIP/2.0
Via: SIP/2.0/UDP moon.sipA.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: userB <sip:userB@sipB.com>
From: userA <sip:userA@sipA.com>;tag=1234567890
Call-ID: a84b4c76e66710@moon.sipA.com
CSeq: 314159 INVITE
Contact: <sip:userB@sun.sipB.com>
Content-Type: application/sdp
Content-Length: 142
(SDP not shown)
```

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 as follows:

- **VIA:** This header field contains the address which will be used to send the responses back for 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 their route back using the VIA header than to refer back to the location server or DNS.
- **TO:** This contains the name (display name) and the SIP URI (`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 (`sip:userA@sip.com`) that indicates 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 a 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 hostname or IP address from the IP phone. A combination of the **TO**, **FROM**, and **CALL-ID** tags 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 with 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 by a username and a FQDN (fully qualified domain name). Sometimes, the domains are not registered and thus IP addresses are permitted too. While the **VIA** header field tells the other elements where to send a response, the **CONTACT** field 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 which is decremented by one each hop.
- **CONTENT-TYPE:** This contains a body message description.
- **CONTENT-LENGTH:** This contains a byte count of the body message.

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

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**. The sequence is as follows:

1. In this example, the proxy receives the **INVITE** request and sends a message "100 trying" back to **userA**, signaling that the proxy receives 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 **VIA** header field such as the **INVITE** request. This allows **userA**'s phone to correlate to the **INVITE** request sent.

2. ProxyA locates proxyB consulting a DNS server (SRV records) to find out 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 their own address. The INVITE request already had the address of userA in the first VIA header field.
3. ProxyB receives the INVITE request and responds with a "100 Trying" message back to the proxyA indicating that it is processing the request.
4. ProxyB consults their own location database for userB's address, and then adds another VIA header field with their own address to the INVITE request and forwards it to userB's IP address.
5. userB's phone receives the INVITE request and starts ringing. The phone indicates back this condition by sending a message of "180 Ringing".
6. This message is routed back through both proxies in the reverse direction. Each proxy uses the VIA header field to determine where to send the response and removes their 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 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" message, it starts to ring back, signaling the user that the call is ringing on the other side. Some phones show this in the display.

In this example, **userB** decides to attend the call. When he or she picks up the handset, the phone sends a response of "200 Ok" to indicate that the call was taken. The "200 Ok" message contains a session description in its body, specifying codecs, ports, and everything pertaining to the session. It uses the SDP protocol for this duty. As a result, an exchange occurs 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 message "200 Ok" won't be sent and a message signaling the condition (that is, "487 busy here") will be sent instead.

Response 200 Ok

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP sipB.com
;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP moon.sipA.com
;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP phoneA.sipA.com
;branch=z9hG4bK776asdhds;received=192.0.2.1
To: userB <sip:userB@sipB.com>;tag=a6c85cf
From: userA <sip:userA@sipA.com>;tag=1928301774
Call-ID: a84b4c76e66710@phoneA.sipA.com
CSeq: 314159 INVITE
Contact: <sip:userB@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
```

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

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

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

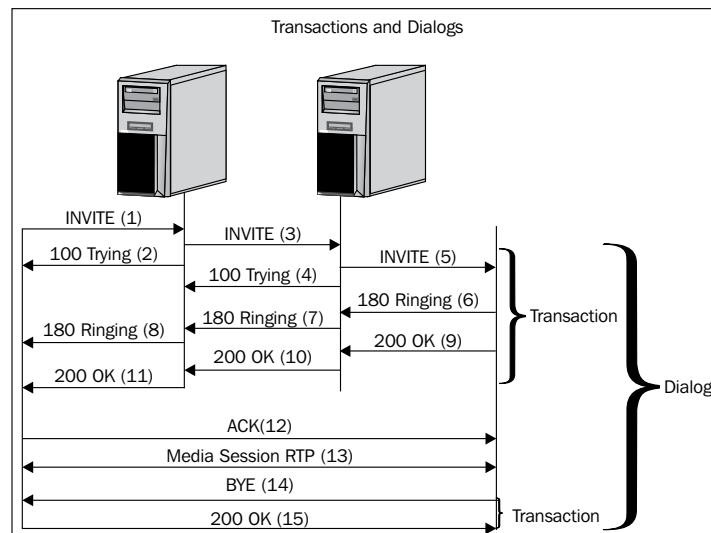
1. In this case, the message "200 Ok" is sent back through both proxies and is received by **userA**, and then the phone stops ringing back indicating that the call was accepted.
2. Finally, **userA** sends an ACK message to **userB**'s phone confirming the reception of the "200 Ok" message. In this example, the ACK is sent directly from phoneA to phoneB, avoiding both proxies. ACK is the only SIP method which has no reply. The endpoints learned each other's addresses from the CONTACT header fields during the INVITE process. This ends the INVITE/200 OK/ACK cycle, also known as the 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's characteristics by 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's 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 learns the location of userB. When the phone initializes, or at regular time intervals, the 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 servers are in the same computer and use the same software. OpenSIPS is capable of playing all three roles. A URI can only be registered by a single device at a time.

SIP transactions and dialogs



It is important to understand 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 initial request to the final response (including all interim responses). The responses can be provisional, starting with 1 followed by two digits (for example, "180 Ringing") or final starting with 2 followed by two digits (for example, "200 Ok"). The scope of a transaction is defined by the stack of the VIA headers of the SIP messages. So, after the initial invite, the user agents don't need to rely on DNS or location tables to route the messages.

According to RFC3261:

A dialog represents a peer-to-peer SIP relationship between two user agents, which persists for some time. A dialog is identified at each UA with a dialog ID, which consists of a Call-ID value, a local tag, and a remote tag.

A dialog is a succession of transactions which control the creation, existence, and termination of the dialog. All dialogs have a transaction to create them and may or may not have a transaction to change them (mid-transaction). Also, the end dialog transaction may be missing (some dialogs end based on timeouts rather than by explicit termination).

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, "Successful Session Establishment" and "Session Establishment Through Two Proxies", are already covered in this chapter. Some of the others will be seen in the chapter dedicated to call forwarding such as "Unsuccessful 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 on RFC 3550. 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 **Real Time Control Protocol (RTCP)** 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. G.711 is the most popular codec and it does not use compression. 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 (WANs)**, 64 Kbps can be too expensive to buy for a single channel. Codecs such as G.729 and GSM can compress the voice packets to as low as 8 Kbps, 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. The 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 a 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 the 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 "200 Ok" contains the answer message. These messages are shown in the next screenshot. 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.

Here is the INVITE (SDP offer) message:

```

Session Initiation Protocol
+ Request-Line: INVITE sip:8520@8.8.30.36:42989 SIP/2.0
+ Message Header
+ Message body
  Session Description Protocol
    Session Description Protocol version (v): 0
    Owner/Creator, Session Id (o): root 10968 10968 IN IP4 8.8.1.4
      Owner Username: root
      Session ID: 10968
      Session Version: 10968
      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 17412 RTP/AVP 0 3 18 101
    + Media Attribute (a): rtpmap:0 PCMU/8000
    + Media Attribute (a): rtpmap:3 GSM/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 - - - -
  
```

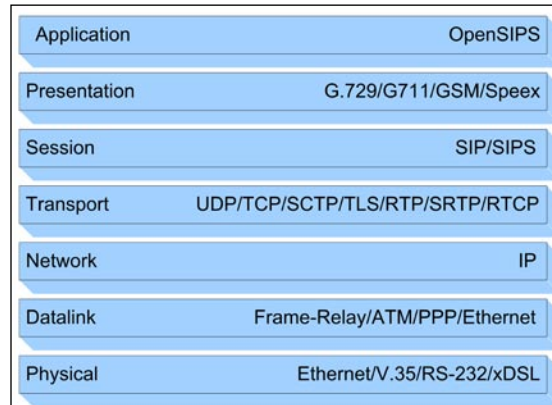
The following screenshot shows the "200 Ok" (SDP answer) reply:

```

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 - - - -
  
```

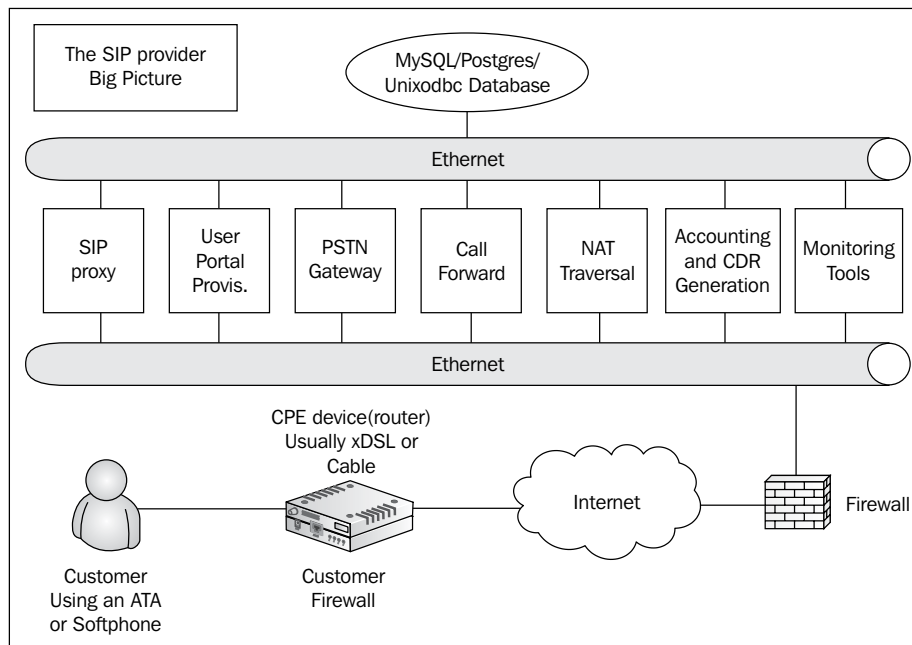

The SIP protocol and the OSI model

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



VoIP provider, the big picture

Before we start digging into the SIP proxy, it is important to understand all the components for a VoIP provider solution. They are shown in the following image:



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 the chapters to help you realize where you are.

SIP proxy

The SIP proxy is the central component of our solution. It is responsible for registering the users and keeping the location database (maps IP to SIP addresses). The entire SIP routing and signaling is handled by the SIP proxy, and it is also responsible 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 also be capable of buying credits, changing passwords, and verifying his or her account. On the other hand, administrators should be able to remove users, change user credits, and 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 soft-phones.

PSTN gateway

To communicate with the public switched telephone network, a PSTN gateway is required. Usually, this gateway will interface the PSTN using E1 or T1 trunks. The most common products in this arena are gateways from CiscoTM, AudioCodesTM, and QuintumTM. Asterisk is gaining market in this area because of their price per port cost, which is sometimes 75% less than the competitors. To evaluate a good gateway, check the support of SIP extensions such as RFC3515 (Refer), RFC3891 (Replaces), and RFC 3892(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. **SIP Express Media Server (SEMS)** 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 their customers. The Media Proxy is an RTP bridge that helps the users behind symmetric firewalls to access the SIP provider. Without them, it won't be possible to service a large share of the user base. You can implement a universal NAT traversal technique using these components. The Media Proxy can also help you in the accounting correction for unfinished SIP dialogs which, for some reason, didn't receive the BYE message.

Accounting and CDR generation

An AAA (Authentication, Authorization and Accounting) server can be used along with OpenSIPS. FreeRADIUS is a common choice. In several implementations, you can skip RADIUS and use SQL accounting. Some VoIP providers will leverage an existing AAA server, while some others will prefer the low-overhead MySQL accounting. SerMyAdmin can now be used for accounting purposes. The CDR generation is beyond accounting, where the duration of the calls is calculated.

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. The OpenSIPS has a module called SIPTRACE. We will use that too.

Where you can find more information

The best reference for the SIP protocol is RFC3261. Reading the RFCs is a little bit boring and sleep inducing (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, 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

implementers: <https://lists.cs.columbia.edu/mailman/listinfo/sip-implementors>

Summary

In this chapter, you have learnt what the SIP protocol and its functionality is. You had the opportunity to learn about the SIP components such as the SIP proxy, SIP REGISTRAR, user agent client, user agent server, and PSTN gateway. You saw the SIP architecture, and its main messages and processes. You can explore some sources to find further information listed too.

2

Introduction to OpenSIPS

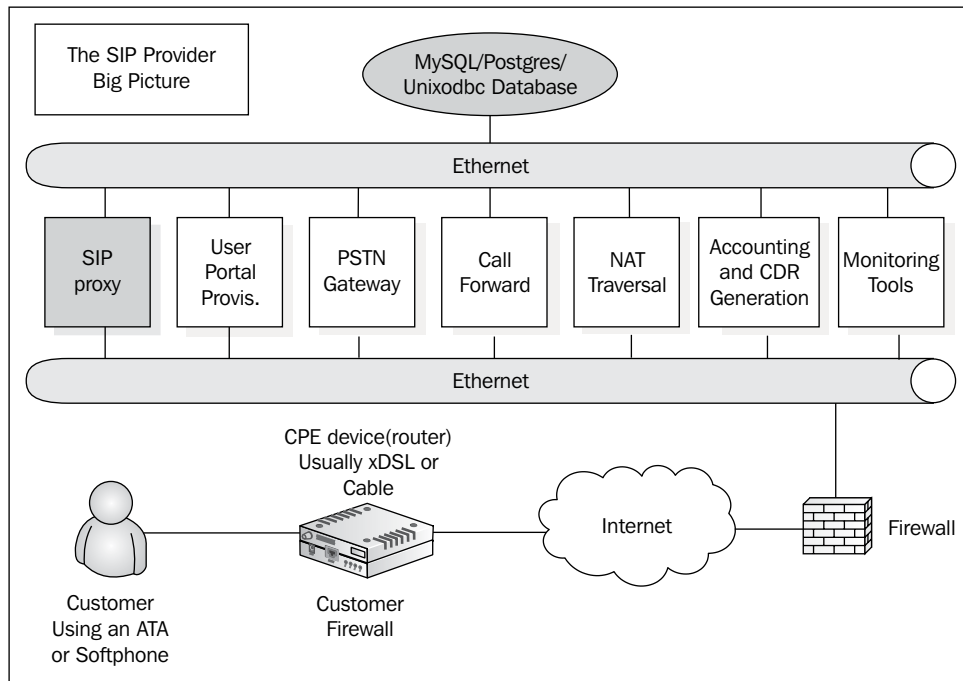
In the last chapter, we discussed the big picture of running a VoIP provider. Usually, a VoIP provider is composed of several components. These components can reside on the same machine or be spread over several machines, depending upon your dimensioning. One of these components is the SIP proxy server; in our case, the server running the OpenSIPS software. What best describes the OpenSIPS server is a SIP router. It is capable of manipulating the SIP headers and routing packets at extremely high speeds. Third-party modules give OpenSIPS extreme flexibility to play roles that were 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 OpenSIPS server.

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

- Explain what OpenSIPS is
- Describe their usage scenarios
- Distinguish between the different sections of the `opensips.cfg` file
- Describe the processing of the SIP messages

Where we are

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



What is OpenSIPS?

OpenSIPS is an open source SIP proxy server compliant with the IETF RFC3261 SIP protocol. It is targeted at large-volume applications.

With its small footprint, OpenSIPS is extremely fast in forwarding requests and can handle thousands of users with a single server. It is being used by both large VoIP providers and embedded IP PBXs with very low processing power.

OpenSIPS history

OpenSIPS is based on SER, originally developed by the FhG Fokus research institute in Berlin, Germany and released under the GPL license. OpenSERs was the first fork of the original SER project. In 2004, FhG Fokus started a spin-off of the SER project creating the `iptel.org`. In 2005, the commercial variant of `iptel.org` was sold to Tekelec. The core development team was split into two; some of them went to `iptel.org` and the others left FhG to start a company called Voice System, the main maintainer of the OpenSER project that started in 2005.

The concept of this book started in late 2005 based on the SER project. At that time, I was interested in a NAT traversal solution that was available only using SER. The scalability of *Asterisk* was not good enough to host a SIP provider, so I started playing with SER. The documentation was really hard to understand, so I started writing my own to train the administrators of the SIP providers. The book evolved to OpenSER and more recently to OpenSIPS. The OpenSER project was renamed and forked in 2008. Now, there are two variants – Kamailio and OpenSIPS.

I don't want to get into the politics of SER, Kamailio, and OpenSIPS. The concepts presented here are valid for both. I wrote the book using OpenSIPS because, in my opinion, they are delivering software in a consistent manner. Another decision is to write about the development version. I hope it will be released by the time the book is available.

OpenSIPS has a flexible plugin model for third-party applications. Applications can be easily created and plugged into the server. Thus, accounting plugins such as RADIUS, ENUM, Presence, and SMS were written. Newer modules are being added every month. You can check available modules for OpenSIPS 1.5.x at:

<http://www.opensips.org/index.php?n=Resources.DocsModules>.

OpenSIPS is not used just by service providers. Today, it can be used to construct SIP appliances. SIP firewalls, **session border controllers (SBCs)**, and load balancers using code borrowed from the OpenSIPS project. OpenSIPS was chosen by Linksys for the one PBX platform, probably because of the small footprint and high performance available.

Main characteristics

Some of the main characteristics of the OpenSIPS project makes it specially suited for certain environments. When you need the speed, flexibility, scalability, portability, and the capability to run in environments with low computing capacity, OpenSIPS may be the perfect match.

Speed

OpenSIPS can handle tens of thousands of calls per second even on low-cost hardware. This speed allows networks to be configured with off-the-shelf hardware with little maintenance. The development using ANSI C with some assembler routines is responsible for this kind of performance.

Flexibility

OpenSIPS is very open and their administrators can define their behavior using a script language. The script language is flexible and powerful to attend even the most complicated scenarios.

OpenSIPS is extendable

OpenSIPS can be extended by linking new codes developed in C. The new code can be developed independent of the OpenSIPS core and linked at the execution time. The concept is similar to the modules in Apache Web Servers. Recently, new layers of programming were added. It is possible to use **Call Processing Language (CPL)** to simplify the routing scripts. **WeSIP** is an application program interface that allows you to use Java and servlets to extend the OpenSIPS server creating a SIP Application Server. These extensions (providing new functionalities) are called modules. OpenSIPS 1.6 provides almost 100 different modules, which can be selectively loaded and used in the configuration/routing file.

Portability

OpenSIPS was written in ANSI C. Thus, it is highly portable and available to UNIX-like systems such as Linux, Solaris, and BSD.

Small footprint

The OpenSIPS core is very thin. The oldest versions were 300 KB in size. With some modules, this can increase to a few megabytes. Because of this characteristic, it is being used in several embedded platforms.

Usage scenarios

OpenSIPS is primarily used as a SIP proxy and REGISTRAR. However, it can be used in some other applications such as a proxy dispatcher/balancer, Jabber Gateway, Presence server, SIP multirouter, and NAT traversal together with Media Proxy and RTP Proxy. It supports IP versions 4 and 6, and can serve multiple domains. OpenSIPS can be executed on the Linux, Solaris, and FreeBSD platforms.

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

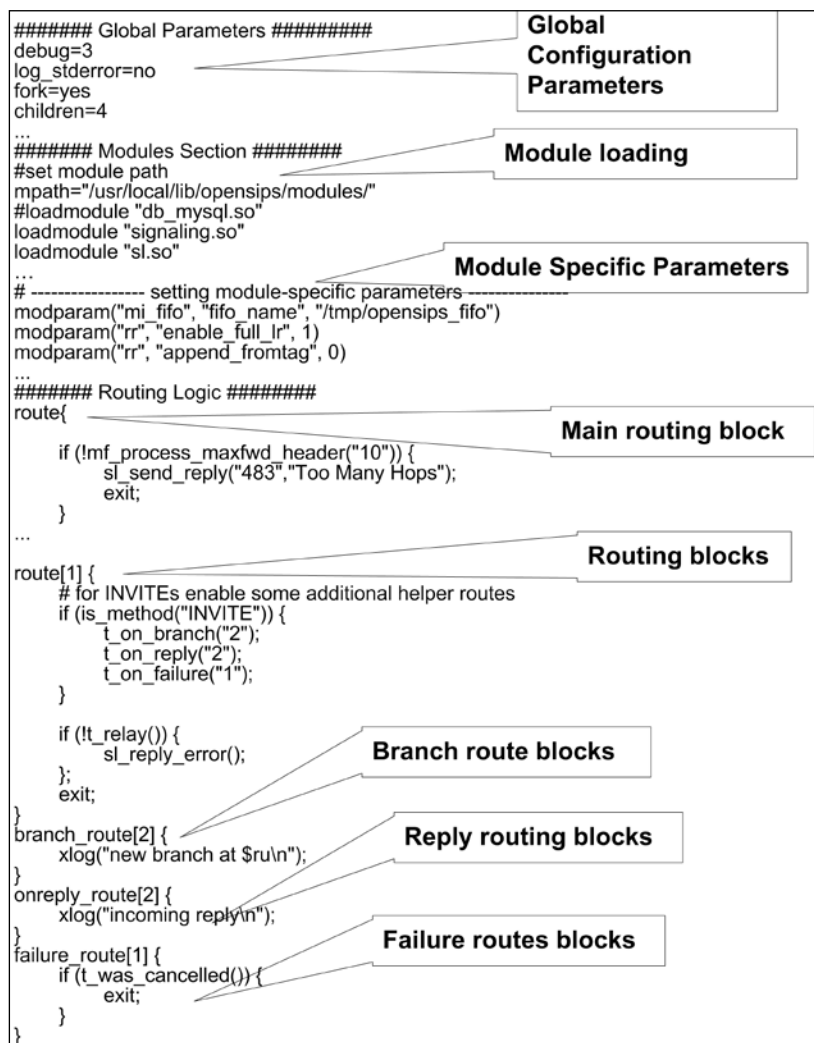
Modules	Functionality
DISPATCHER, PATH	Load balancing
MEDIAPROXY, RTPPROXY, NATHELPER	Nat traversal
PRESENCE	Presence server
IMC, XMPP	Instant messaging

Let's see the most common usage scenarios for OpenSIPS. In all these scenarios, OpenSIPS works like glue that binds all SIP components together, such as:

- VoIP providers
- Instant messaging providers
- SIP load balancing
- Embedded IP PBX
- NAT traversal
- SIP.edu

OpenSIPS configuration file

The file depicted in the following image is the main OpenSIPS configuration file, `opensips.cfg`:



Core and modules

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

Sections of the `opensips.cfg` file

The `opensips.cfg` file has several sections, which are as follows:

- **Global definitions:** This portion of the file contains several working parameters for OpenSIPS, including the listening `IP:PORT` pair for the SIP service and debug level. These global parameters affect the OpenSIPS core and all the modules globally.
- **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 need to be set appropriately. These parameters are configured using `modparam(modulename, parametername, parametervalue)`.
- **Main routing block:** The main routing block is where the SIP request processing starts. It controls the processing of each SIP request received.
- **Secondary routing blocks:** The administrator can define new routing blocks using the `route()` command. These routing blocks work like subroutines in the OpenSIPS script.
- **Reply routing blocks:** Reply routing blocks are used to process reply messages (provisional, successful final replies, or negative final replies), usually `200 OK`.
- **Failure routing blocks:** Failure routing blocks are used to process failure conditions such as busy or timeout.
- **Branch routing blocks:** Contains the logic to be executed for each branch of the SIP request, just before forwarding it out.
- **Local routing blocks:** Local routing blocks are executed when OpenSIPS internally generates a request (acting as UAS only) using the **Transaction Module (TM)**.
- **Error routing block:** This route is executed when a parsing error for a SIP request is detected.



The `opensips.cfg` file will be covered in detail in the chapters ahead.

Sessions, dialogs, and transactions

It is important to understand the following SIP concepts which are used in the OpenSIPS processing:

- **SIP transaction:** A SIP request 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

Message processing in the `opensips.cfg`

The `opensips.cfg` file is a script which is executed for each SIP message received. For example, if userA wants to talk to userB, it sends an INVITE message. This message is processed in the main routing block. The processing may contain a `t_relay()` (forward) or a `t_reply/sl_send_reply` (send a negative reply), or the processing may even discard the message at the end of the block using the `exit()` command. The `t_relay()` and `sl_send_reply()` commands do not end the script; you need to explicitly call the `exit()` command after them.

SIP proxy—expected behavior

It is important to understand the basic processing of a **SIP proxy** according to the 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 a reverse order.

A SIP proxy can operate in a **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 the stateless mode discards any internal information about the message, after the message has been forwarded. This characteristic limits the failure treatment and billing.

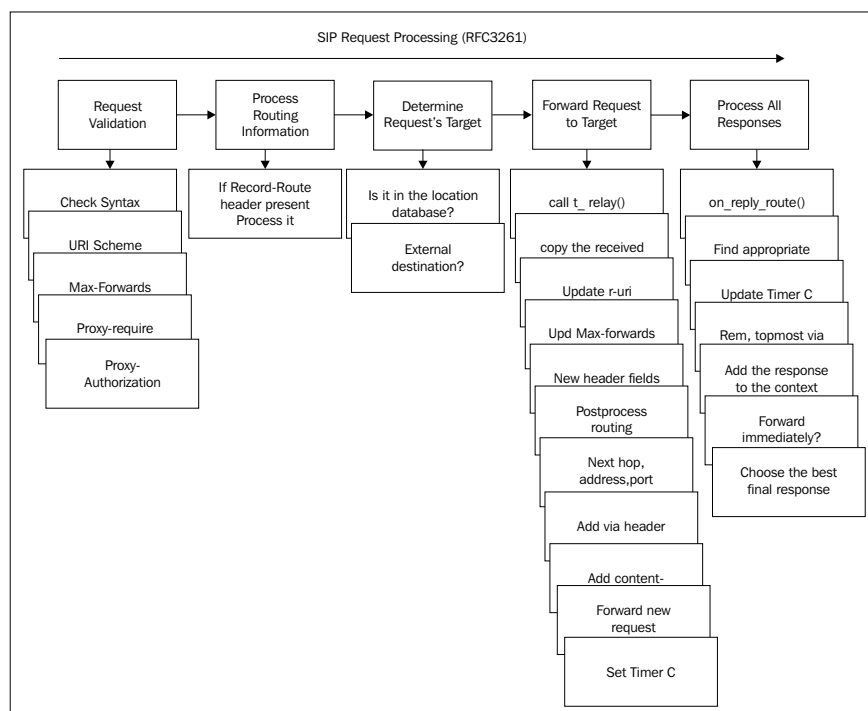
When OpenSIPS knows that the 200 OK message corresponds to a specific INVITE, we say that it is working in stateful mode. This simply means that you can now manage the response in an `on_reply_route()` block. With stateless processing, each message is handled without a context. Stateless processing is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

When you need more sophisticated resources such as billing, call forwarding, 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 uses the `t_relay()` command. Also, more advanced SIP functionalities such as serial and parallel forking work only in a stateful mode.

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

Stateful operation

The following image is a simplified depiction of the stateful operation:



You will find a complete and more detailed description in the RFC3261 text. There is a close resemblance between the `opensips.cfg` sections and the previous figure. However, some processes are manual, such as checking 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 a stateful mode, a proxy is simply a SIP transaction processor. The following processing steps are required:

1. Validate the request.
2. Preprocess the routing information.
3. Determine the request's target.
4. Forward the request to the target.
5. 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 preprocessing

- If a record-route header exists, 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 `t_relay()` function, and the OpenSIPS will do all the job for you statefully

Step 5: Response processing

- Usually, it is done automatically by OpenSIPS. Sometimes, you can use the `on_reply_route[]` section to handle some responses. For example, in a "call forward on busy" scenario, we could use the response 487 (Busy) to direct the call to a voicemail server.

Summary

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

3

OpenSIPS Installation

The installation is just the beginning of the work. It is very important to install OpenSIPS correctly from the source code. It can be installed much faster from the Debian packages or using the `apt-get` utility. However, installation from the source code is much more flexible as it allows you to select the modules to be compiled. So, we won't use any shortcuts in the installation. I strongly advise you to install OpenSIPS using Debian.

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

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

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

Hardware requirements

There are no minimum hardware requirements for OpenSIPS. It will run on 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(R) Core(TM)2 CPU 6300 @ 1.86GHz, 1 GB of memory, and 100Mbps Ethernet card. Unfortunately, there are currently no formulas for OpenSIPS dimensioning. The correct hardware, CPU, and memory shall be obtained empirically.

Software requirements

The OpenSIPS software runs on a variety of platforms such as Linux, BSD, and Solaris. Some generic packages are available to certain versions of Linux and Solaris. These packages can be downloaded from www.opensips.org/Resources/Downloads. The following packages are required to compile OpenSIPS.

- gcc (or any other C compiler as suncc or icc)
- bison or yacc (Berkley yacc)
- flex
- GNU make
- GNU tar
- GNU install
- libxml2-dev (if you want to use the presence modules)
- libxml-rpc (for mi_xmlrpc)

Some modules such as MySQL, POSTGRES, RADIUS, and others will require additional packages for compilation. We will discuss these in their respective chapters.

Lab—installing Linux for OpenSIPS

All of these labs were prepared using a VMware virtual machine with Debian installed. You may download it from <http://cdimage.debian.org/debian-cd/>.

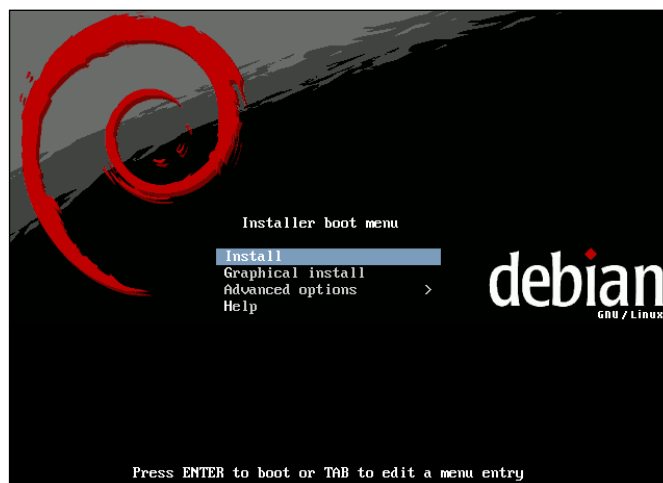


Warning:

The instructions for this lab formats the computer. Back up all the data on your PC before proceeding or follow these instructions in a virtual environment such as VMware or XEN.

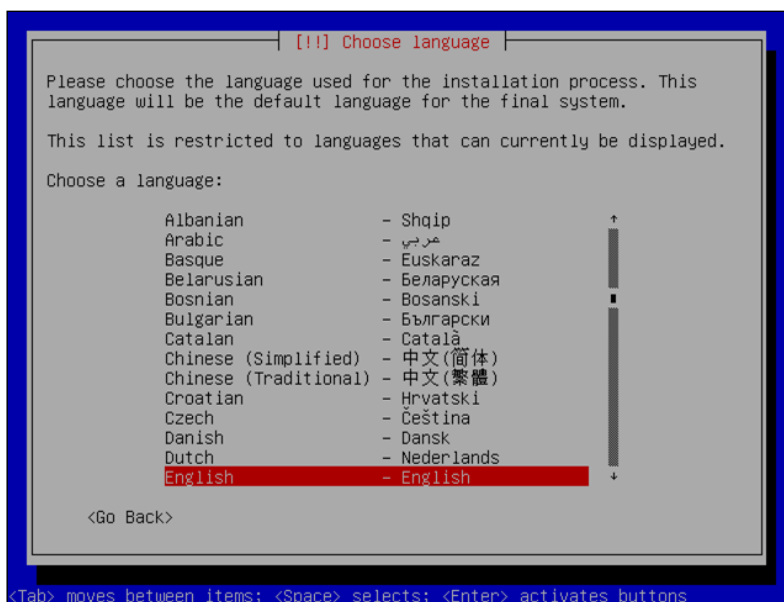
The steps for installing Linux are as follows:

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

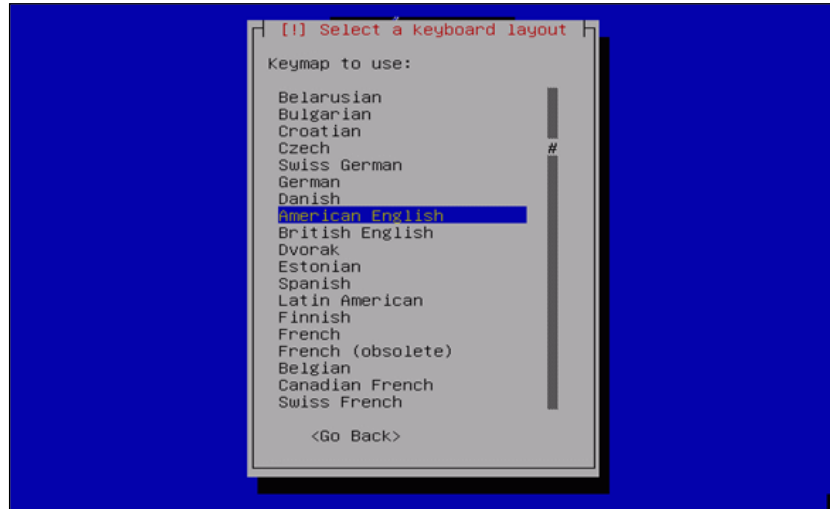


Here, you can also select boot and installation options. Sometimes you may need to choose some hardware-specific parameters for your installation. Press *F1* for help, if needed.

Step 2: Choose a language of your preference for use in the installation process.

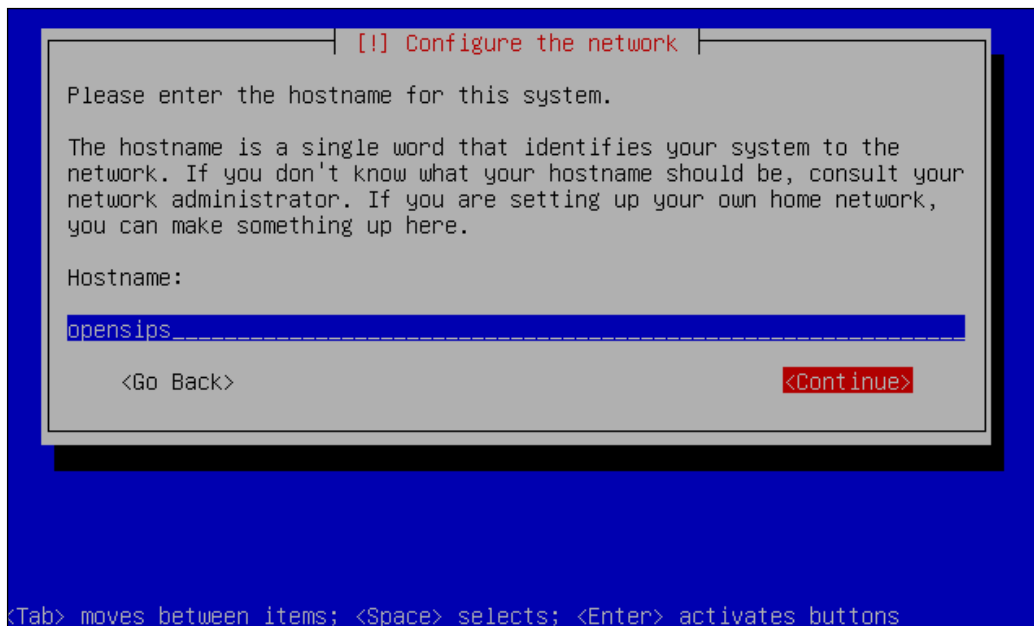


Step 3: Choose the keyboard layout.



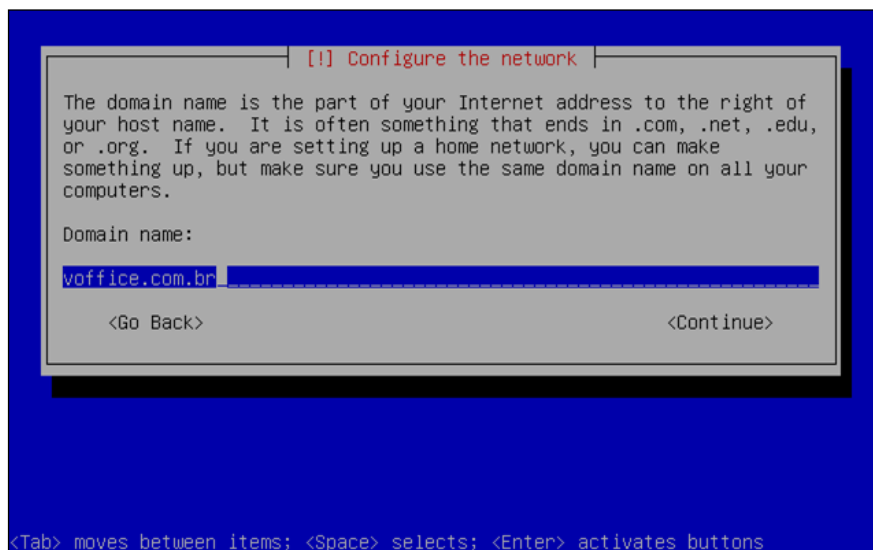
It is very common to have to choose a keyboard layout, especially in European and Asian countries.

Step 4: Choose the Hostname.



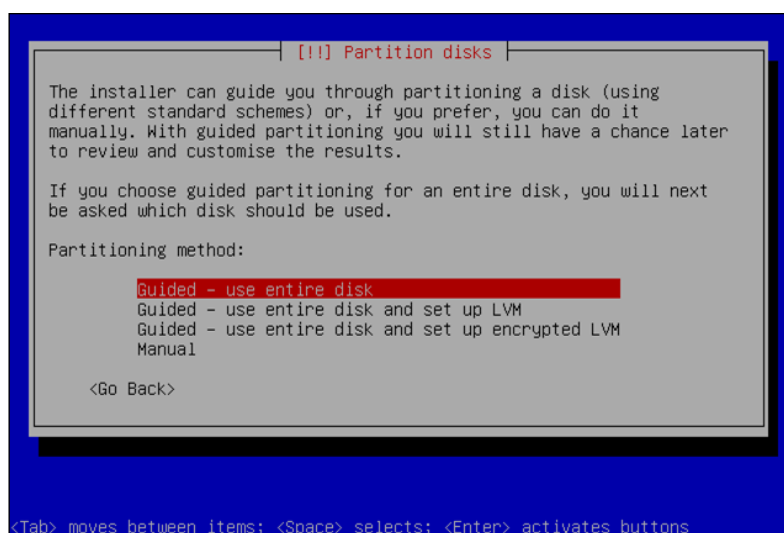
It is important to choose the name of the server because later you will use this name to access the server.

Step 5: Choose the Domain name.



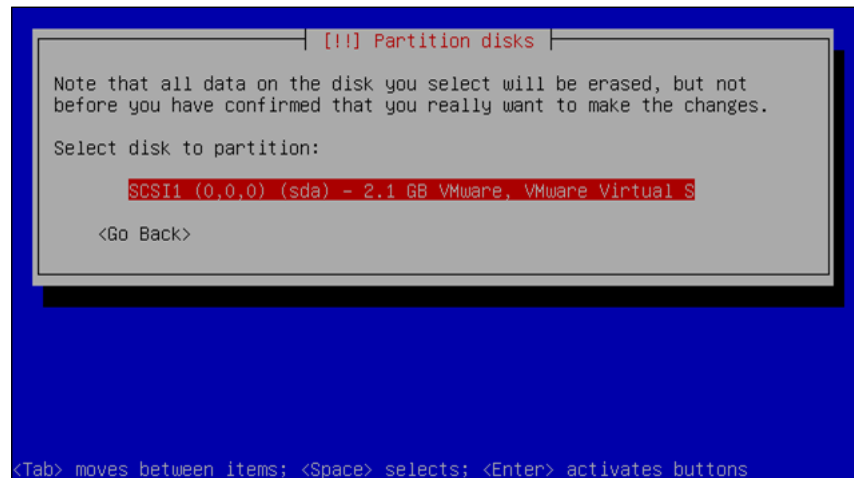
The domain name is obvious but important because OpenSIPS use domains to distinguish users, so be sure to enter the domain name correctly.

Step 6: Choose a Partitioning method.



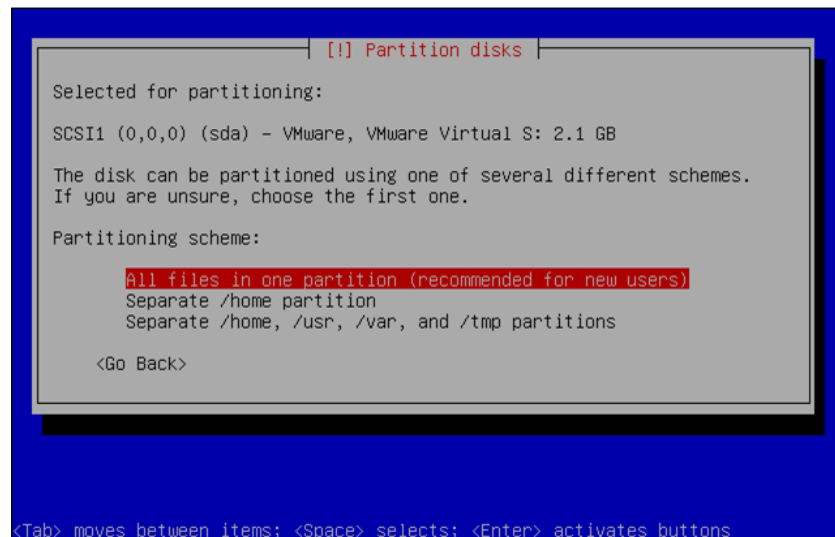
We could write a whole chapter about partitioning. Linux geeks, certainly, will use the manual option. For the purpose of learning, you can simply use the entire disk. Consult a Linux specialist for the best partitioning scheme for your server.

Step 7: Select disk to partition.



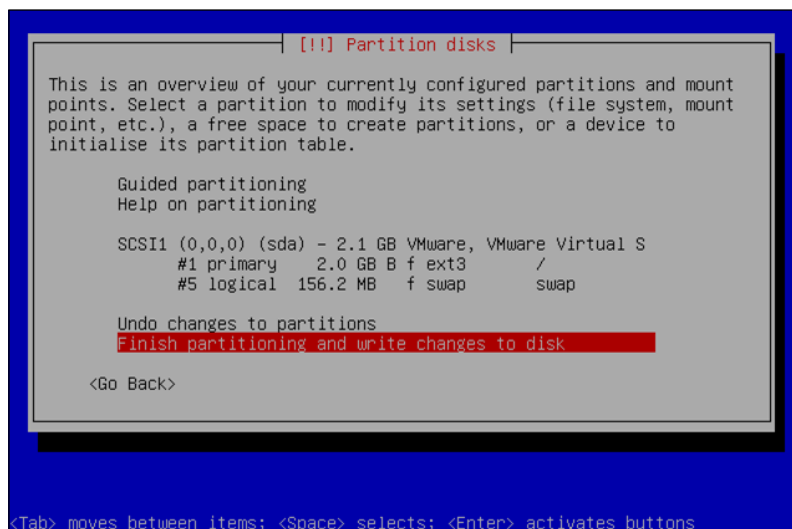
Now select the disk being used to install Linux.

Step 8: Select All files in one partition (recommended for new users).



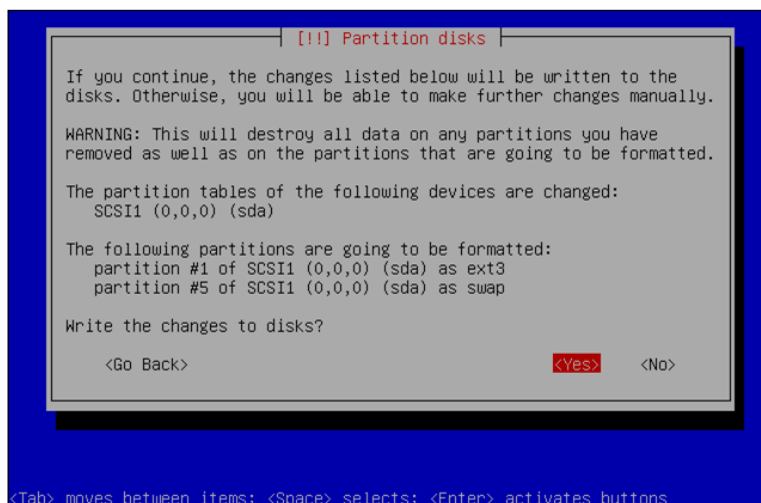
Again, you can choose how to partition the system. Let's stick with the default installation again. Some advanced users may want to change it.

Step 9: Now Finish the partitioning and write changes to disk.

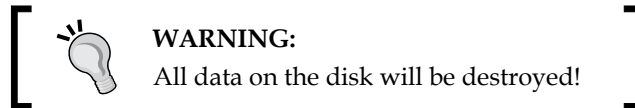


Now finish the partitioning step and write changes to the disk. However, never do it if you want to preserve your disk. After the partitioning, all the pre-existing content of the disk will be erased. So think carefully before doing this. I use VMware to test OpenSIPS; it is free and creates a virtual machine where I can work safely.

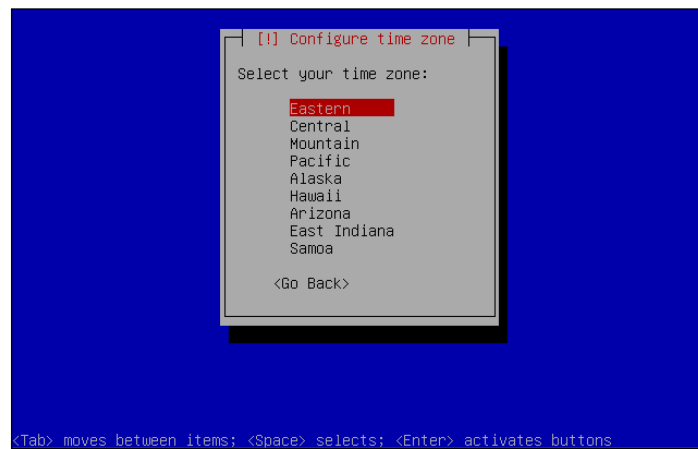
Step 10: Write changes to the disk.



Here comes the scary part. Confirm that you want to erase all the content of the disk. Well, think twice or even thrice before saying **Yes**.

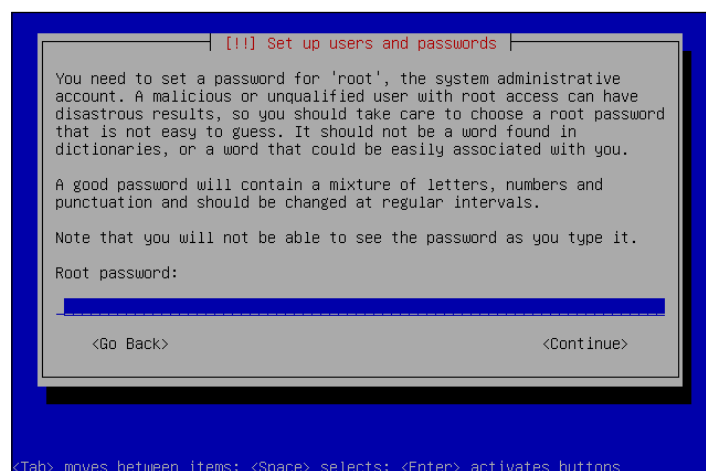


Step 11: Now Configure time zone.



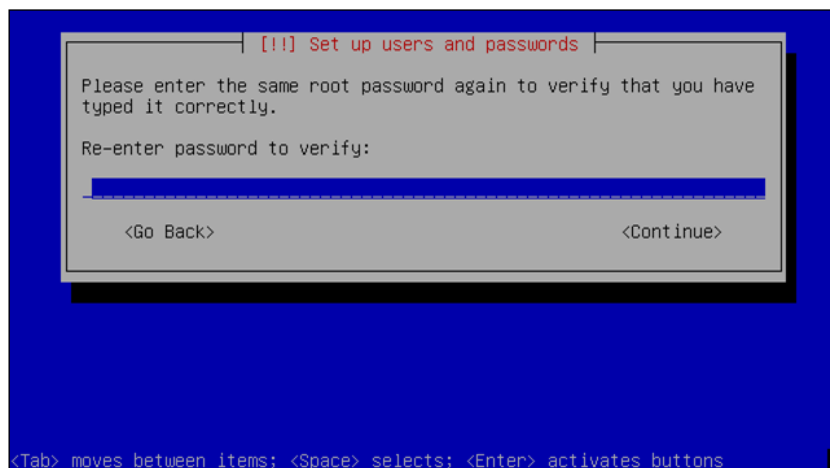
Select the time zone. It is important to have the correct time zone, mainly for reports. If you don't see it correctly, you will end up with voicemail messages having the wrong time.

Step 12: Set the Root password to OpenSIPs.



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

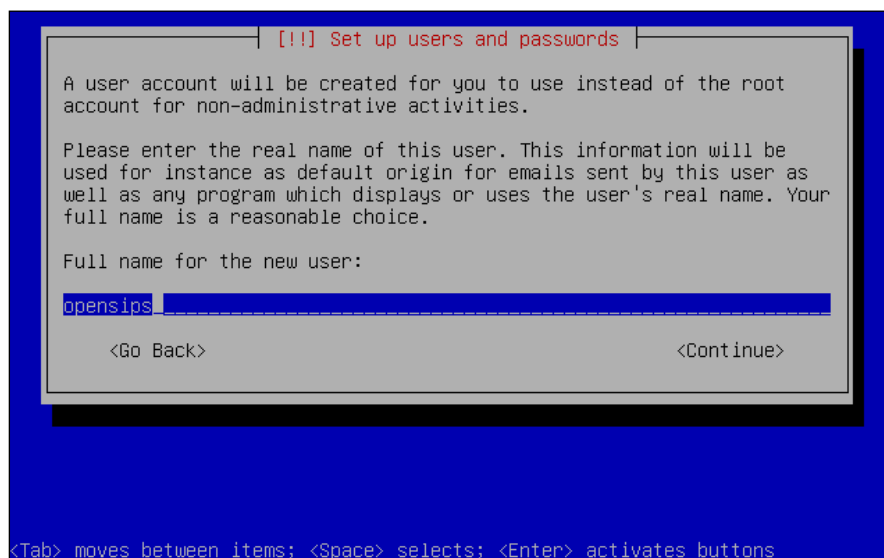
Step 13: Re-enter password to verify.



The screenshot shows a terminal window with a blue background. A gray dialog box titled "[!!] Set up users and passwords" is centered. Inside the dialog, the text reads: "Please enter the same root password again to verify that you have typed it correctly." followed by "Re-enter password to verify:". Below this text is a blue password input field. At the bottom of the dialog are two buttons: "<Go Back>" on the left and "<Continue>" on the right. Below the dialog box, a line of text in the terminal reads: "<Tab> moves between items; <Space> selects; <Enter> activates buttons".

Please re-enter the password for confirmation purpose. Try to use a password that is hard to crack (a password having minimum eight characters comprising of letters, numbers, and some special characters such as "*" or "#").

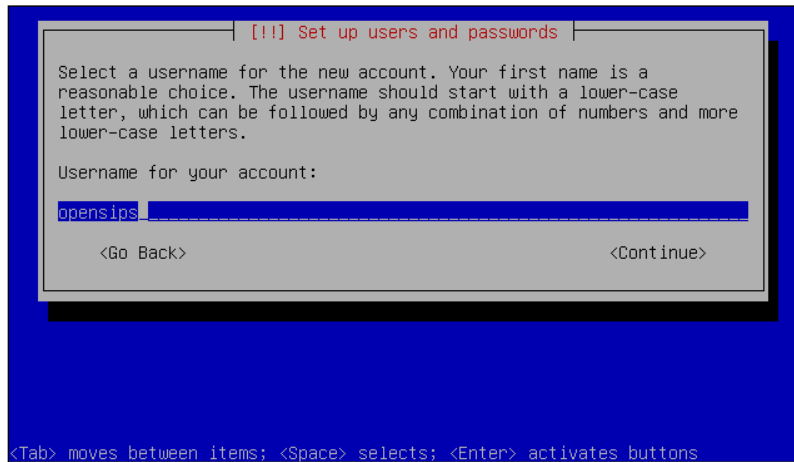
Step 14: Enter the Full name for the new user as opensips.



The screenshot shows a terminal window with a blue background. A gray dialog box titled "[!!] Set up users and passwords" is centered. Inside the dialog, the text reads: "A user account will be created for you to use instead of the root account for non-administrative activities." followed by "Please enter the real name of this user. This information will be used for instance as default origin for emails sent by this user as well as any program which displays or uses the user's real name. Your full name is a reasonable choice." followed by "Full name for the new user:". Below this text is a blue input field containing the text "opensips". At the bottom of the dialog are two buttons: "<Go Back>" on the left and "<Continue>" on the right. Below the dialog box, a line of text in the terminal reads: "<Tab> moves between items; <Space> selects; <Enter> activates buttons".

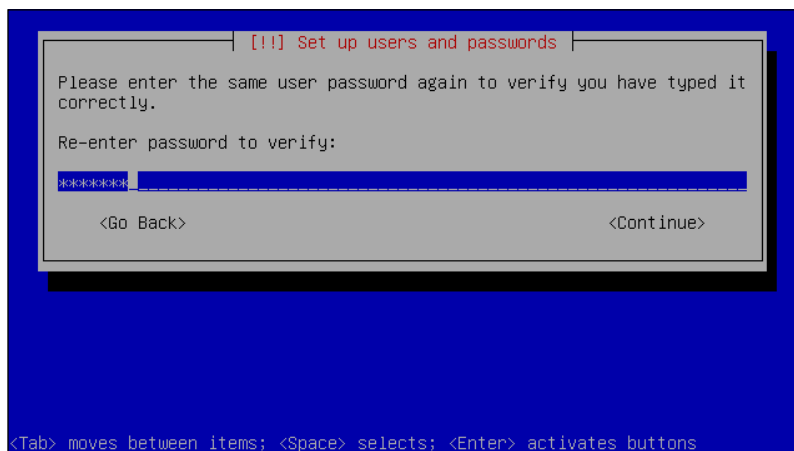
Some systems require you to create at least one user. Let's do it, starting with the full username.

Step 15: Enter the name for the user account as **opensips**.



The screenshot shows a terminal window with a blue background. At the top, a title bar reads "[!!] Set up users and passwords". The main text area contains the following instructions: "Select a username for the new account. Your first name is a reasonable choice. The username should start with a lower-case letter, which can be followed by any combination of numbers and more lower-case letters." Below this, it says "Username for your account:". A text input field contains the word "opensips". At the bottom of the window, there are two buttons: "<Go Back>" on the left and "<Continue>" on the right. A footer line at the very bottom of the terminal window reads: "<Tab> moves between items; <Space> selects; <Enter> activates buttons".

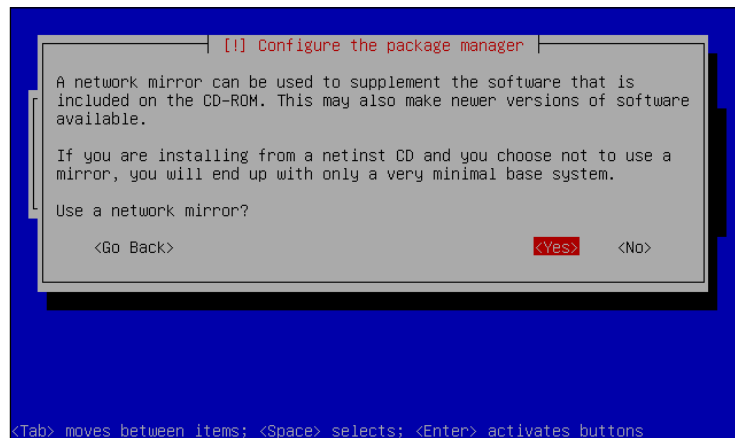
Step 16: Enter the password for the user account as **opensips** and re-enter to confirm.



The screenshot shows a terminal window with a blue background. At the top, a title bar reads "[!!] Set up users and passwords". The main text area contains the following instructions: "Please enter the same user password again to verify you have typed it correctly." Below this, it says "Re-enter password to verify:". A text input field contains seven asterisks "*****". At the bottom of the window, there are two buttons: "<Go Back>" on the left and "<Continue>" on the right. A footer line at the very bottom of the terminal window reads: "<Tab> moves between items; <Space> selects; <Enter> activates buttons".

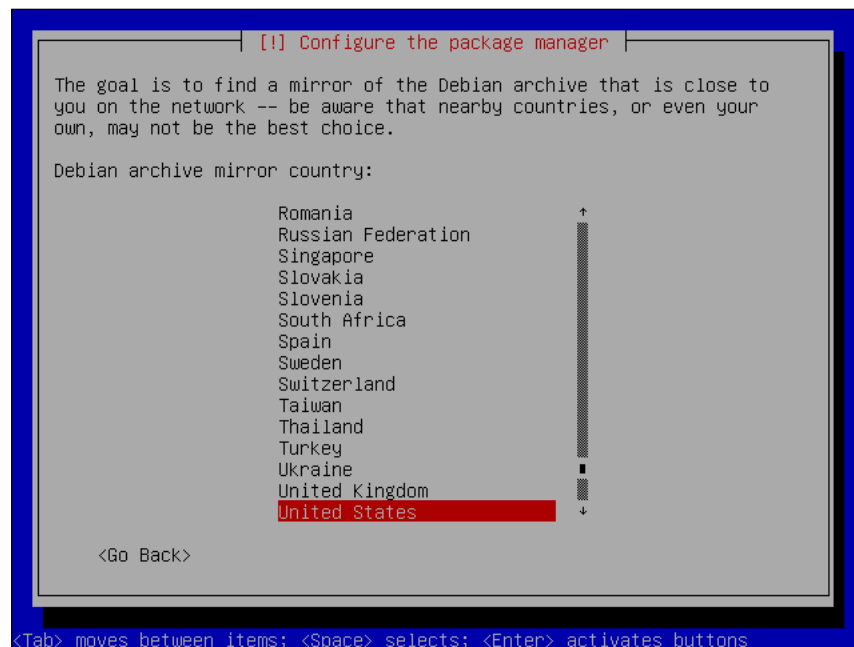
Enter the password and confirm it. Again, try to use a password that is hard to crack.

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



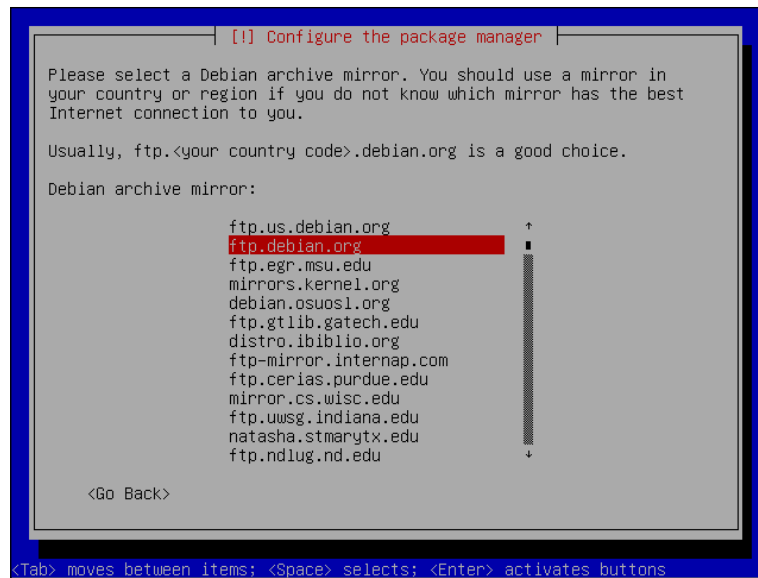
During the process of installation, we will use several packages distributed by Debian.

Step 18: Select a mirror country.

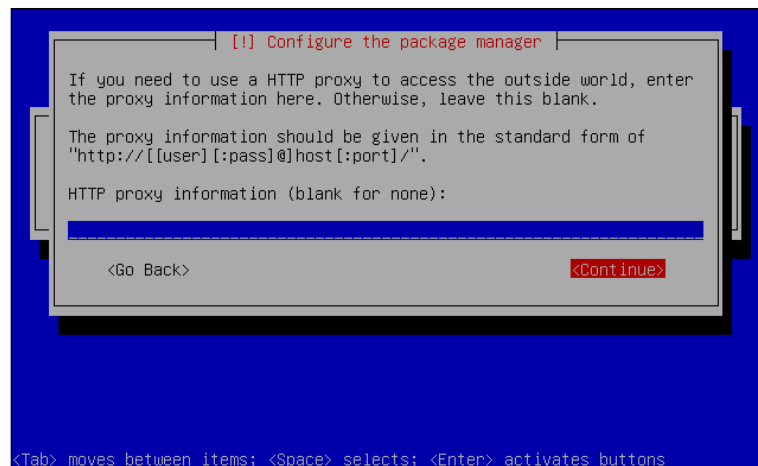


This screen will allow you to select the place from where you will download the packages.

Step 19: Select **ftp.debian.org** or your preferred mirror. Select the one nearest to your network to speed up the download of the packages.

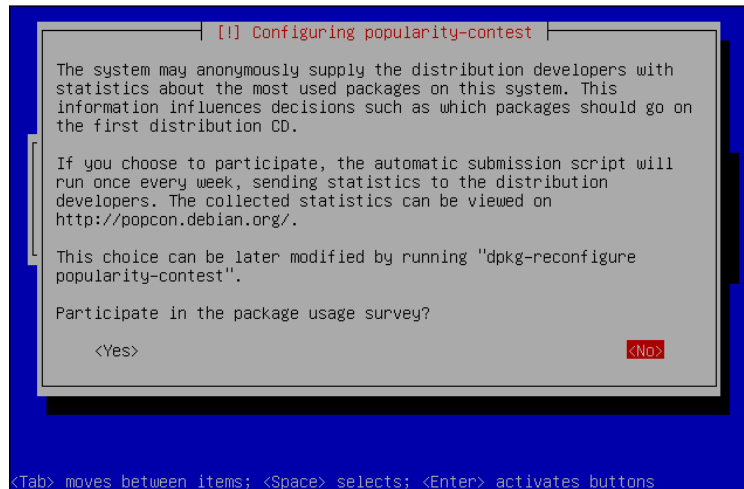


Step 20: Leave the **Http proxy information** blank or enter the appropriate parameters.



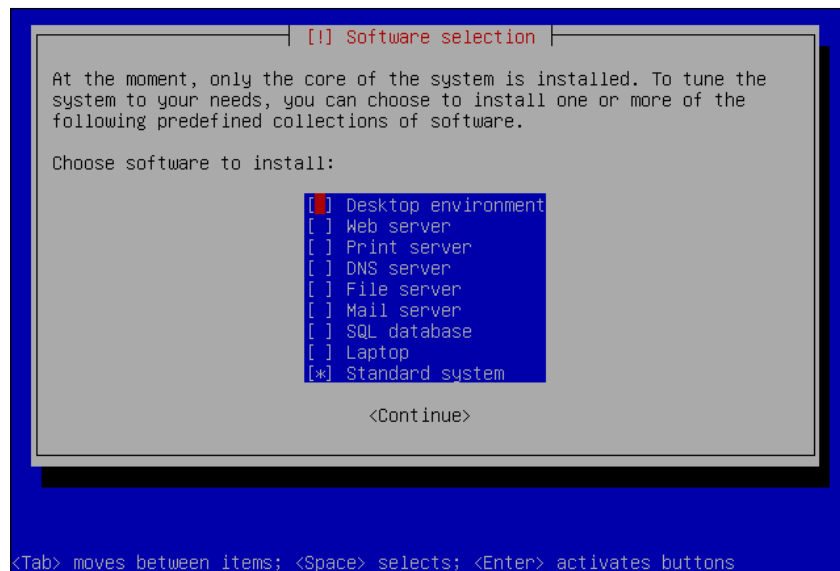
If you use an HTTP proxy such as Squid or Microsoft SA Server, please enter the appropriate parameters to allow Internet access for downloads.

Step 21: Select **No** to package popularity survey.



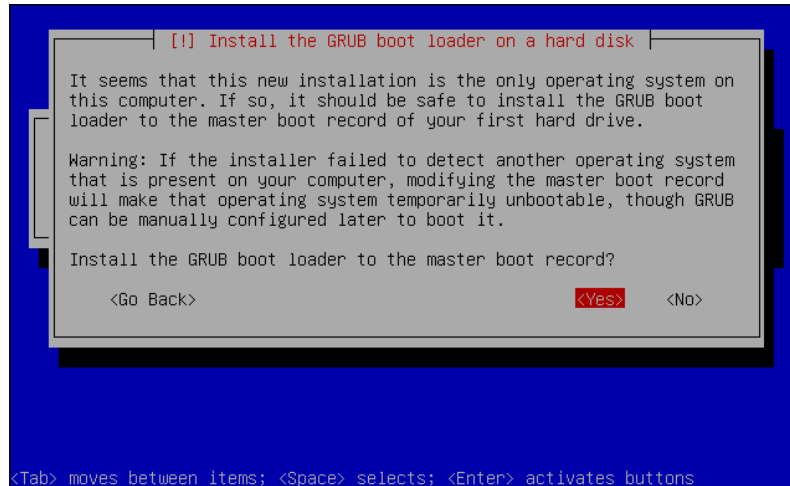
The popularity package survey generates statistics about the most downloaded packages.

Step 22: Select **Standard** system.



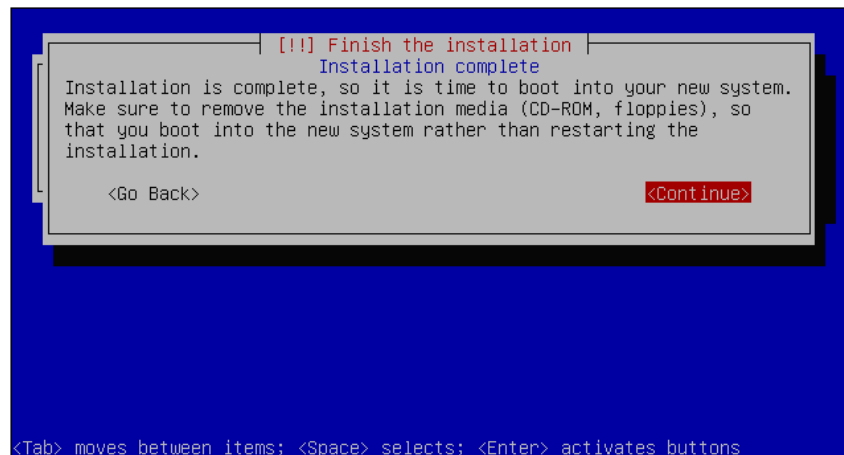
Debian comes in several predefined installations such as desktop, server, and standard system. The desktop installation, as an example, installs a GUI for Linux such as GNOME or KDE. We don't need this for our installation. Therefore, choose **Standard system**. Later, we will manually install components 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 do some tricks during the boot process.

Step 24: Finish the Installation.



Finish the installation and boot the system. The system will now reboot automatically.

Step 25: Just after the reboot, install SSH as follows:

```
apt-get install ssh
```

Downloading and installing OpenSIPS v1.6.x

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

Step 1: Install the dependencies.

```
apt-get install gcc bison flex make openssl libmysqlclient-dev
libradiusclient-ng2 libradiusclient-ng-dev mysql-server libxmlrpc-c3-dev
```



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 packages and decompress them. (Replace x with the current version.) There are two packages—`tls` and `no-tls`. The `tls` package contains the `tls` directory and some changes in the core to support encryption for the signaling.

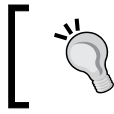
```
cd /usr/src
wget http://opensips.org/pub/opensips/1.6.x/src/opensips-1.6.x-tls_src.tar.gz
tar -xzf opensips-1.6.x-tls_src.tar.gz
```

Step 3: Compile and install the core and the modules. Include the `db_mysql` and `aaa_radius` modules.

```
cd opensips-1.6.x.tls
make prefix=/ all include_modules="db_mysql aaa_radius"
make prefix=/ install include_modules="db_mysql aaa_radius"
```

Step 4: Make the required adjustments

```
mkdir /var/run/opensips
```

Warning:

The detailed instructions change often. Please check the OpenSIPS website for updates.

OpenSIPS console

OpenSIPS comes with a new administration utility called `osipsconsole`. This utility is written in the PERL language and uses some additional libraries. To install, carry out the following instructions:

Step 1: Download the dependencies.

```
apt-get install libdbi-perl libdbd-mysql-perl libfrontier-rpc-perl
libterm-readline-gnu-perl
```

Step 2: Try running the console

```
osipsconsole
```

In the console prompt, try using the `help` and `quit` commands.

Lab—running OpenSIPS at the Linux boot

In order to run OpenSIPS at the Linux boot, perform the following steps:

Step 1: Include OpenSIPS in the Linux boot

```
cd /usr/src/opensips-1.6.x-tls/packaging/debian
cp opensips.default /etc/default/opensips
cp opensips.init /etc/init.d/opensips
update-rc.d opensips defaults 99
```

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

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

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

Step 4: Edit `/etc/default/opensips`, and change the memory parameter to 128 MB and `RUN_OPENSIPS` to `yes`.

Step 5: Edit the `init` script to make sure that the daemon is pointing to the correct directory.

```
vi /etc/init.d/opensips
```

The file before making changes:

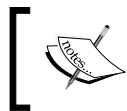
```
DAEMON=/usr/sbin/opensips
```

The file after making changes:

```
DAEMON=/sbin/opensips
```

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

```
ps -ef | grep opensips
```



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

OpenSIPS v1.6.x directory structure

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

Configuration files (`etc/opensips`)

These are the files copied to this directory. The files include the RADIUS dictionary to be used for OpenSIPS, the main configuration file `opensips.cfg`, the `opensipsctl` resource file, `opensipsctlrc`, and the `osipsconsole` resource file, `osipconsole.rc`.

```
opensips-1:/etc/opensips# ls -l
```

```
total 20
-rw-r--r-- 1 root staff 1559 2009-10-09 16:59 dictionary.opensips
-rw-r--r-- 1 root staff 1559 2009-10-09 17:04 dictionary.opensips.
sample
-rw----- 1 root staff 12437 2009-10-09 18:34 opensips.cfg
-rw-r--r-- 1 root staff 3661 2009-08-24 11:27 opensipsctlrc
-rw-r--r-- 1 root staff 2878 2009-05-19 14:02 osipsconsole.rc
```

Modules (/lib/opensips/modules)

This directory contains all the modules compiled for OpenSIPS. It is where you should look for missing modules.

```
opensips-1:/lib/opensips/modules# ls
```

aaa_radius.so	diversion.so	options.so	rr.so
acc.so	domainpolicy.so	path.so	seas.so
alias_db.so	domain.so	pdt.so	signaling.so
auth_aaa.so	drouting.so	peering.so	siptrace.so
auth_db.so	enum.so	permissions.so	sl.so
auth_diameter.so	exec.so	pike.so	sms.so
auth.so	gflags.so	presence_dialoginfo.so	speeddial.so
avpops.so	group.so	presence_mwi.so	sst.so
b2b_entities.so	imc.so	presence.so	statistics.so
benchmark.so	lcr.so	presence_xcapdiff.so	stun.so
call_control.so	load_balancer.so	presence_xml.so	textops.so
cfgutils.so	localcache.so	pua_bla.so	tm.so
closeddial.so	mangler.so	pua_dialoginfo.so	uac_redirect.so
db_flatstore.so	maxfwd.so	pua_mi.so	uac.so
db_mysql.so	mediaproxy.so	pua.so	uri_db.so
db_text.so	mi_datagram.so	pua_usrloc.so	uri.so
db_virtual.so	mi_fifo.so	pua_xmpp.so	userblacklist.so
dialog.so	msilo.so	qos.so	usrloc.so
dialplan.so	nathelper.so	ratelimit.so	xlog.so
dispatcher.so	nat_traversal.so	registrar.so	

Binaries (/sbin)

This is where the binary files are found. It is useful to know where these directories are, in case you want to uninstall OpenSIPS.

```
opensips-1:/sbin# ls -l op*
```

```
total 2832
-rwxr-xr-x 1 root staff 2594007 2009-10-09 17:04 opensips
-rwxr-xr-x 1 root staff   52695 2009-10-09 17:04 opensipsctl
-rwxr-xr-x 1 root staff    6270 2009-10-09 17:04 opensipsdbctl
-rwxr-xr-x 1 root staff   13442 2009-10-09 17:04 opensipsunix
-rwxr-xr-x 1 root staff  212692 2009-10-09 17:04 osipsconsole
```

Log files

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

```
Sep 10 14:25:56 Opensips-1 Opensips: init_tcp: using epoll_lt as the io watch
method (auto detected)
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: INFO: statistics manager
successfully initialized
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: StateLess module -
initializing
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: TM - initializing...
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: Maxfwd module- initializing
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: INFO:ul_init_locks: locks
array size 512
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: TextOPS - initializing
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: INFO: udp_init: SO_RCVBUF is
initially 109568
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: INFO: udp_init: SO_RCVBUF is
finally 262142
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: INFO: udp_init: SO_RCVBUF is
initially 109568
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7791]: INFO: udp_init: SO_RCVBUF is
finally 262142
Sep 10 14:25:56 Opensips-1 /sbin/Opensips[7792]: INFO:mi_fifo:mi_child_
init(1): extra fifo listener processes created
```

Redirecting OpenSIPS log files

By default, the log files are sent to the syslog and are shown at /var/log/syslog. Sometimes this is not good, because you have OpenSIPS logs mixed with other system's logs. You may redirect the logs to a specific file by changing the syslog configuration file.

Step 1: Redirecting log files to local 0 facility.

Change the opensips.cfg file and include the following command:

```
log_facility=LOG_LOCAL0
```

Step 2: Create the log file in the /var/log directory

```
cd /var/log
```

```
touch opensips.log
```

Step 3: Change the file `/etc/syslog.conf` and include the highlighted command.

```
lpr.*                -/var/log/lpr.log
mail.*               -/var/log/mail.log
user.*               -/var/log/user.log
local0.*              -/var/log/opensips.log
```

Step 4: Restart the syslog server and OpenSIPS and check the file `/var/log/opensips.log`.

```
/etc/init.d/syslogd restart
/etc/init.d/opensips restart
cat /var/log/syslog
```

Startup options

OpenSIPS can be started using the `init` scripts or using the `opensipsctl` utility. If you start OpenSIPS using `init` scripts, you can stop OpenSIPS using `init` scripts only. The same is valid if you start using `opensipsctl` utility.

Starting, stopping, and restarting OpenSIPS using the `init` scripts:

```
/etc/init.d/opensips start|stop|restart
```

Starting, stopping, and restarting OpenSIPS using the `opensipsctl` utility:

```
opensipsctl start|stop|restart
```

The OpenSIPS executable has several startup options. The following options allow you to change the configuration of the DAEMON. Some of the most useful options 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 many other options that allow you to fine-tune your configuration. For each option, there is a related core parameter that you can put in the configuration file.

```
debian:/sbin# opensips -h
version: opensips 1.6.0-notls (i386/linux)
Usage: opensips -l address [-l address ...] [options]
```

Options:

-f	file	Configuration file (default //etc/opensips/opensips.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 localhost, -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.

Summary

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

4

Script and Routing Basics

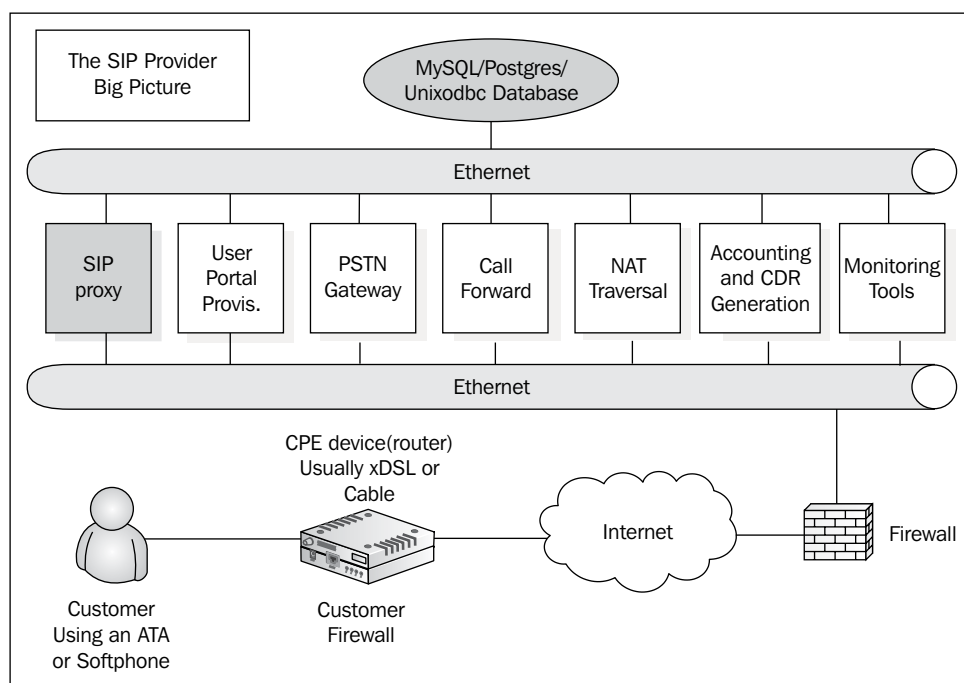
This is one of the most important chapters of this book. Please make sure you understand the concepts before going to the next chapter. In this chapter, we are going to show the basics needed to construct a working routing script. The OpenSIPS standard configuration file is installed at `/etc/opensips/opensips.cfg`. It is one of the simplest configuration files for OpenSIPS. It is the ideal script to start explaining the functioning of OpenSIPS. There are several topics 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 main global configuration parameters
- Compare these parameters with the ones in the default script
- Identify the modules loaded in the default script and the parameters of the script
- Identify the main routing statements available
- Understand the concept of variables, pseudo-variables, AVPs, and flags
- Identify the limitations of the standard routing configuration
- Use the `ngrep` utility to track SIP transactions

Where we are

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



Scripting OpenSIPS

There is basically only one configuration file for OpenSIPS. This file is named `opensips.cfg`. It contains several sections; such as "Global Parameters", "Load Modules", "Module Parameters", and "Routing Script". In order to create this file, you have several commands and functions derived from the core and the modules. We are going to show some of the most common functions for each section and then how this section is populated in the default script.

OpenSIPS comes with a minimum configuration provided in the installation. This simple configuration is a good starting point. After starting, OpenSIPS will:

- Establish itself as an SIP server on your network
- Allow you to connect UACs from your internal network
- Allow you to make calls between two different UACs

However, there are some things OpenSIPS still doesn't do:

- There is no authentication for phones or database support. This will be added later on in the book.
- There is no support for the PSTN. Phones can only communicate among themselves.

Global parameters

As the name implies, global parameters control the behavior of the daemon. The reference documentation for core parameters can be found at:

<http://www.opensips.org/index.php?n=Resources.DocsCoreFcn16>

We will show some of the most important parameters and the ones which are present in the standard configuration file.

Listen interfaces

This parameter specifies the IP interface to bind on for OpenSIPS to receive SIP traffic. You should set these parameters only for the interfaces really being used. If you leave them unchanged, the system will *autodiscover* and bind to all available interfaces, consuming more processes and resources than necessary. The `port` parameter defines the port the SIP server listens to. The default value for it is 5060.

```
listen=udp:192.168.152.148:5060
listen=tcp:192.168.152.148:5061
listen=tls:192.168.152.148:5062
port=5060
```

Logging

There are some parameters to control the logging system. One of most important is the `debug` parameter. You may use `debug=3` (the recommended production set), which is the default and will print only error and critical messages while `debug=9` (the recommended debugging set) will print all messages up to the debug level. You may change the level of debugging using an MI command such as `opensipsctl fifo debug 1`. The default is 2. The higher the number, the more will be the information written to the log. With a debug value of 4, the system's performance can become sluggish. The log levels are:

- `L_ALERT (-3)` – this level should be used if the error requires immediate action

- L_CRIT (-2) – this level should be used if the error is a critical situation
- L_ERR (-1) – this level should be used to report errors during data processing which 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

log_facility and log name are related to syslog. Check the syslog documentation (<http://www.aboutdebian.com/syslog.htm>) for further details.

Core dump is a useful resource when you want to debug crashes. You can use `gdb` to debug what module and specific resource caused the problem. If required, enable it using `disable_core_dump=no`.



Use `tail /var/log/syslog -n100` to see the last 100 lines of the syslog or `tail /var/log/syslog -f` to see the log scrolling.

Number of processes

The `fork` directive tells the OpenSIPS process to execute in foreground or background. To operate in background, set `fork=yes`. Sometimes you will find it useful to start in the foreground to locate script errors. If `fork` is disabled, OpenSIPS will not be able to listen on more than one interface and the TCP/TLS support will be automatically disabled. In a single process mode, only one UDP interface is accepted. You can disable the listening on selected protocols. Disabling UDP is not an option and it is against the RFC3265 (UDP is mandatory). The `children` directive informs OpenSIPS of how many *child* processes per interface to create in order to 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.

```
fork = yes
children = 4 #total number of UDP SIP worker processes per interface
tcp_children=6 #total number of TCP SIP worker processes in total
disable_tcp=no
disable_tls=no
```

Daemon options

Daemon options can be used to set the user and group and they are useful to avoid running OpenSIPS as the root of the system.

```
gid/group=sip # unix group
uid/user=sip # unix user
wdir="/" # working directory
chroot="/usr/local/opensips-1.6"
```

SIP identity

The SIP identity is useful to set the standard parameters used inside the SIP requests and responses. You can use it to hide the identity of the software you are using.

```
server_header="Server: My openSIPS
#default is "OpenSIPS (<version> (<arch>/<os>))"
server_signature = yes
user_agent_header="User-Agent: My openSIPS"
```

Miscellaneous

The alias parameter is very important; it defines the domains being served. Later you can check whether the request is coming from or going to a served domain by using the core value `myself`. Using `auto_aliases=yes` will make the system discover the aliases using reversed DNS. DNS failover is used to failover destinations using the domain name system. SIP warning can help you debug issues by adding a debug header in the SIP replies.

```
alias="mydomain.sip" # to set alias hostnames for the server
auto_aliases=no # discover aliases via reversed DNS
disable_dns_failover = yes
sip_warning=yes #add a debugging header in replies
```

Standard script for global parameters

Following is the standard configuration with comments:

```
##### Global Parameters #####
debug=3 # set the debug leve to 3
log_stderr=no # log to syslog
log_facility=LOG_LOCAL0 # Log to facility LOG_LOCAL0
fork=yes # Run as a daemon
children=4 # Open 4 child process for
each UDP address

/* uncomment the next line to disable TCP (default on) */
#disable_tcp=yes

/* uncomment the next line to enable the auto temporary blacklisting
of
```

```
    not available destinations (default disabled) */
#disable_dns_blacklist=no

/* uncomment the next line to enable IPv6 lookup after IPv4 dns
   lookup failures (default disabled) */
#dns_try_ipv6=yes
/* uncomment the next line to disable the auto discovery of local
aliases
   based on revers DNS on IPs (default on) */
#auto_aliases=no

/* uncomment the following lines to enable TLS support (default off)
*/
#disable_tls = no
#listen = tls:your_IP:5061
#tls_verify_server = 1
#tls_verify_client = 1
#tls_require_client_certificate = 0
#tls_method = TLSv1
#tls_certificate = "//etc/opensips/tls/user/user-cert.pem"
#tls_private_key = "//etc/opensips/tls/user/user-privkey.pem"
#tls_ca_list = "//etc/opensips/tls/user/user-calist.pem"

port=5060 # Run on port 5060

/* uncomment and configure the following line if you want opensips to
   bind on a specific interface/port/proto (default bind on all
available) */
#listen=udp:192.168.1.2:5060
```

Modules and their parameters

You can load modules simply using the instruction `loadmodule` as described in this section. The `mpath` statement simplifies the task by setting the module's path.

```
mpath="/usr/lib/opensips/modules/"
loadmodule "tm.so"
```

In order to configure the module parameters, you can use the `modparam` statement. This statement has three parameters – the module name, the module parameter, and the parameter value:

```
modparam("tm", "fr_inv_timer", 20)
```

It is also possible to set the same parameter for multiple modules:

```
modparam("usrloc|auth_db", "db_url",
"mysql:opensips@localhost/opensips")
```

Standard configuration for modules and parameters

The following are the main modules used in the default script. In this book, you will have the opportunity to see and learn about some of them. The modules reference documentation can be found at <http://www.opensips.org/Resources/DocsCookbooks>.

```
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"
    (merged with uri.so in 1.6)
loadmodule "uri.so"
loadmodule "xlog.so"
loadmodule "acc.so"
loadmodule "signaling.so"
```

The above lines load OpenSIPS 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/opensips_fifo")
```

They also have the name of the FIFO file to be created for listening and reading external commands.

```
modparam("rr", "enable_full_lr", 1)
modparam("rr", "append_fromtag", 0)
```

The first statement sets the `enable_full_lr` parameter of the module `rr` (record routing) to 1. It tells OpenSIPS to be fully compliant with older SIP clients that do not manage `record_route` header fields. If set to 1, then `;lr=on` will be used instead of just `;lr`. If the `append_fromtag` parameter is turned on, the request's from-tag is appended to record-route; that's useful for understanding whether subsequent requests (such as BYE) come from the caller (that is, route's from-tag==BYE's from-tag) or the callee (that is, route's from-tag==BYE's to-tag)

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

The `modparam` directive configures the corresponding module. The `usrloc` module in the line is responsible for the location service. In other words, when a UAC registers to an SIP proxy, the OpenSIPS will save their contact information, also known as **address-of-record (AOR)** to the location indicated by the `db_mode` parameter. The location of this table depends on the value of the `db_mode` parameter. A `db_mode` parameter set to 0 indicates that this data would not be saved into a database. In other words, if OpenSIPS is turned off, all the records will be lost.

```
We will use the URI modules because of the totag() function.
modparam("acc", "early_media", 1)
modparam("acc", "report_ack", 1)
modparam("acc", "report_cancels", 1)
/* by default ww do not adjust the direct of the sequential requests.
   if you enable this parameter, be sure the enable "append_fromtag"
   in "rr" module */
modparam("acc", "detect_direction", 0)
/* account triggers (flags) */
modparam("acc", "failed_transaction_flag", 3)
modparam("acc", "log_flag", 1)
modparam("acc", "log_missed_flag", 2)
/* uncomment the following lines to enable DB accounting also */
modparam("acc", "db_flag", 1)
modparam("acc", "db_missed_flag", 2)
```

The above parameters implement accounting for `syslog`. In Chapter 11, *Monitoring Tools*, we will see all of them in detail.

Scripting basics

Before stepping into the routing logic, it is important to know the main statements available to build it. OpenSIPS uses a specialized scripting language similar to C, but focused on the task of routing SIP requests and handling SIP replies.

Some functions and values are provided by the core, while some are provided by modules. So it is important to understand what is available from the core and from modules. Frequently you will try to start OpenSIPS and receive an error message stating that a determined function is not available. In most cases it is just the module that is not loaded.

Core functions

Core functions are functions that are available without any module loaded. Some are critical for the behavior of the script. Core functions have no restrictions about the number of parameters they can accept. Module functions can have a maximum of six parameters. Using the core functions you can decide, for example, what to do with the script.

- `forward()` ;: Route the request "stateless"(based on R-URI)
- `drop()` ;: Stop the execution of the configuration script and alter the implicit action which will be taken afterwards
- `exit()` ;: End the script processing now and send

Other important core functions are `seturi()`, `setflag()`, `isflagset()`, `strip()`, `prefix()` ;, and `rewritehostport()`.

Core values

Some values in the script are predefined and can be very useful. Some examples are:

- `INET/INET6` — set if the protocol is IPv4 or IPv6
- `TCP/TLS/UDP` — set depending on the protocol used
- `myself` is a reference to the list of local IP addresses, hostnames, and aliases

Core keywords

Core keywords are used to identify values in the SIP message. Some examples are:

- `af` — Address family (INET/INET6)
- `proto` — Protocol (TCP/TLS/UDP)
- `dst_ip` — The IP of the local interface at which the SIP message was received
- `method` — The variable is a reference to the SIP method of the message
- `Status` — If used in `onreply_route`, this variable is a reference to the status code of the reply
- `rcode` — It represents the value returned by last function executed
- `uri` — This variable can be used to test the value of the request URI
- `from_uri` — This script variable is a reference to the URI of the FROM header
- `to_uri` — This variable can be used to test the value of the URI from the TO header

Some coding examples using core keywords and core values are as follows:

```
if (af==INET6) {  
    log("Message received over IPv6 link\n");  
};  
if (is_method("INVITE") && from_uri=~".*@opensips.org")  
{  
    log("the caller is from opensips.org\n");  
};
```

Pseudo-variables

Pseudo-variables are system variables that you can use in your script to access various types of information from SIP messages (such as headers, R-URI, and source IP) or OpenSIPS (such as time, and process ID). These pseudo-variables can be directly used from the script (using assignments and text operations) or can be passed as parameters to script functions. The variables are evaluated at runtime based on the context and the processed SIP message. Some modules can receive pseudo-variables, such as:

- ACC
- AVPOPS
- TEXTOPS
- UAC
- XLOG

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 predefined set of pseudo-variables. The complete list of pseudo-variables in OpenSIPS 1.6 is available at:

<http://www.opensips.org/Resources/DocsCoreVar16>

Script variables

Script variables refer to variables that can be used in the configuration script. These kinds of variables are much faster than AVPs as they are attached only to the script. Script variables exist only during the script's execution and once the script ends, they are destroyed. A script variable can have numerical or string values.

\$var (name)

Some of the examples include:

- `$var(b)=1;`
- `$var(b)="1";`
- `$var(b)="$fu"+"$tu";`
- `$var(b)=1+2;`

The arithmetic operations available are:

- `+`: Plus
- `-`: Minus
- `/`: Divide
- `*`: Multiply
- `%`: Modulo division
- `|`: Bitwise OR
- `&`: Bitwise AND
- `^`: Bitwise XOR
- `~`: Bitwise NOT

String transformations available include:

- `{s.len}`
- `{s.int}`
- `{s.substr,offset,length}`
- `{s.select,index,separator}`
- `{uri.user}`
- `{uri.host}`
- `{uri.params}`
- `{param.value,name}` - returns the value of parameter name

For example:

```
"a=1;b=2;c=3"{param.value,c} = "3"
```

Attribute-Value Pair (AVP) overview

An **Attribute-value pair (AVP)** is a variable that is attached to the SIP message of the SIP transaction the message belongs to (if in the *stateful* mode) – so AVPs are transaction-persistent variables. The AVP is allocated when the transaction begins and unallocated when it's completed. The AVP name can be a number or a string and the value of the AVP can also be a number or a string.

The introduction of AVPs in OpenSIPS processing has created several new possibilities for services' implementation and user preferences 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 variables:

```
$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.

Example:

```
$avp(i:700)
$avp(s:blacklist)
```

For those who know Asterisk, the AVPOPS module is for OpenSIPS 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 of 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*: Stores AVPs into the database
- *avp_db_delete*: Deletes AVPs from the database
- *avp_db_query*: Makes a database query and stores the results in AVP
- *avp_delete*: Deletes AVPs from memory
- *avp_pushto*: Pushes the AVP values into the SIP message
- *avp_check*: Checks the value of the AVP using an operator (equal, greater than, and a value). For example, `avp_check("i:500", "lt/i:501");`.
- *avp_copy*: Copies an AVP to another
- *avp_printf*: Formats a string to an AVP

- `avp_subst`: Finds and replaces values into an AVP
- `avp_op`: Allows math operations on the AVPs
- `is_avp_set`: Checks if this AVP's name is set
- `avp_print`: Prints 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_push` to that will be used in 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. However, the loading of AVPs from the database is very confusing. The default table is the `usr_preference` (user preferences) table. 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 forwarding, we have a call forward associated to user. It is actually a user preference. Let's check the `usr_preference` table structure.

id	uuid	username	domain	attribute	type	value	Last_modified
	1001			callfwd	0	sip:1004@domain	

- The `id` is an auto-increment field
- `uuid` is a unique user identifier
- `username` is for the username
- `domain` is for the 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 to any of these parameters. You can associate an AVP with a `uuid` (unique user id), to a username (single-domain setup), or with username and domain (multi-domain setup).

Flags

Very often you will see the use of script flags. These flags are used to trigger some processes such as accounting, dialog control, NAT handling, and others. There are three types of flags – message flags, script flags, and branch flags. See the following table:

Type	Persistence	Function	Purpose
Message flag	Transaction	<code>setflag(flag_idx)</code>	To activate some functions in the transaction level
Branch flag	Branch	<code>setbflag(flag_idx)</code>	To activate some functions in the branch level
Script flag	Top-level Routing	<code>setsflag(flag_idx)</code>	Used to save the other flags, valid only for scripting

You can check the decimal values of the flags set using the pseudo-variables – `$mf` (message flags), `$bf` (branch flags), and `$sf` (script flags).

The module GFLAGS

The module GFLAGS allow the use of external flags. These flags can be set using the MI commands. GFLAGS can be set using the functions `set_gflag()`, `is_gflag()`, and reset using `reset_gflag()` inside the script or from an external program using the MI interface.

Statements

Let us look at the following three statements.

if-else

Example:

```
if ( t_check_trans() ) {
    t_relay();
    exit;
} else {
    exit;
}
```

Switch

```
switch($retcode)
{
    case -1:
        log("process INVITE requests here\n");
        break;
    case 1:
        log("process REGISTER requests here\n");
        break;
    case 2:
    case 3:
        log("process SUBSCRIBE and NOTIFY requests here\n");
        break;
    default:
        log("process other requests here\n");
}
```

Subroutes

```
route[1]{
    if(is_method("INVITE"))
    {
        return(-1);
    };
    if(is_method("REGISTER"))
        return(1);
}
if(is_method("SUBSCRIBE"))
    return(2);
}
if(is_method("NOTIFY"))
    return(3);
}
return(-2);
}
```

Routing basics

It is hard to understand how to route packets without a small introduction about the principles of SIP routing. Pay special attention to the difference between initial and sequential requests.

Routing requests and replies

The requests are routed using some mechanisms in the OpenSIPS scripts; usually for inter-domain calls, we use a DNS server to discover the address for the destination, while intra-domain calls are often routed using the user location table. The replies are routed back based on the VIA headers inserted during the path of the request. For *stateful* routing, the transaction is matched based on the branch parameter in the VIA header.

See the following example, the address of the SIP proxy is 192.168.1.201:5060, the IP address of the user 1000 is 192.168.1.159:39132 and the IP address of the user 1001 is 192.168.1.159:5060. All of the other headers were omitted for simplicity.

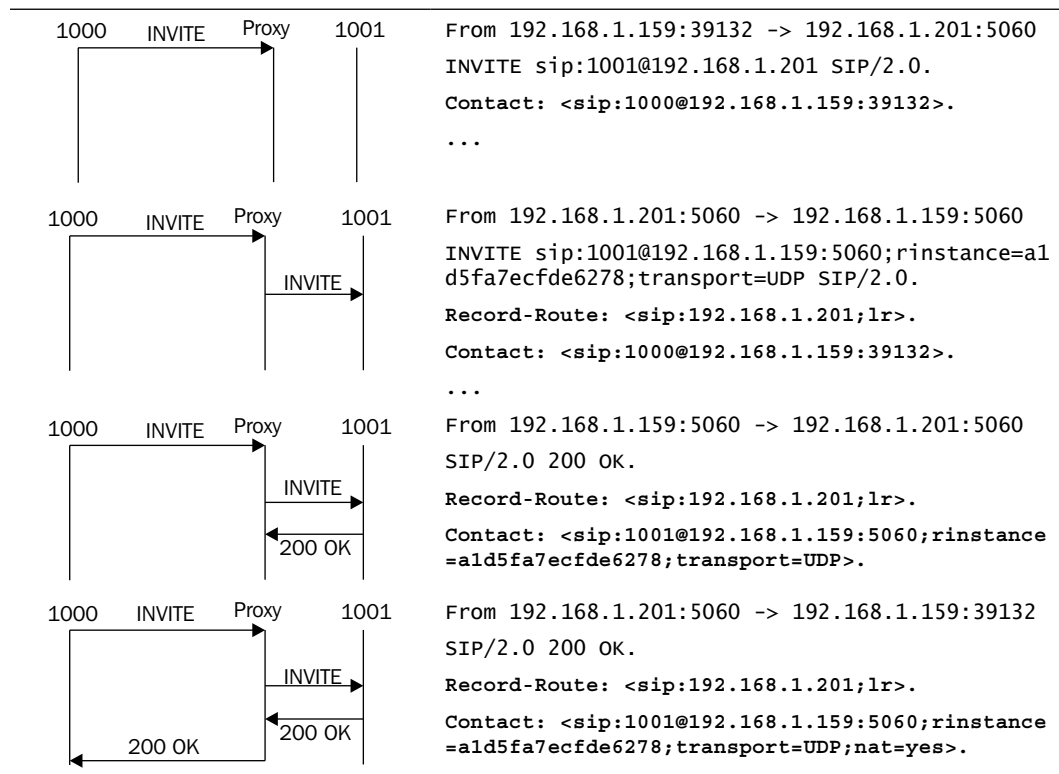
Image	Header
<pre> sequenceDiagram participant 1000 participant Proxy participant 1001 1000->>Proxy: INVITE Proxy->>1001: INVITE </pre>	<pre> From 192.168.1.159:39132 -> 192.168.1.201:5060 INVITE sip:1001@192.168.1.201 SIP/2.0. Via: SIP/2.0/UDP 192.168.1.159:39132;branch=z9hG4bK-d87543-f467f33a206c333a-1--d87543-;rport. ... </pre>
<pre> sequenceDiagram participant 1000 participant Proxy participant 1001 1000->>Proxy: INVITE Proxy->>1001: INVITE </pre>	<pre> From 192.168.1.201:5060 -> 192.168.1.159:5060 INVITE sip:1001@192.168.1.159:5060;rinstance=a1d5fa7ecfde6278;transport=UDP SIP/2.0. Via: SIP/2.0/UDP 192.168.1.201;branch=z9hG4bKf5b7.34401122.0. Via: SIP/2.0/UDP 192.168.1.159:39132;received=192.168.1.159;branch=z9hG4bK-d87543-f467f33a206c333a-1--d87543-;rport=39132. ... </pre>
<pre> sequenceDiagram participant 1000 participant Proxy participant 1001 1000->>Proxy: INVITE Proxy->>1001: INVITE 1001-->>Proxy: 200 OK </pre>	<pre> From 192.168.1.159:5060 -> 192.168.1.201:5060 SIP/2.0 200 OK. Via: SIP/2.0/UDP 192.168.1.201;branch=z9hG4bKf5b7.34401122.0. Via: SIP/2.0/UDP 192.168.1.159:39132;received=192.168.1.159;branch=z9hG4bK-d87543-f467f33a206c333a-1--d87543-;rport=39132. ... </pre>
<pre> sequenceDiagram participant 1000 participant Proxy participant 1001 1000->>Proxy: INVITE Proxy->>1001: INVITE 1001-->>Proxy: 200 OK Proxy-->>1000: 200 OK </pre>	<pre> From 192.168.1.201:5060 -> 192.168.1.159:39132 SIP/2.0 200 OK. Via: SIP/2.0/UDP 192.168.1.159:39132;received=192.168.1.159;branch=z9hG4bK-d87543-f467f33a206c333a-1--d87543-;rport=39132. ... </pre>

Observe the content of the VIA headers, they have enough information to route the replies back. Another important parameter is the `branch`—it is used for transaction matching in the *stateful* mode. The `received` and `rport` fields are used by the RFC3581 for SIP NAT traversal. We will cover NAT traversal later in this book.

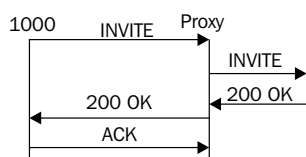
Initial and sequential requests

Before proceeding, it is crucial to understand the difference between initial requests and sequential requests. The routing logic is different, depending on the kind of request.

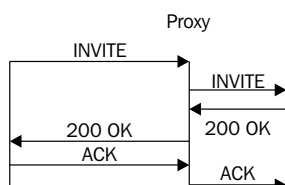
Initial requests are routed based on a discovery mechanism, usually location table or DNS, in order to locate the destination. The initial request records the relevant hops (SIP proxies) in the path, from source to destination using the mechanism called **record routing**. An example of initial request is the initial INVITE.



Sequential requests are routed based on the information collected by the initial requests. These collected routes are known as the **route set**. In the script, you will use the `loose_route()` function to route based on the route set. Sequential requests are usually ACKs, BYEs, and re-INVITES. You can distinguish between initial and sequential requests based on the TAG parameter in the TO header. The sequential requests are routed based on the Route header and the URI. In other words, the UAC receives the route set based on the Record-Route and Contact headers. Once the client has obtained the route set, it mirrors the Contact header in the request URI and the Record-Route header in the Route header. For the proxy, it is a lot faster to use the routing information contained in the request using the `loose_route` function than to rediscover the destination using the user location table or a DNS request.



```
From 192.168.1.159:39132 -> 192.168.1.201:5060
ACK sip:1001@192.168.1.159:5060;rinstance=ald5f
a7ecfde6278;transport=UDP;nat=yes SIP/2.0.
Route: <sip:192.168.1.201;lr>.
...
```



```
From 192.168.1.201:5060 -> 192.168.1.159:5060
ACK sip:1001@192.168.1.159:5060;rinstance=ald5f
a7ecfde6278;transport=UDP;nat=yes SIP/2.0.
```

Sample route script

The routing script `opensips.cfg` is available just after the installation. The commands will be explained just next to the script. This is the beginning of the routing logic for a SIP request. The block starts with a `{`. The SIP requests will be processed in this block. The following is an overview of the script:

```
# 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;
};
```

When a request gets into the main routing block, some checks are performed. `mf_process_maxfwd_header` for security check and should always be included in the first line of the main routing block. This function is exposed by the `MaxForward` (`maxfwd.so`) module and is used to register how many times a SIP request has passed over the SIP server. It is used to avoid loops.

If a looping situation occurs, OpenSIPS will inform a SIP client that an error occurred. The `sl_send_reply` function is responsible for doing this job. The `sl_send_reply` function is exposed by the `stateless` (`sl.so`) module and what it does is to send a stateless request to the SIP client. This means that OpenSIPS will not wait for a message acknowledgement. Some phones will show this message on the display. The `exit` instruction will tell OpenSIPS to stop the request processing and exit.

```
if (has_totag()) {
    # sequential request withing a dialog should
    # take the path determined by record-routing
    if (loose_route()) {
        if (is_method("BYE")) {
            setflag(1); # do accounting ...
            setflag(3); # ... even if the transaction fails
        } else if (is_method("INVITE")) {
            # even if in most of the cases is useless, do RR for
            # re-INVITEs alos, as some buggy clients do change route set
            # during the dialog.
            record_route();
        }
        # route it out to whatever destination was set by loose_route()
        # in $du (destination URI).
        route(1);
    } else {

        if ( is_method("ACK") ) {
            if ( t_check_trans() ) {
                # non loose-route, but stateful ACK; must be an ACK after
                # a 487 or e.g. 404 from upstream server
                t_relay();
                exit;
            } else {
                # ACK without matching transaction ->
                # ignore and discard
                exit;
            }
        }
        sl_send_reply("404","Not here");
    }
    exit;
}
```

The above section is used to process sequential requests. If a packet has a tag in the `To:` header field, this indicates that this request is not an initial request, but a sequential request instead. Sequential requests are usually loose routed. Requests such as `BYE` and `CANCEL` for existing transactions will be forwarded. Packets with the `To:` tag, but without the `;lr` will be discarded with an error message. The sequence handles ACKs, discarding ACKs without a matching transaction.

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

This is the handling of `CANCEL` requests. You don't need to route the `CANCEL` requests manually. If it belongs to an existing `INVITE` transaction you may only need to relay it to the destination the `INVITE` was already routed to.

```
t_check_trans();
```

The `t_check_trans()` function is used to determine if a specific request belongs to a transaction. In this point of the script, the function is being used to stop the script if the request is a retransmission.

```
# authenticate if from local subscriber (uncomment to enable auth)
# authenticate all initial non-REGISTER request that pretend to be
# generated by local subscriber (domain from FROM URI is local)
##if (!(method=="REGISTER") && from_uri==myself) /*no multidomain*/
##if (!(method=="REGISTER") && is_from_local()) /*multidomain*/
##{
##    if (!proxy_authorize("", "subscriber")) {
##        proxy_challenge("", "0");
##        exit;
##    }
##    if (!db_check_from()) {
##        sl_send_reply("403","Forbidden auth ID");
##        exit;
##    }
##
##    consume_credentials();
##    # caller authenticated
##}
```

This is the authentication section for non-register requests. We will explain these commands in the next chapter. The default script does not authenticate any requests.

```
# preloaded route checking
if (loose_route())
{
    xlog("L_ERR",
        "Attempt to route with preloaded Route's [$fu/$tu/$ru/$ci]");
    if (!is_method("ACK"))
        sl_send_reply("403", "Preload Route denied");
    exit;
}
```

The section above checks for requests without the To: tag but with Route: headers. If found, they are discarded, except for ACKs.

```
# record routing
if (!is_method("REGISTER|MESSAGE")) record_route();
```

If the request is not targeted to the own server, record the routes for later processing using `loose_route()`.

```
# account only INVITES
if (is_method("INVITE")) {
    setflag(1); # do accounting
}
```

Starting with 1.3, the default script is capable of accounting. Mark the INVITE requests with the flag 1 for accounting.

```
if (!uri==myself)
    ## replace with following line if multi-domain support is used
    ##if (!is_uri_host_local())
    {
        append_hf("P-hint: outbound\r\n");
        # if you have some interdomain connections via TLS
        ##if($rd=="tls_domain1.net") {
        ##    t_relay("tls:domain1.net");
        ##    exit;
        ##} else if($rd=="tls_domain2.net") {
        ##    t_relay("tls:domain2.net");
        ##    exit;
        ##}
        route(1);
    }
```

The previous code will handle the requests for a domain not served by our proxy. `if (!uri==myself)` forwards the request 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. Some lines are commented, they allow you to use TLS for external domains.

```
## uncomment this if you want to enable presence server
## and comment the next 'if' block
## NOTE: uncomment also the definition of route[2] from below
##if( is_method("PUBLISH|SUBSCRIBE"))
##      route(2);
if (is_method("PUBLISH"))
{
    sl_send_reply("503", "Service Unavailable");
    exit;
}
```

This is another interesting piece of code. You can opt to handle or not handle the presence features. Remove the comments (#) from the first two lines and comment the other three, and *voilà* you have a presence agent!

```
if (is_method("REGISTER"))
{
    # authenticate the REGISTER requests (uncomment to enable auth)
    ##if (!www_authorize("", "subscriber"))
    ##{
    ##    www_challenge("", "0");
    ##    exit;
    ##}
    ##
    ##if (!db_check_to())
    ##{
    ##    sl_send_reply("403", "Forbidden auth ID");
    ##    exit;
    ##}
    if (!save("location"))
        sl_reply_error();
    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 – this time, the 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.

```
if ($rU==NULL) {
    # request with no Username in RURI
    sl_send_reply("484","Address Incomplete");
    exit;
}
```

The preceding code discards requests without a complete URI.

```
# apply DB based aliases (uncomment to enable)
##alias_db_lookup("dbaliases");
```

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 proceeding. By default, this is not active.

```
# do lookup with method filtering
if (!lookup("location","m")) {
    switch ($retcode) {
        case -1:
        case -3:
            t_newtran();
            t_reply("404", "Not Found");
            exit;
        case -2:
            sl_send_reply("405", "Method Not Allowed");
            exit;
    }
}

# when routing via usrloc, log the missed calls also
setflag(2);
route(1);
```

The preceding code is very tricky, but very smart too. Now, we will search the location database to find an AOR (address of record). The parameter `m` in the command `lookup("location", "m")` enables method filtering. So, now only the contacts that support the method being requested from the contacts found will be permitted. Return codes for the `lookup()` function follow:

- 1 — contacts found and returned
- -1 — no contact found
- -2 — contacts found, but method not supported
- -3 — internal error during processing

The `lookup("location", "m")` 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). If the AOR is found, we will end up with `route(1);`.

```
route[1] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {
        t_on_branch("2");
        t_on_reply("2");
        t_on_failure("1");
    }
    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 TM (`tm.so`) module (`tm.so`) and is responsible for sending the requests and handling any resends and responses. If the request can't be sent to the destination successfully, then the `t_relay()` function will return an error condition. The `sl_replay_error()` function will send a reply back to the UA if a failure occurs.

```
# Presence route
/* uncomment the whole following route for enabling presence
   NOTE: do not forget to enable the call of this route from the main
   route */
##route[2]
##{
##    if (!t_newtran())
```

```
##      {
##          sl_reply_error();
##          exit;
##      };
##
##      if(is_method("PUBLISH"))
##      {
##          handle_publish();
##          t_release();
##      }
##      else
##      if( is_method("SUBSCRIBE"))
##      {
##          handle_subscribe();
##          t_release();
##      }
##
##      exit;
##}
##}
```

If you want to enable the presence agent, just uncomment the lines in the previous code block.

```
branch_route[2] {
    xlog("new branch at $ru\n");
}

onreply_route[2] {
    xlog("incoming reply\n");
}

failure_route[1] {
    if (t_was_cancelled()) {
        exit;
    }
    # uncomment the following lines if you want to block client
    # redirect based on 3xx replies.
    ##if (t_check_status("3[0-9][0-9]")) {
    ##t_reply("404", "Not found");
    ##    exit;
    ##}
    # uncomment the following lines if you want to redirect the failed
    # calls to a different new destination
    ##if (t_check_status("486|408")) {
    ##    sethostport("192.168.2.100:5060");
```

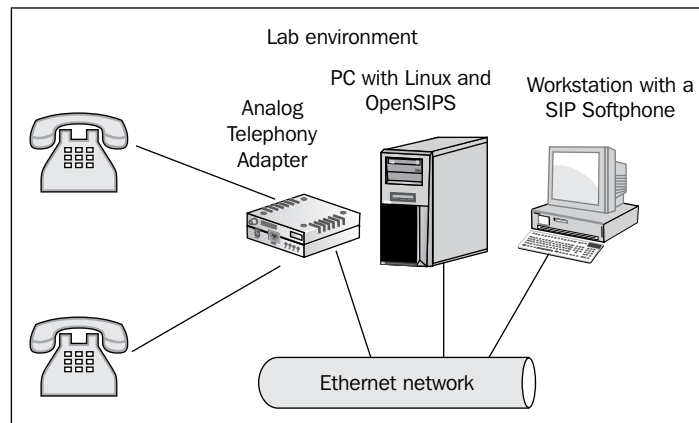


```
##      # do not set the missed call flag again
##      t_relay();
##}
}
```

These lines are just placeholders for `branch_route`, `onreply_route`, and `failure_route`. We will explain these functions in the related chapters.

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. Please adapt this lab to your needs.



- Start capturing packets using `ngrep`. If `ngrep` is not installed, then you can install it using:
`apt-get install ngrep`
- To capture the packets use:
`ngrep -p -q -W byline port=5060 >test.txt`
- 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 with 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.

- Check if the phone is registered using:
`opensipsctl ul show`
- Using the first UAC, dial 1001. The second UAC will ring.
- Verify that this capture does not exhibit the **407 - Proxy authentication required** message for the INVITE request and the **401- Unauthorized** message for the REGISTER requests. This proves that an authentication is not being asked.
- You can see the capture using the following command:
`more test.txt`

Common issues

I will now present a list of some common mistakes made when using this material. I could observe this in the classroom when teaching about OpenSIPS.

Daemon does not start

It 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 during the loading of modules.

This usually solves most problems.

Client unable to register

This is by far the most common. 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 address into the database too.
2. Plaintext or encrypted passwords? You can't mix plaintext with encrypted passwords. There are two places to check:

Use the following code for plaintext passwords in the `opensips.cfg` file:

```
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
```

In the `opensipsctlrc` file, be sure to leave a comment:

```
#STORE_PLAINTEXT=0
```

On the other hand, if you want to use encrypted passwords, use them in the `openser.cfg` file:

```
modparam("auth_db", "calculate_ha1", 0)
modparam("auth_db", "password_column", "ha1")
```

In the `opensipsctlrc` file, be sure to leave the comment:

```
STORE_PLAINTEXT=0
```

If you mix these things, you will end up not authenticating. The `opensipsctlrc` file regulates the creation of the users using `opensipsctl`. So, if you change this setting, the new settings will be valid only for the new users.

Too many connections

It is very common to receive this message in the beginning. After setting the script to multi-domain, very often, script writers forget to insert the domains in the domain table. This includes the IP address and the domain itself. If you try to send a call to a domain not defined in the table, the script will identify this domain as an external domain and try to send the call by using a DNS server. As the DNS will resolve the IP address of your server again, this request will get into a loop, sending the request again and again, back to the server, until the maximum number of forwards is reached. Check if the domain is in the domain table and if you have reloaded the domain table using `opensipsctl domain reload`.

Summary

In this chapter, you learned some of the statements for each one of the sections of the `opensips.cfg` file. This is the simplest configuration file.

In the following chapters, we will increase the functionality and the complexity of the script. This chapter served as a starting point to develop more advanced scripts. Although being simple, the scripts allow you to connect two phones and dial one from the other.

5

Adding Authentication with MySQL

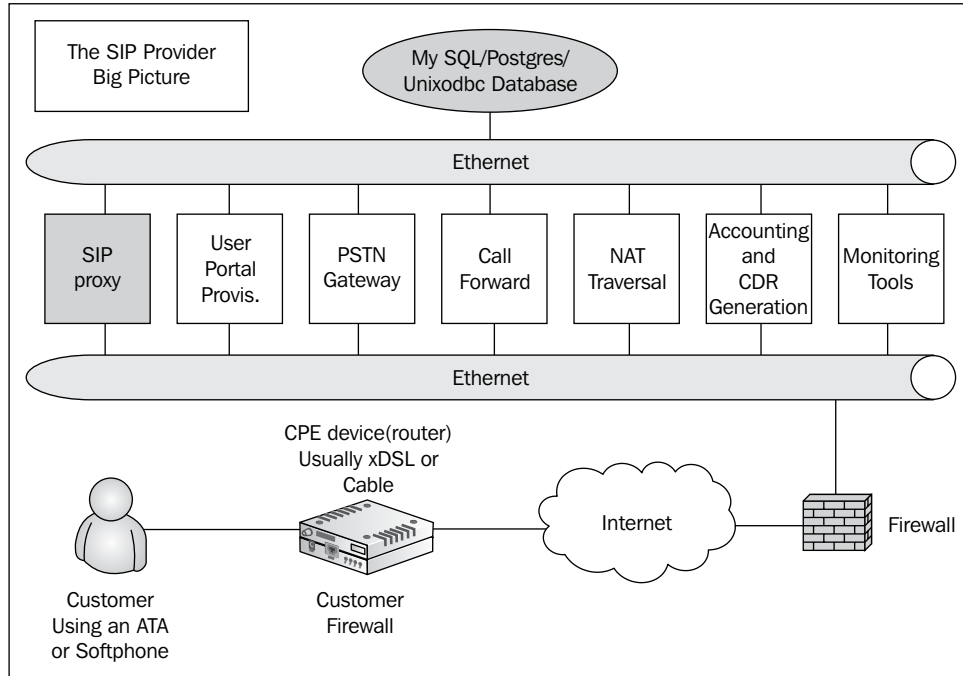
In this chapter, we will learn how to use several different database backends to authenticate SIP requests and save data such as location and alias tables. Primarily, we will do everything with MySQL. This chapter is divided into two parts. In the first part, we will learn how to implement authentication and in the second, 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 `opensipsctl` utility for basic operations such as adding users
- Change the `opensips.cfg` script to configure MySQL authentication
- Implement persistence for the subscriber table
- Implement persistence for the location tables
- Restart the server without losing the location records
- Deal correctly with inbound-to-inbound, inbound-to-outbound, outbound-to-inbound, and outbound-to-outbound sessions.

Where we are

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



The AUTH_DB module

The database-based authentication is performed by the AUTH_DB module. Other types of authentication such as those for radius and diameter can be performed using AUTH_AAA and AUTH_DIAMETER respectively. It 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 parameter for the AUTH_DB module:

Parameter	Default	Description
db_url	mysql://opensipsro: opensipsro@localhost/ opensips	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
password_column2	ha1b	Name of the column holding pre-calculated ha1 strings that were calculated by including the domain in the username
calculate_ha1	0 (the server assumes that ha1 strings are already calculated in the database)	Tells the server whether or not it should expect plaintext passwords in the database
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	rpidd	Specifies the credentials to be fetched from the database when the authentication is performed. The load credentials will be stored in AVPs

The AUTH_DB module exports two functions:

1. `www_authorize(realm, table)`: This function is used in the REGISTER authentication that occurs according to RFC2617.
2. `proxy_authorize(realm, table)`: The function verifies credentials according to RFC2617 for the non-REGISTER requests. If the credentials are verified successfully, they will be marked as authorized.

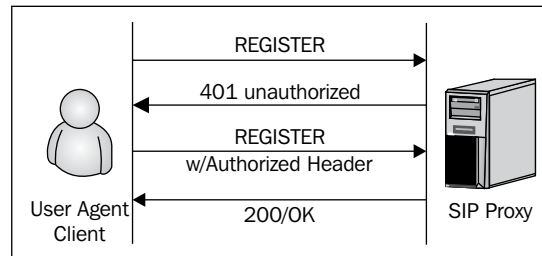
You have to use `www_authorize` when your server is the endpoint of the request. Use `proxy_authorize` when the request's final destination is not your server and you will forward the request ahead, working as a proxy.

The difference between the `www_authorize` and `proxy_authorize` parameters is that if the requests' endpoint is your server (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 `opensips.cfg` script. When OpenSIPS receives the REGISTER message, it checks for the existence of the Authorize header. If it is not found, 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

The registration process shown by the packets captured by `ngrep` follows:

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.

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: Opensips (1.5.1-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: Opensips (1.5.1-notls (i386/linux)).

Content-Length: 0.

Register sequence code snippet

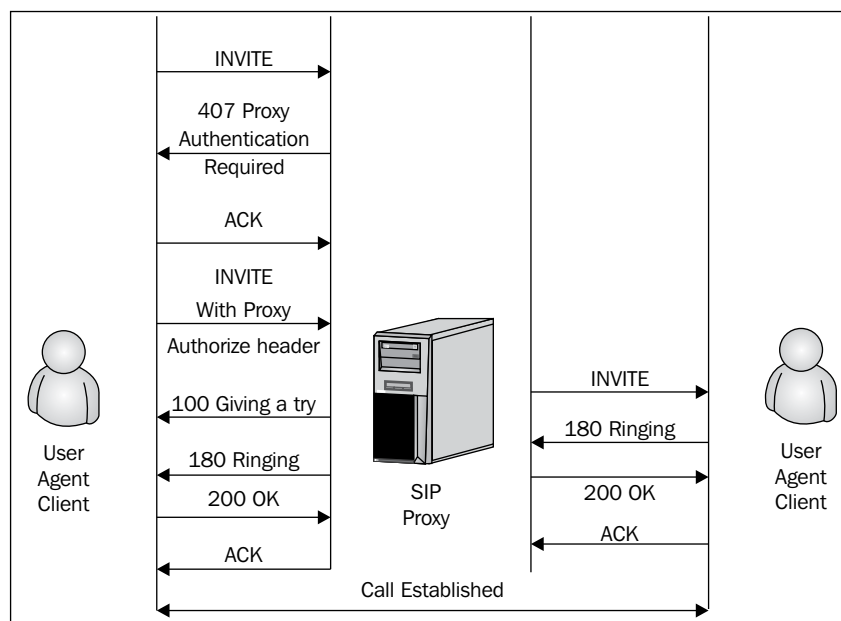
Let's look at how this sequence is coded in the `opensips.cfg` script:

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

In the above sequence, in the first pass, the REGISTER packet is not authenticated by the `www_authorize` function. Then the `www_challenge` instruction 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")` is invoked to save the AOR in the MySQL location table.

The INVITE authentication sequence

The INVITE authentication sequence of an ordinary call is shown in the next image. 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 (number used once)**. Once received by the UAC, this message is replied with a new INVITE. Now, the Authorize header field contains the digest calculated using the username, password, realm, and nonce calculated using the MD5 algorithm. If a match exists between the digest informed 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 previous image. 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: Opensips (1.5.1-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 following code, the SIP proxy will challenge the user for credentials on any request other than REGISTER. After authentication, we consume the credentials—in other words, we remove the Authorize Header from the packet for security reasons, to avoid sending encrypted material ahead.

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

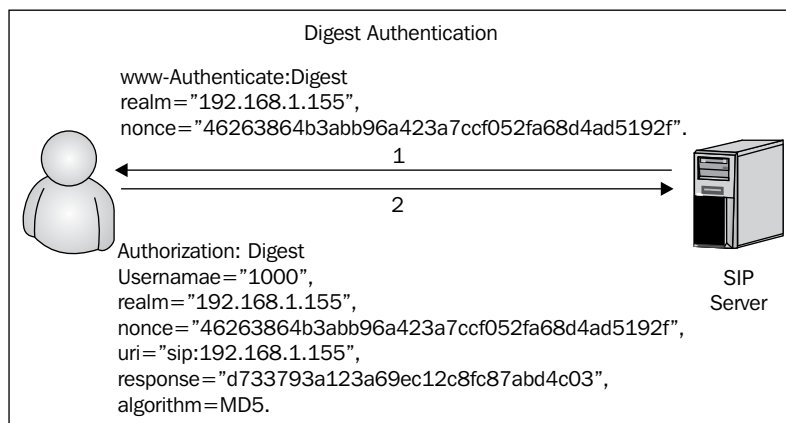
consume_credentials();
lookup("aliases");
if (!uri==myself) {
    append_hf("P-hint: outbound alias\r\n");
    route(1);
};

# 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.



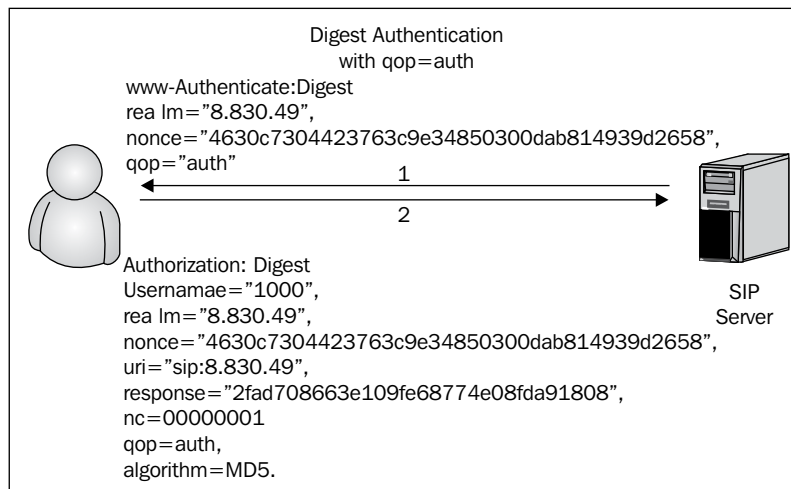
The digest scheme is a simple challenge-response mechanism. It challenges the UA using a nonce value. A valid response includes a checksum with all the parameters. Thus the password is never transmitted in plain text.

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 called **WWW-Authenticate**. This header contains a realm and a nonce.

The Authorization request header

The client is expected to try again, now by passing the Authorize header field. It contains the username, 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 referred 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 on 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 for 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) because it will help you to avoid "replay" attacks. However, some clients can be incompatible with `qop`. A detailed description of digest authentication can be found in RFC2617.

Plaintext or hashed passwords

You can store passwords in clear text or as hashes. Storing passwords as hashes is safer. However, SerMyAdmin is not capable of handling hashed passwords, so in this whole material we will use plain text passwords. Plaintext passwords are controlled by the `opensipsctlrc` and `opensips.cfg` files. The `opensipsctlrc` file controls the `opensipsctl` and `osipsconsole` utilities. It will store the password in plaintext in the `password` column if the `store_plaintext_pw` parameter is set to 1 or the HA1 value in the `ha1` column is set to 0 (`ha1` is a hash; the MD5 computation of `user | realm | password`). The `calculate_ha1` parameter is counterintuitive. When set to 1, it tells the server to expect plaintext passwords in the `password_column` and the HA1 strings will be calculated on the fly. On the other hand, if set to 0, it tells the server to expect HA1 strings directly and it won't need to calculate them as they are already calculated. There are two password columns — `password_column` and `password_column2` — the latest is used to store the HA1 hash including the domain. Some user agents include the domain in the user credentials.

So, in order to use plaintext passwords, set :

- `opensips.cfg`
`modparam("auth_db", "calculate_ha1", 1)`
`modparam("auth_db", "password_column", "password")`
- `opensipsctlrc`
`store_plaintext_pw=1`

In order to use hash passwords, set:

- `opensips.cfg`
`modparam("auth_db", "calculate_ha1", 0 )`
`modparam("auth_db", "password_column", "ha1")`
- `opensipsctlrc`
`store_plaintext_pw=0`

Installing MySQL support

To allow persistence, in other words, to keep the user credentials in a database where they are protected from power outages and reboots, OpenSIPS 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 `opensips-mysql` module compiled and installed.

In Chapter 3, *OpenSIPS Installation*, we have compiled OpenSIPS with MySQL support. Check the directory `oatu /lib/opensips/modules` for the `db_mysql.so` module.

Some additional tasks have to be performed before you can use OpenSIPS with MySQL.

Instructions:

In this lab activity, you will create the database using the `opensipsdbctl` command and change the `opensips.cfg` file to allow authentication of REGISTER and INVITE requests.

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

```
ls /lib/opensips/modules/db_mysql.so
```

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

Step 2: Edit the `opensipsctlrc` file before creating the database and change the attributes that follow:

```
SIP_DOMAIN=192.168.11.138
DBENGINE=MYSQL
DBHOST=localhost
DBNAME=opensips
DBRWUSER=opensips
DBRWPW="opensipsrw"
DBROUSER=opensipsro
DBROPW=opensipsro
ALIASES_TYPE="DB"
OSIPS_FIFO="/tmp/opensips_fifo"
```

Step 3: Create MySQL tables using the `opensipsdbctl` shell script. The syntax for this utility follows:

```
opensipsdbctl create <db name or db_path, optional>
```

Run the script with the next command:

```
opensipsdbctl create
```

The output of the command follows:

```
Opensips:~# opensipsdbctl create
```

MySQL password for root:

INFO: test server charset

INFO: creating database opensips ...

INFO: Core Opensips tables successfully created.

Install presence related tables? (y/n): y

INFO: creating presence tables into opensips ...

INFO: Presence tables successfully created.

Install tables for imc cpl siptrace domainpolicy carrieroute? (y/n): y

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* on both occasions. The script will ask for a domain (realm), enter your domain for the admin user.

The `opensipsdbctl` utility has several options such as `drop`, `reinit`, `backup`, `restore`, `copy`, and so on to install extra tables for presence and other modules. Check the help screen of the utility, issuing it without any parameters.

Step 4: Configure the OpenSIPS to use MySQL.

Step 5: Uncomment the related sections in `/etc/opensips/opensips.cfg`:

```
#loadmodule "db_mysql.so"
#loadmodule "auth.so"
#loadmodule "auth_db.so"

#modparam("usrloc", "db_mode", 2)
#modparam("usrloc", "db_url",
#      "mysql://opensips:opensipsrw@localhost/opensips")

#modparam("auth_db", "calculate_ha1", yes)
#modparam("auth_db", "password_column", "password")
#modparam("auth_db", "db_url",
#      "mysql://opensips:opensipsrw@localhost/opensips")
#modparam("auth_db", "load_credentials", "")

##if (!(method=="REGISTER") && from_uri==myself) /*no multidomain
version*/
##{
##      if (!proxy_authorize("", "subscriber")) {
##              proxy_challenge("", "0");
##              exit;
##      }
##      if (!db_check_from()) {
##              sl_send_reply("403","Forbidden auth ID");
##              exit;
##      }
```

```
##      }
##
##      consume_credentials();
##      # caller authenticated
##}

#authenticate the REGISTER requests
#(uncomment to enable auth)
##if (!www_authorize("", "subscriber"))
##{
##      www_challenge("", "0");
##      exit;
##}
##
##if (!db_check_to())
##{
##      sl_send_reply("403","Forbidden auth ID");
##      exit;
##}
```

Step 6: Restart OpenSIPS

Step 7: Create the users 1000 and 1001 in the database:

```
opensipsctl add 1000 1000
```

```
opensipsctl add 1001 1001
```

Step 8: Stop and restart the UACs

Step 9: Test the authentication

Analysis of the opensips.cfg file

Now the configuration is ready to authenticate REGISTER transactions. We can save the AOR in the location database implementing persistence. This allows us to restart the server without losing the AOR records and affecting the UACs. The file analysis will show you the modules loaded and their respective parameters and the main sections of the code related to the chapter.

To make the authentication work, it is necessary to load the following modules:

- loadmodule "db_mysql.so"
- loadmodule "auth.so"
- loadmodule "auth_db.so"

The MySQL support is added easily by including the `db_mysql.so` in the list of loaded modules. MySQL should be loaded before the other modules. The authentication capability is provided by the `auth.so` and `auth_db.so` modules. These modules are required to enable the authentication functionality.

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

The parameter `calculate_ha1` tells the `AUTH_DB` module to use plaintext passwords. We will use this setting for compatibility with `SerMyAdmin` and `opensips-cp`.

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

Register requests

In the next code snippet, we will check the authentication for the method `REGISTER`.

```
#authenticate the REGISTER requests (uncomment to enable auth)
if (!www_authorize("", "subscriber"))
{
    www_challenge("", "0");
    exit;
}
if (!db_check_to())
{
    sl_send_reply("403","Forbidden auth ID");
    exit;
}
```

If the method is `REGISTER` and the credentials are correct, the `www_authorize` function 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. The realm is usually the domain name or host name. The second parameter tells OpenSIPS which MySQL table to look for.

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

If the packet does not have an `Authorize` header field we will send a message **401 unauthorized** to the UAC. 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.

Non-Register requests

All SIP requests except for CANCEL and ACK are usually authenticated. For requests traversing the SIP proxy, we use the function `proxy_authorize()` instead of `www_authorize()`.

```
if (!(method=="REGISTER") && from_uri==myself)/*no multidomain
version*/
{
    if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "0");
        exit;
    }
    if (!db_check_from()) {
        sl_send_reply("403","Forbidden auth ID");
        exit;
    }

    consume_credentials();
    # caller authenticated
}
```

The sequence for the INVITE request is very similar to that for the REGISTER request. If the request does not have a valid Authorize header, the proxy will send a challenge to the UAC within the message **407 Proxy Authentication Required**. The UAC will then retransmit the request, now containing the `Authorize:` header. Some commands deserve to be mentioned as follows:

```
consume_credentials()
```

We don't want to take risks by sending the digest credentials to servers ahead. Then we use the `consume_credentials()` function 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 don't check the credentials, it could be considered an open relay. The arguments are similar to `www_authorize`.

```
db_check_from()
```

When operating a SIP proxy, you should guarantee that a valid account won't be used by non-authenticated users. The `db_check_to()` and `db_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 (the `Authorize` header field) and has its own password. In the current example, the function checks that the SIP user and the auth user are the same. This is to prevent a user from using the credentials of another user. These functions are enabled by the URI module.

The opensipsctl shell script

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

- Start, stop, and restart OpenSIPS.
- 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 OpenSIPS

We will learn several of its options in the following chapters. The output of the `opensipsctl help` command follows:

```
opensips:~# opensipsctl -help
/sbin/opensipsctl $Revision: 4448 $

Existing commands:

-- command 'start|stop|restart'

restart ..... restart OpenSIPS
start ..... start OpenSIPS
stop ..... stop OpenSIPS

-- 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 'lcr' - manage least cost routes (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 level3 1.2.3.4 5080 sip tcp 1 *
*           lcr addroute +1 '1 1 1' *

```

The resource file—opensipsctlrc

This script is found at `/etc/opensips`. It is parsed by the `opensipsctl` utility to configure the database authentication and some communication parameters. Usually it uses the FIFO mechanism to send commands to the OpenSIPS daemon.



For security reasons, it is important to change the default username and password used for database access.

The opensipsctlrc file

To show the file, issue the following command:

```
vi opensipsctlrc
# $Id: opensipsctlrc 4331 2008-06-06 14:36:01Z anca_vamanu $
#
# The OpenSIPS configuration file for the control tools.
#
# Here you can set variables used in the opensipsctl and opensipsdbctl
# setup
# scripts. Per default all variables here are commented out, the
# control tools
# will use their internal default values.
## your SIP domain
SIP_DOMAIN=192.168.11.138
## chrooted directory
# $CHROOT_DIR="/path/to/chrooted/directory"
## database type: MYSQL, PGSQL, ORACLE, DB_BERKELEY, or DBTEXT, by
# default none is loaded
# If you want to setup a database with opensipsdbctl, you must at
# least specify
# this parameter.
DBENGINE=MYSQL
## database host
DBHOST=localhost
## database name (for ORACLE this is TNS name)
DBNAME=opensips
# database path used by dbtext or db_berkeley
# DB_PATH="/usr/local/etc/opensips/dbtext"
## database read/write user
DBRWUSER=opensips
```

```
## password for database read/write user
DBRWPW="opensiprsw"

## database read only user
DBROUSER="opensipsro"

## password for database read only user
DBROPW="opensipsro"

## database super user (for ORACLE this is 'scheme-creator' user)
DBROOTUSER="root"

# user name column
# USERCOL="username"

# SQL definitions
# If you change this definitions here, then you must change them
# in db/schema/entities.xml too.
# FIXME

# FOREVER="2020-05-28 21:32:15"
# DEFAULT_ALIASES_EXPIRES=$FOREVER
# DEFAULT_Q="1.0"
# DEFAULT_CALLID="Default-Call-ID"
# DEFAULT_CSEQ="13"
# DEFAULT_LOCATION_EXPIRES=$FOREVER

# Program to calculate a message-digest fingerprint
# MD5="md5sum"

# awk tool
# AWK="awk"

# grep tool
# GREP="grep"

# sed tool
# SED="sed"

# Describe what additional tables to install. Valid values for the
variables
# below are yes/no/ask. With ask (default) it will interactively ask
the user
# for an answer, while yes/no allow for automated, unassisted
installs.
#

# If to install tables for the modules in the EXTRA_MODULES variable.
# INSTALL_EXTRA_TABLES=ask

# If to install presence related tables.
# INSTALL_PRESENCE_TABLES=ask

# Define what module tables should be installed.
```



```
# If you use the postgres database and want to change the installed
tables, then you
# must also adjust the STANDARD_TABLES or EXTRA_TABLES variable
accordingly in the
# opensipsdbctl.base script.

# opensips standard modules
# STANDARD_MODULES="standard acc lcr domain group permissions
registrars usrloc msilo
#          alias_db uri speeddial avpops auth_db pdt dialog dispatcher
#          dialplan"

# opensips extra modules
# EXTRA_MODULES="imc cpl siptrace domainpolicy carrieroute
userblacklist"

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

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

## path to FIFO file
OSIPS_FIFO="/tmp/opensips_fifo"

## MI_CONNECTOR control engine: FIFO, UNIXSOCK, UDP, XMLRPC
# MI_CONNECTOR=FIFO:/tmp/opensips_fifo
# MI_CONNECTOR=UNIXSOCK:/tmp/opensips.sock
# MI_CONNECTOR=UDP:192.168.2.133:8000
# MI_CONNECTOR=XMLRPC:192.168.2.133:8000

## 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

## OPENSIPS START Options
## PID file path - default is: /var/run/opensips.pid
# PID_FILE=/var/run/opensips.pid
```

```
## Extra start options - default is: not set
# example: start opensips with 64MB share memory: STARTOPTIONS="-m 64"
# STARTOPTIONS=
```

Using OpenSIPS with authentication

Now, let's implement the authentication in a practical way using the following steps:

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

Step 2: Restart OpenSIPS with `/etc/init.d/opensips restart`

Step 3: Configure `opensipsctlrc` using the default parameters used with `opensipsctl`.

```
SIP_DOMAIN=your-sip-domain
DBENGINE=MYSQL
DBHOST=localhost
DBNAME=opensips
DBRWUSER=opensips
DBROUSER=opensipsro
DBROPW=opensipsro
DBROOTUSER="root"
ALIASES_TYPE="DB"
CTLENGINE="FIFO"
OSIPS_FIFO="/tmp/opensips_fifo"
VERIFY_ACL=1
ACL_GROUPS="local ld int voicemail free-pstn"
VERBOSE=1
#STORE_PLAINTEXT_PW=0
```

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

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

 When asked for the password, use `opensipsrw`.

You can remove users using the `opensipsctl rm` command and change a password using `opensipsctl passwd` command.

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 username and password

Step 7: Verify that the phones are registered using `#opensipsctl ul show`

Step 8: You can verify which users are online using `#opensipsctl online`

Step 9: You can ping a client using `#opensipsctl ping 1000`

Step 10: Verify the authentication messages using the `ngrep` utility

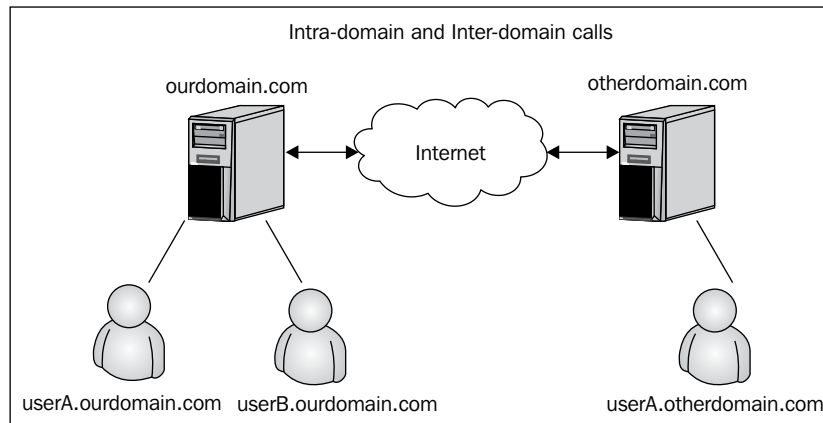
Step 11: Make a call from one phone to another

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

```
#pg register.pkt
```

Enhancing the script

Besides authentication, it is important to check the source and the destination of your message. In this section, we are going to show you how to handle calls coming in and going out to different domains.



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

- Intra-domain
- Outbound inter-domain
- Inbound inter-domain
- Outbound to outbound

The script, by default, uses the statement, `from_uri==myself` to identify if the `From` header of the request belongs to one of the domains served by this computer. The domains served by this computer are defined by the core parameter `alias=`. There are two ways to populate the domains served. The first is by creating one line of `alias` for each domain and another by allowing the automated discovery of the alias using reverse DNS. The first one is, in my opinion, the easiest and the safest. Look at the following instruction:

```
if (!is_method("REGISTER")) && from_uri==myself)
{
    if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "0");
        exit;
    }
}
```

If the method is different from `register` and the `uri` presented in the `From:` header belongs to one of the domains being served, authenticate the request. This means that if a request comes from an unknown domain, it will bypass the authentication. Later in the script you will find (I removed the comments to make it clearer):

```
if (!uri==myself)
{
    route(1);
}
```

The above block of code routes any request with a request URI in an unknown domain. With these two statements, we can conclude that the default script is an open-relay script. It allows requests from an unknown domain to be relayed to another unknown domain and this is not good. It is fairly easy to stop relaying unknown requests by replacing the previous script with the following script:

```
if (!uri==myself)
{
    if(from_uri==myself) {
        route(1);
    } else
        sl_send_reply("403", "Not here");
}
```

So, let's analyze how the default script routes the requests depending on the `From` header and the request URI's domain:

- **Intra-domain** authentication is required and the destination will be evaluated, usually using the user location table:

```
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;
    }
}
route(1);
```

- **Outbound inter-domain** is being sent to `route(1)` to be routed using the DNS server:

```
if (!uri==myself)
{
    route(1);
}
```

- **Inbound inter-domain:** It is being routed to DNS or user location table depending on the request URI. It cannot be authenticated as the user is external, and if allowed, can expose you to VoIP SPAM.
- **Outbound-to-outbound:** It is being routed. You can avoid it as shown before.

The conclusion is that, by using the `uri==myself` and `from_uri==myself` statements and the core value `alias` correctly, you can control domains from and to which you want to relay your requests.

Managing multiple domains

Until now, we have authenticated the requests using only the username part of the URI. To manage multiple domains, it is *not* required anymore to change the behavior of some functions to consider the domain part of the URI by using the `use_domain` parameter. In the Version 1.6, the `use_domain` parameter was removed and all the functions by default considered the whole `uri`. To use the old behavior, you have

to append a `d` flag after the function that requires the domain name. In order to allow multi-domain operations, you will need to load the DOMAIN module and to replace some functions inside your script. The domains will have to be inserted in the Domains table of our database before being used. The following modules and parameters are required:

```
loadmodule "domain.so"
# ----- domain params -----
modparam("domain", "db_url", "mysql://opensips:opensipsrw@localhost/opensips")
modparam("domain", "db_mode", 1)    # Use caching
```

We used to verify 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, namely `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()` that 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 made by users of this material is to forget to insert the domains in the MySQL database before registering the phones!



To insert a domain into the database, you can use the command:

```
opensipsctl domain add domain
```

After inserting a new domain, it is necessary to `reload` the domain database using:

```
opensipsctl domain reload
```

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 the following command:

```
#opensipsctl alias_db add flavio@opensips.org sip:1000@opensips.org
```

The output appears as:

```
database engine 'MYSQL' loaded
Control engine 'FIFO' loaded
MySql password for user 'opensips@localhost':
```

There are two implementations of the alias functionality. The first one is the alias database. In this modality, you use the `lookup("aliases")` function. This function goes to the aliases table, checks the existence of the alias, and replaces the R-URI with the canonical form of the alias. Recently, a new module called `ALIAS_DB` has been created as an alternative for user aliases via `usrloc`. The main feature is that it does not store all adjacent data as for user location and always uses the database for search, it does not use memory caching and thus is much easier to provision. Aliases are often used for DID redirection.

Before you can use an alias, you will need to load the modules and the related parameters.

```
loadmodule "alias_db.so"

# ----- alias_db params -----
modparam("alias_db", "db_url",
    "mysql://opensips:opensipsrw@localhost/opensips")
```

To search the aliases and replace the RURI with the results, use:

```
alias_db_lookup("dbaliases");
```

The `alias_db_lookup("dbaliases")` function checks the `dbaliases` table in the database and if a register is found, it translates it to the canonical address (the one in subscriber's table).

Handling CANCEL request and retransmissions

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

Some parts of the script and some comments were deleted to reduce the space required for printing. Some of the changes are highlighted:

```
##### Modules Section #####
#set module path
mpath="/usr/local/lib/opensips/modules/"

/* uncomment next line for MySQL DB support */
loadmodule "db_mysql.so"
loadmodule "signaling.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrars.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
loadmodule "uri.so"
loadmodule "xlog.so"
loadmodule "acc.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "alias_db.so"
loadmodule "domain.so"
#loadmodule "presence.so"
#loadmodule "presence_xml.so"

# ----- setting module-specific parameters -----
# ----- mi_fifo params -----
modparam("mi_fifo", "fifo_name", "/tmp/opensips_fifo")
# ----- rr params -----
```



```
# add value to ;lr param to cope with most of the UAs
modparam("rr", "enable_full_lr", 1)
# do not append from tag to the RR (no need for this script)
modparam("rr", "append_fromtag", 0)

# ----- registrar params -----

/* uncomment the next line not to allow more than 10 contacts per AOR
*/
modparam("registrar", "max_contacts", 10)

# ----- usrloc params -----
modparam("usrloc", "db_mode", 0)
/* uncomment the following lines if you want to enable DB persistency
for location entries */
modparam("usrloc", "db_mode", 2)
modparam("usrloc", "db_url",
        "mysql://opensips:opensipsrw@localhost/opensips")

# ----- uri params -----
/* by default we disable the DB support in the module as we do not
need it in this configuration */
modparam("uri", "use_uri_table", 0)

# ----- acc params -----
/* what sepcial events should be accounted ? */
modparam("acc", "early_media", 1)
modparam("acc", "report_ack", 1)
modparam("acc", "report_cancels", 1)
/* by default ww do not adjust the direct of the sequential requests.
if you enable this parameter, be sure the enable "append_fromtag"
in "rr" module */
modparam("acc", "detect_direction", 0)
/* account triggers (flags) */
modparam("acc", "failed_transaction_flag", 3)
modparam("acc", "log_flag", 1)
modparam("acc", "log_missed_flag", 2)
/* uncomment the following lines to enable DB accounting also */
modparam("acc", "db_flag", 1)
modparam("acc", "db_missed_flag", 2)

# ----- auth_db params -----
/* uncomment the following lines if you want to
enable the DB based authentication */
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("auth_db", "db_url",
        "mysql://opensips:opensipsrw@localhost/opensips")
modparam("auth_db", "load_credentials", "")
```

```

# ----- alias_db params -----
/* uncomment the following lines if you want to
    enable the DB based aliases */
modparam("alias_db", "db_url",
    "mysql://opensips:opensipsrw@localhost/opensips")

# ----- domain params -----
/* uncomment the following lines to
    enable multi-domain detection support */
modparam("domain", "db_url",
    "mysql://opensips:opensipsrw@localhost/opensips")
modparam("domain", "db_mode", 1)    # Use caching

# ----- presence params -----
/* uncomment the following lines if you want to enable presence */
#modparam("presence|presence_xml", "db_url",
#    "mysql://opensips:opensipsrw@localhost/opensips")
#modparam("presence_xml", "force_active", 1)
#modparam("presence", "server_address", "sip:192.168.1.2:5060")

##### Routing Logic #####
route{
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483", "Too Many Hops");
        exit;
    }
    if (has_totag()) {
        # sequential request withing a dialog should
        # take the path determined by record-routing
        if (loose_route()) {
            if (is_method("BYE")) {
                setflag(1);
                setflag(3);
            } else if (is_method("INVITE")) {
                record_route();
            }
            route(1);
        } else {
            ##if (is_method("SUBSCRIBE") &&
                $rd == "your.server.ip.address") {
            ##    # in-dialog subscribe requests
            ##    route(2);
            ##    exit;
            ##}
            if ( is_method("ACK") ) {
                if ( t_check_trans() ) {

```

```

        t_relay();
        exit;
    } else {
        exit;
    }
}
sl_send_reply("404","Not here");
}
exit;
}

#initial requests
# CANCEL processing
if (is_method("CANCEL"))
{
    if (t_check_trans())
        t_relay();
    exit;
}
t_check_trans();
if (!is_method("REGISTER")) && is_from_local() {
    if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "0");
        exit;
    }
    if (!db_check_from()) {
        sl_send_reply("403","Forbidden auth ID");
        exit;
    }
    consume_credentials();
    # caller authenticated
}

# preloaded route checking
if (loose_route()) {
    xlog("L_ERR",
        "Attempt to route with preloaded Route's [$fu/$tu/$ru/
$ci]");

    if (!is_method("ACK"))
        sl_send_reply("403","Preload Route denied");
    exit;
}

# record routing
if (!is_method("REGISTER|MESSAGE"))
```

```

        record_route();
# account only INVITEs
if (is_method("INVITE")) {
    setflag(1); # do accounting
}
if (!is_uri_host_local())
{
    if(is_from_local()) {
        route(1);
    } else {
        sl_send_reply("403","Not here");
    }
}
##if( is_method("PUBLISH|SUBSCRIBE"))
##    route(2);
if (is_method("PUBLISH"))
{
    sl_send_reply("503", "Service Unavailable");
    exit;
}
if (is_method("REGISTER"))
{
    if (!www_authorize("", "subscriber"))
    {
        www_challenge("", "0");
        exit;
    }
    if (!db_check_to())
    {
        sl_send_reply("403","Forbidden auth ID");
        exit;
    }
    if (!save("location"))
        sl_reply_error();
    exit;
}
if ($rU==NULL) {
    # request with no Username in RURI
    sl_send_reply("484","Address Incomplete");
    exit;
}

```

```
# apply DB based aliases (uncomment to enable)
alias_db_lookup("dbaliases");
if (!lookup("location","m")) {
    switch ($retcode) {
        case -1:
        case -3:
            t_newtran();
            t_reply("404", "Not Found");
            exit;
        case -2:
            sl_send_reply("405",
                "Method Not Allowed");
            exit;
    }
}

# when routing via usrloc, log the
# missed calls also setflag(2);
route(1);
}

route[1] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {
        t_on_branch("2");
        t_on_reply("2");
        t_on_failure("1");
    }
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

branch_route[2] {
    xlog("new branch at $ru\n");
}

onreply_route[2] {
    xlog("incoming reply\n");
}

failure_route[1] {
    if (t_was_cancelled()) {
        exit;
    }
}
}
```

Lab—multi-domain support

In this LAB, we are going to use the new multi-domain script. Please note that is mandatory to include the domains in the domain table when you use this script.

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

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

To add a domain, use the `opensipsctl` utility.

```
opensipsctl domain add your-ip-address
```

```
opensipsctl domain add your-domain
```

Repeat the process for every domain.

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

Lab—using aliases

Aliases are often used to give a name or number to a subscriber. **Direct Inward Dial (DID)** redirection is another frequent use of aliases.

Step1: Add an alias to the subscriber 1000.

```
#opensipsctl alias_db add john@youripordomain 1000@youripordomain
```

The output for the above command is:

```
database engine 'MYSQL' loaded
```

```
Control engine 'FIFO' loaded
```

```
MySql password for user 'opensips@localhost':
```

[ Use opensipsrw as the password]

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

The call will now be completed, John will be translated to the canonical name (1000, and the subscriber will be registered) before processing in the user location table.

Summary

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

6

Graphical User Interfaces for OpenSIPS

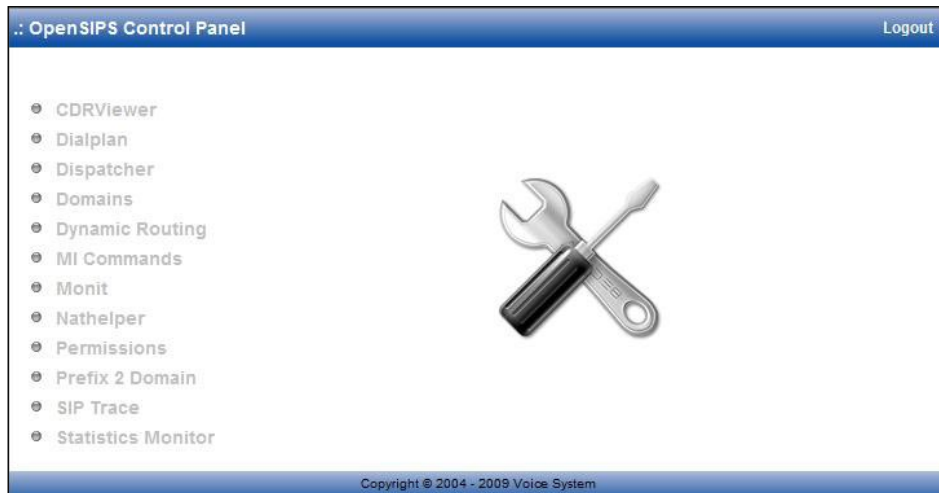
In the last chapter, we implemented authentication using a MySQL database. Now we will need a tool to help users and administrators. Obviously, this tool has to be easier than **opensipsctl**. It is very hard to manage thousands of users manually, so a user-provisioning tool becomes very important in our process. In this chapter, we are going to see two of these tools – **SerMyAdmin** and **OpenSIPS Control Panel (OpenSIPS-CP)**. SerMyAdmin is more focused on the administration of the users, while OpenSIPS-CP is excellent for monitoring and provisioning parameters to the system. Instead of explaining each item in this chapter, we will show the tool working in the chapters ahead when you need to edit the OpenSIPS database.

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
- Install and configure OpenSIPS Control Panel
- Use opensips-cp for daily tasks

OpenSIPS Control Panel

This tool, also known as **opensips-cp**, is the new graphical user interface for the SIP proxy. It was designed to be the primary tool to provision parameters for the OpenSIPS modules in the database.



The screenshot shows the tools available for OpenSIPS Control Panel. This tool is focused on the provisioning of system parameters. As it is usual for the VoIP providers to develop their own interface with the end users, the tools to manage end users are not available. Developed using PHP, opensips-cp currently features the following modules:

- CDRviewer
- dialplan
- domains
- dispatcher
- drouting
- load balancer
- Mi
- Monit
- nathelper
- pdt
- Permissions
- siptrace
- smonitor

Installation of opensips-cp

The step-by-step instructions to install opensips-cp in a server with Debian 5.0 are as follows:



Installation instructions change very often. For updates, check the opensips-cp project's website at <http://opensips-cp.sourceforge.net/>.

Step 1: Install Apache and PHP as follows:

```
apt-get install apache2 php5
```

Step 2: Install **php5-mysql** and **php5-xmlrpc** packages, and set the right parameters in the `php.ini` file. (It has been done for you to save time.)

```
apt-get install php5-mysql php5-xmlrpc php-pear
vi /etc/php5/apache2/php.ini
```

Please verify that you have set the `short_open_tag = On` ; option in your `php.ini` file.

Step 3: Download opensips-cp and untar the file.

Download opensips-cp from <http://opensips-cp.sourceforge.net/index.php?req=download> (or copy from DVD) and copy the `opensip-cp_3.0.tgz` file to `/var/www` as follows:

```
cd /var/www
wget http://sourceforge.net/projects/opensips-cp/files/opensips-cp/3.0/opensips-cp_3.0.tgz/download
tar -xzf opensips-cp_3.0.tgz
chown www-data:www-data opensips-cp -R
```

Step 4: Install MDB2.

```
pear install MDB2
pear install MDB2#mysql
pear install log
```

Step 5: Configure Apache for OpenSIPS Control Panel. Edit the `apache2.conf` file

```
vi /etc/apache2/apache2.conf
```

After doing so, include the following line below the last line:

```
Alias /cp "/var/www/opensips-cp/web"
```

Also, change the owner of the log file:

```
chown www-data:www-data /var/www/opensips-cp/config/access.log
```

Step 6: Install the cdr table schema:

```
cd /var/www/opensips-cp/web/tools/cdrviewer
mysql -D opensips -p < cdrs.sql
mysql -u root -p
mysql> use opensips
mysql -D opensips -p < opensips_cdrs_1_6.mysql
```

Step 7: Edit the `cron_job/generate-cdrs.sh` file and change the MySQL connection data (hostname, username, password, and database) as follows:

```
cd /var/www/opensips-cp/cron-job
vi generate_cdrs.sh
```

Step 8: Edit the `/etc/crontab` file and add the following line for a three-minute interval:

```
vi /etc/crontab
*/3 * * * * root /var/www/opensips-cp/cron_job/generate-cdrs.sh
```

Step 9: For the `smonitor` module, you must add two tables to the OpenSIPS database:

```
cd /var/www/opensips-cp/web/tools/smonitor
mysql -p opensips <tables.sql
```

Step 10: Add a cron job that collects data from the OpenSIPS machine(s). Here is a cron job that collects data at a one-minute interval. (This interval is not arbitrary. It must be set to one minute by design.)

```
vi /etc/crontab
* * * * * root php /var/www/opensips-cp/cron_job/get_opensips_stats.php > /dev/null
```

The cron jobs do not need to run as root. You might want to change the user.

Step 11: Restart OpenSIPS and Apache.

Installing Monit

Monit is a system-monitoring utility that allows an admin to easily monitor files, processes, directories, or devices on your system. It can also be used for automatic maintenance/repairs by executing particular commands when errors arise. Providing instructions on how to install Monit is beyond the scope of this book, but we have provided some here in order to avoid errors when you select the Monit tool in the OpenSIPS-CP.

The step-by-step instructions are as follows:

Step 1: To install Monit on your server, simply use `apt-get`:

```
apt-get install monit
```

Step 2: Once installed, you'll find the main configuration file.

```
vi /etc/monit/monitrc

set daemon 120
set logfile syslog facility log_daemon
set alert root@localhost
set httpd port 2812 and
use address yourdomain.com
allow localhost          # allow localhost to connect to the server and
allow youripaddress      # allow 192.168.1.2 to connect to the server,
                        # You can give only one per entry
allow admin:monit        # user name and password for authentication.
check process opensips with pidfile /var/run/opensips.pid
#Below is actions taken by monit when service got stuck.
start program = "/etc/init.d/opensips start"
stop program  = "/etc/init.d/opensips stop"

# Admin will notify by mail if below of the condition satisfied.
if cpu is greater than 70% for 2 cycles then alert
if cpu > 90% for 5 cycles then restart
```

Step 3: After modifying the configuration file, you should check for the syntax to make sure that everything is correct. To do this, run:

```
# monit -t
```

Step 4: Edit the `/etc/default/monit` file and change the parameters as follows:

```
# You must set this variable to for 1 monit to start
startup=1

# To change the intervals which monit should run uncomment
# and change this variable.
# CHECK_INTERVALS=180
```

Configuring OpenSIPS Control Panel

Step 1: Configure the database access parameters. Edit the `db.inc.php` file, which is valid for all modules. You may change the database parameters for a single module inside the module configuration section.

For example:

```
cd /var/www/opensips-cp/config
vi db.inc.php

//database host
$config->db_host = "localhost";

//database port - leave empty for default
$config->db_port = "";

//database connection user
$config->db_user = "root";

//database connection password
$config->db_pass = "opensips";

//database name
$config->db_name = "opensips";

if ($config->db_port != "")$config->db_host=$config->db_host":"$config->db_port;
```

Step 2: Configure the FIFO access in the `boxes.global.inc.php` file:

```
cd /var/www/opensips-cp/config/
vi boxes.global.inc.php

$box_id=0;

// mi host:port pair || fifo_file
$boxes[$box_id]['mi']['conn']="/tmp/opensips_fifo";

// monit host:port
$boxes[$box_id]['monit']['conn']="127.0.0.1:2812";
$boxes[$box_id]['monit']['user']="admin";
$boxes[$box_id]['monit']['pass']="monit";
$boxes[$box_id]['monit']['has_ssl']=0;

// description (appears in mi , monit )
$boxes[$box_id]['desc']="Primary SIP server";

$boxes[$box_id]['assoc_id']=1;

// enable local smonitor charts on this box : 0=disabled 1=enabled
// (cron)
$boxes[$box_id]['smonitor']['charts']=1;
```

Step 3: Access http://server_ip_address/cp and check each module for the correct functionality.

SerMyAdmin

SerMyAdmin was originally created to be an alternative for provisioning user and system information to the database. It is licensed according to GPL-2 and developed in **Grails (Groovy on Rails)**. It can be downloaded from <http://sourceforge.net/projects/sermyadmin>.



The utility SerMyAdmin has the following modules:

- Users
- Groups
- AVPs
- Alias
- Permissions
- Drouting
- LCR
- Dialplan
- Load Balance
- Dispatcher
- Nathelper

- PDT
- ACC
- CDRS
- Speed Dial

Some more interesting features of the interface are:

- User auto-registration with e-mail confirmation
- Captcha to prevent spam in the auto-registration process
- A user portal to allow end users to change preferences, speed dial, and password
- It allows enabling and disabling modules
- Support for internationalization (i18N)
- An installation utility

The utilities in OpenSIPS-CP that are not found in SerMyAdmin are **siptrace**, **smonitor**, and **monit**.

Lab—installing SerMyAdmin

We started the SerMyAdmin project in 2007 as a GUI for OpenSER because none was available. We migrated the tool for OpenSIPS recently; the idea is not to compete with OpenSIPS-CP, but to provide an easy interface to edit OpenSIPS tables and users. The new version is capable of automatically registering users with e-mail confirmation and providing user parameters. There is a plan to rename the tool to **SipMyAdmin** in the near future for coherency. Many customers don't really see a product if you don't have a graphical user interface. The administration costs to operate OpenSIPS in the command line are high because you need a specialized professional just to create users, groups, and so on. I think SerMyAdmin is a good start for a user and an administration portal.

SerMyAdmin uses the Grails framework, which is an excellent platform to develop web applications connected to a `jdbc` database. It is possible to include a new module in a matter of hours. Grails has a nice learning curve, so within a few days you can start changing or enhancing SerMyAdmin to fulfill your needs.

On the other hand, Grails needs an application server. You can choose from many different application servers, such as Tomcat or Glassfish. In this book, we will use Apache's Tomcat because it's free and easy to install. As we use some Java 1.6 features, we'll need Sun's Java JDK; rather than the free alternative **GNU Compiler for Java (GCJ)**.

Step 1: The first step is to update your sources list to use the contrib repository and the non-free packages. Your `/etc/apt/sources.list` should look like:

```
vi /etc/apt/sources.list
```

```
deb http://ftp.us.debian.org/debian/ lenny main contrib non-free
deb-src http://ftp.us.debian.org/debian/ lenny main contrib non-free
deb http://security.debian.org/ lenny/updates main contrib non-free
deb-src http://security.debian.org/ lenny/updates main contrib non-free
```



We've added only the contrib and non-free keywords after our repository definitions.

Step 2: Update the package listing using the following command:

```
apt-get update
```

Step 3: Download and uncompress `sermyadmin-install-2.x.tar.gz` from the website `www.sermyadmin.org`.



The URLs for downloadable files change very often. Please check the website first and change x to the latest version of the software.

```
cd /usr/src
```

```
wget http://www.sermyadmin.org/pub/sermyadmin-install-2.x.tar.gz
```

```
tar -xzf sermyadmin-install-2.x.tar.gz
```

```
cd sermyadmin-install
```

Step 4: Run the installation shell script and answer yes to the questions:

```
./install.sh
```

Step 5: To make sure everything is running fine, reboot the server and try to open the URL `http://localhost:8080` in your browser. If everything is okay, you'll be greeted with Tomcat's start page.

Step 6 (optional): Configure Debian's **message transfer agent (MTA)** in case you want your own server to send e-mails, but this is not the easiest setup. The utility SerMyAdmin can be configured to use any e-mail service; all you have to do is set the parameters.

```
apt-get install exim4
```

```
dpkg-reconfigure exim4-config
```


You will be greeted with a dialog-based configuration menu. On this menu, it's important to pay attention to two options:

- **General type of mail configuration:** This should be set to **Internet Site**, so that we can send and receive mails directly using SMTP
- **Domains to relay mail for:** This should be set to the domain which you want the e-mails from SerMyAdmin to appear to come from

SerMyAdmin configuration

Step 1: Declare the datasource for SerMyAdmin to connect to OpenSIPS's database. You can do this in an XML file found at `/usr/local/tomcat6/conf/context.xml`. The file should look like the following:

```
<?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/opensips_MySQL"
    type="javax.sql.DataSource"
    url="jdbc:mysql://localhost:3306/opensips"
    username="opensips" password="opensipsrw"/>
</Context>
```

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

Step 2: Customize the `/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin/WEB-INF/spring/resource.xml` file, which contains the parameters that dictate which e-mail server is used to send mails and from whom these e-mails should appear to come from. The following is a sample 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="smtpAuthenticator" class="SmtAuthenticator">
    <constructor-arg value="email@sermyadmin.org" />
    <constructor-arg value="password" />
  </bean>
  <bean id="mailSession" class="javax.mail.Session" factory-
    method="getInstance">
    <constructor-arg>
      <props>
```

```

        <prop key="mail.smtp.auth">true</prop>
        <prop key="mail.smtp.socketFactory.port">465</prop>
        <prop key="mail.smtp.socketFactory.class">
            javax.net.ssl.SSLSocketFactory
        </prop>
        <prop key="mail.smtp.socketFactory.fallback">false</prop>
    </props>
</constructor-arg>
<constructor-arg ref="smtpAuthenticator" />
</bean>
<bean id="mailSender"
    class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host" value="smtp.gmail.com" />
    <property name="session" ref="mailSession" />
</bean>
<bean id="mailMessage" class="org.springframework.mail.SimpleMailMessage">
    <property name="from" value="email@sermyadmin.org" />
</bean>
</beans>

```

These parameters are for sending e-mails using the Spring framework. The first parameter to be changed is the server that we will use to send e-mails. The following is the parameter from which those e-mails will appear to come from, and then you can set authentication parameters and mail servers. Restart Tomcat again and we're ready to go.

Step 3: Restart Tomcat and point your browser to `http://< server address>:8080/serMyAdmin`. You should be greeted with the login page.

username: admin@setup

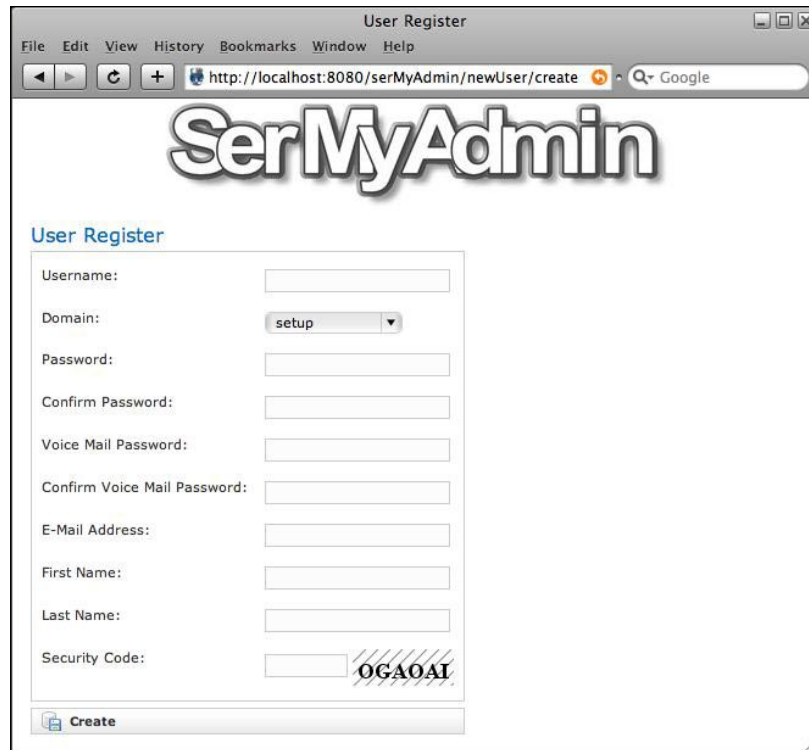
password: secret

Basic tasks

You can now use SerMyAdmin for a bunch of tasks. In this chapter, we will learn how to create and administer new users and groups. In later chapters, we will use SerMyAdmin for other tasks such as managing the OpenSIPS tables.

Registering a new user

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



The screenshot shows a web browser window titled "User Register". The address bar displays "http://localhost:8080/serMyAdmin/newUser/create". The page features the "SerMyAdmin" logo at the top. Below the logo, the heading "User Register" is followed by a registration form. The form includes the following fields: Username, Domain (with a dropdown menu currently showing "setup"), Password, Confirm Password, Voice Mail Password, Confirm Voice Mail Password, E-Mail Address, First Name, Last Name, and Security Code. A security code "0G40AI" is displayed next to the Security Code field. At the bottom of the form is a "Create" button.

Enter the fields **Username**, **Domain**, **Password**, **E-mail Address**, **First Name**, **Last Name**, **Voice Mail Password**, and the **Security Code**. Click on 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 e-mail about the registration. Before the user can make any calls, the administrator will have to previously approve him or her. In the newest version, the user can confirm the account by e-mail. He will receive an e-mail with a code that must be used to confirm the registration request. Anyway, you can still approve users manually.

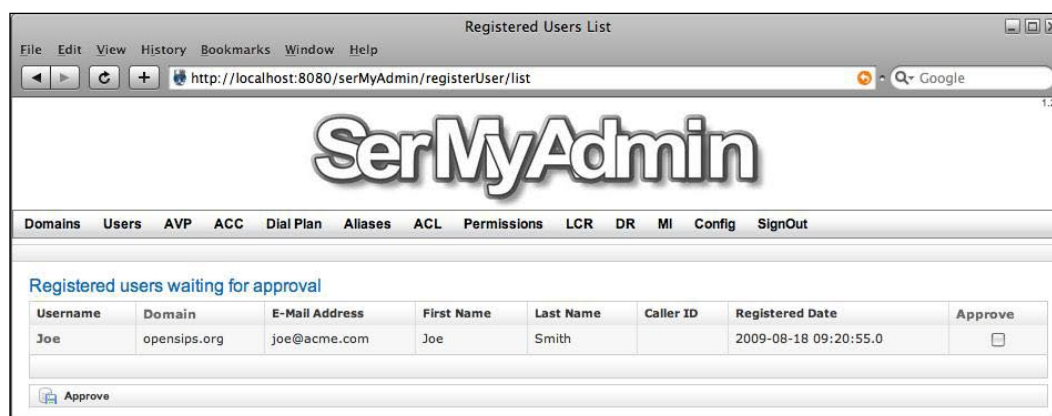
Approving a new user

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

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

Step 2: Select the **Users | Pending** menu item.

In the following screenshot, select the users you want to add and select the **Approve** checkbox. Click on the **Approve** button to add the user. The user will be removed from the `register_user` table and moved to the `subscriber` table, and then he or she will be able to register to OpenSIPS and start making calls.

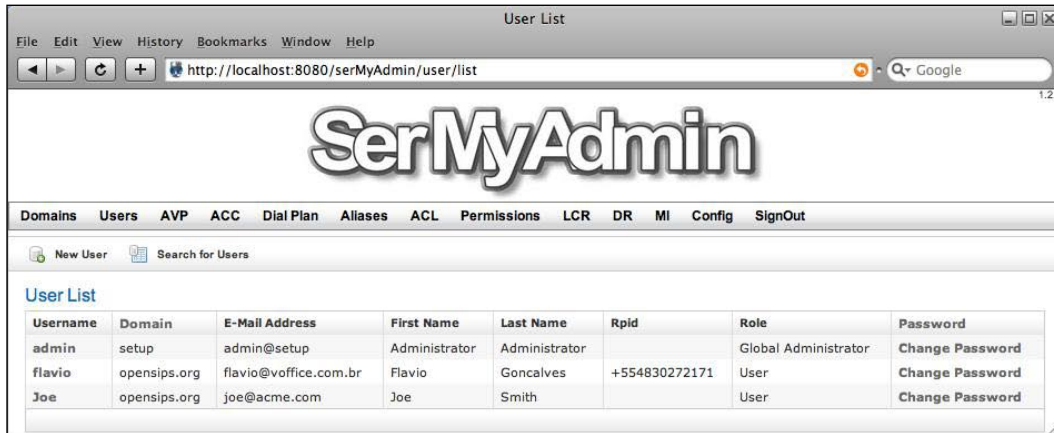


Step 3: Verify the user's insertion in the user's database, by selecting the **Users** menu item.

The user should now appear on your user list. Check this by clicking on the **Users** menu item. Don't forget to include the user in the right groups to allow dialing.

User management

You can view, add, edit, and delete users via the **Users | User List** menu. After selecting, all users will be displayed.

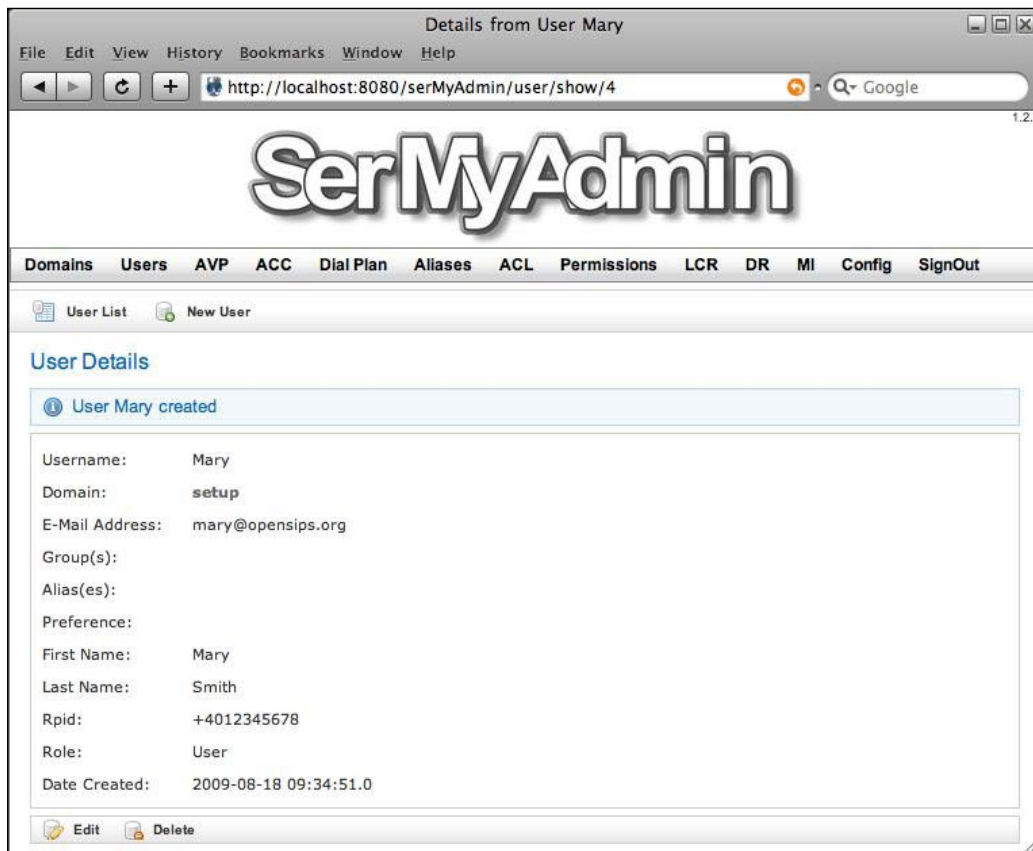


To add a new user, you must click on the **New User** link. You'll be directed to the following page:

The screenshot shows the "Created User" form in the SerMyAdmin application. The form is titled "Created User" and contains the following fields and options:

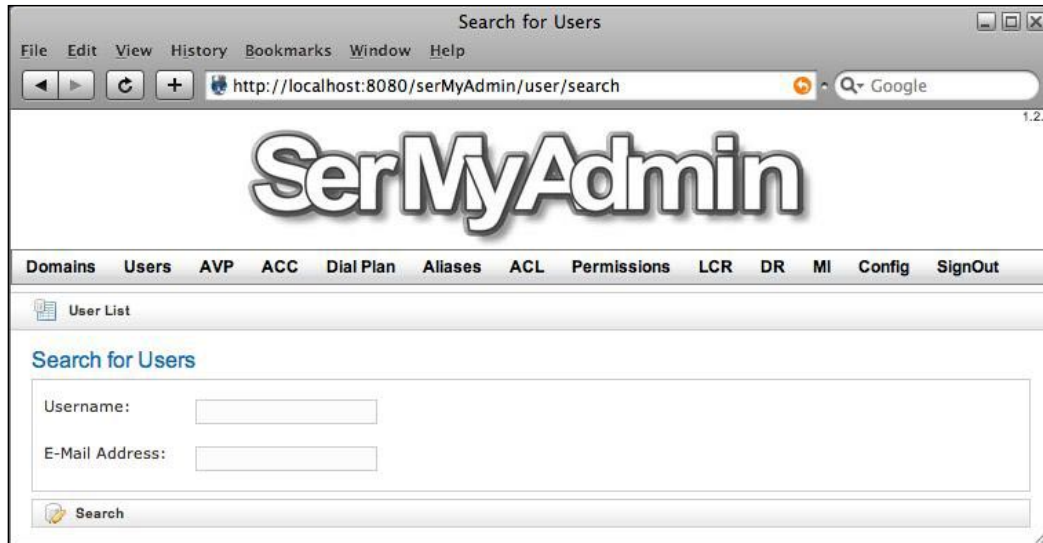
- Username:
- Domain:
- Password:
- Voice Mail Password:
- E-Mail Address:
- Group:
 - ☐ Int
 - ☐ 900
 - ☐ Ld
 - ☐ Mobile
 - ☐ Local
 - ☐ Toll Free
 - ☐ Interdomain
 - ☐ Net
 - ☐ Media
- First Name:
- Last Name:
- Rpid:
- Role:

In the last page, you must complete the fields and click on the **Create** link. 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 e-mail; click on the **Search** link, and fill only the desired criteria. On the following page, we will search for all users with the username **jdoe**. Click on **Search** and 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. Here, 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 who belong to that domain.

Interface customization

The SerMyAdmin interface is licensed according to GPL. You can download and modify the code. To do this, I suggest you to read at least the Grail tutorial. After learning some Grail, you may create a development environment using NetBeansIDE.

For its site layout, SerMyAdmin uses the **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 may be found at `/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin/WEB-INF/grails-app/views/layouts`.

There, you'll find the `main.gsp` and `notLoggedIn.gsp` files. These files are **Groovy Server Pages (GSP)** that control how the pages are displayed.

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

You may now change the `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 the `<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 in `/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin/images` and edit the tag that points to the `logo.png` in the layout files, as shown in the following code snippet:

```
<div class="logo"></div>
```

In the tag we just saw, we replaced the SerMyAdmin logo with one of our own, just changing the highlighted parameter.

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

Comparing OpenSIPS-CP and SerMyAdmin

The OpenSIPS-CP tool is more focused on the provisioning of OpenSIPS data, and to generate statistics and traces. It is the tool the super administrator will use most of the time. The SerMyAdmin tool is focused on the management of users. It has been developed to be used by users and administrators in the daily tasks related to users, DID (aliases), passwords, and accounting. You may deactivate the routing menus if you don't want your administrators to use it.

Summary

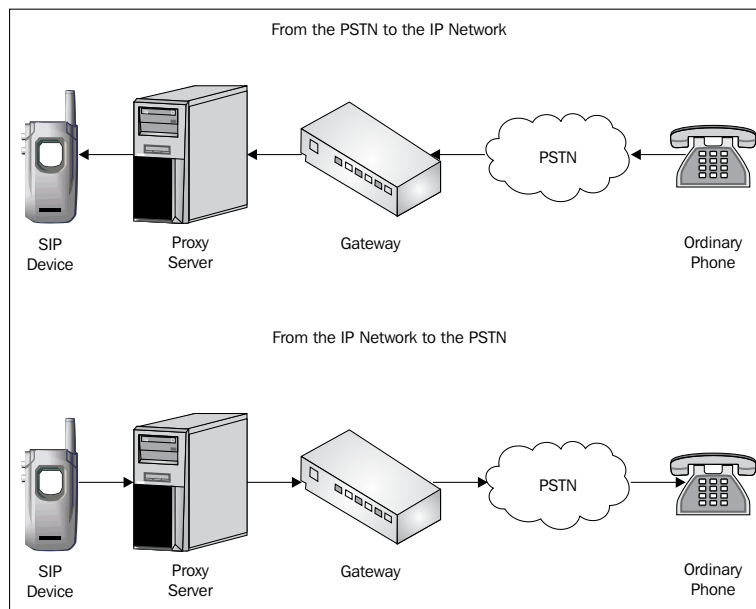
In this chapter, you learned why it is important to have a user and an administrator portal. It is a piece of software to which you should pay a lot of attention. Several VoIP providers fail to allocate time and resources to the important task of building the portal.

OpenSIPS is an amazing SIP proxy, but an SIP proxy is just one of the components in a VoIP provider. Without good administrator and user interfaces, a VoIP provider project may easily fail. SerMyAdmin is our contribution to your project. Developed using Groovy on Rails, it is licensed according to GPL. You 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. OpenSIPS Control Panel is the new tool developed by the OpenSIPS project. It is focused on the super administrator tasks and provisioning of data for advanced modules.

7

Connectivity to the PSTN

In the last two chapters, we prepared OpenSIPS to handle calls using authentication and a database. We used SerMyAdmin and `opensips-cp` to handle the database records. However, you still can't make calls to ordinary phones because you are not connected to the **public switched telephone network (PSTN)**. Now the challenge is to route calls to and from the PSTN.



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

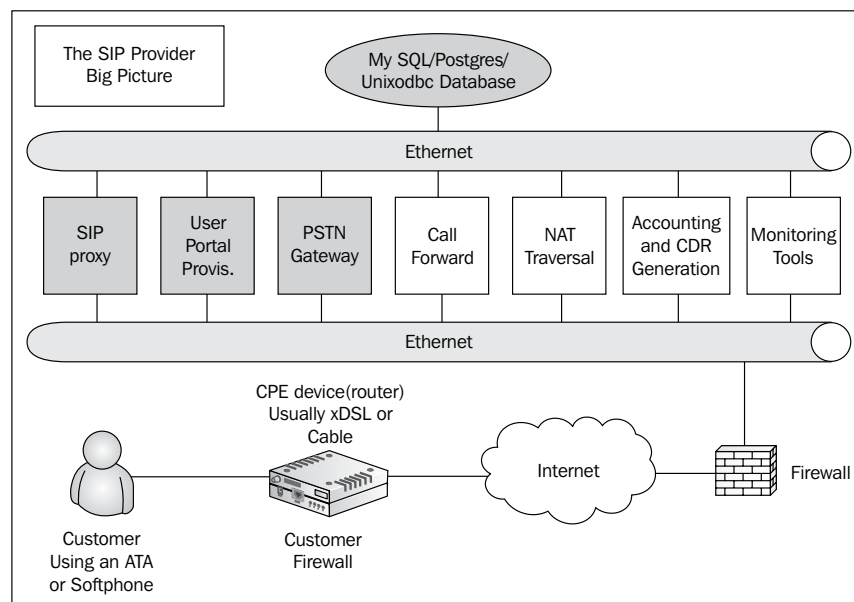
By the end of this chapter, you should be able to:

- Interconnect OpenSIPS to a SIP gateway
- Apply permissions to inbound and outbound calls
- Use ACLs to protect the PSTN gateway from unauthorized usage
- Use the **DIALPLAN** module to build dynamic dial plans
- Use the **DROUTING (dynamic routing)** module to route your calls
- Use opensips-cp to manage gateways permissions, gateways, and routes

In this chapter, you will learn how to make calls to the PSTN. We will introduce four new modules (DROUTING, DIALPLAN, 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 are going to be used to route the calls. It is very easy to find a tutorial on **regexps** on the Internet. If you are not familiar with regular expressions, which are also referred to as regexps, this quick reference card may help you: <http://www.visibone.com/regular-expressions/>.

The big picture

A VoIP provider solution has many components. To avoid losing the perspective, we will show this image in every chapter. In this chapter, we are working with the SIP proxy and the PSTN gateway.



After this chapter, our VoIP provider will be capable of sending calls to the PSTN using a SIP gateway.

Requests sent to the gateway

For the requests addressed to the gateway, we have to verify which group a certain user belongs to and check if he or she might 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");
    }
};

```

In the previous image, we show how to use the GROUP module. It is possible to check the membership of any user using the `db_is_user_in()` function. We have used the `credentials` criteria, which means we are checking if the user present in the Authorize header field belongs to a specific group.

The GROUP module

This module has two options, strict membership checking and regular expression based checking. With a strict membership database, there is no caching; whereas with regular expressions, caching is available due to performance reasons. For strict membership checking, we are going to use the following function:

```
db_is_user_in(URI, group)
```

Example:

```
db_is_user_in("credentials", "ld") #Use digest credential to verify
```

In this chapter, we are going to use strict checking because it is usually faster. Check the reference documentation if you prefer to use regular expression matching and the `db_get_user_groups()` function. For strict membership checking, this module exports the `db_is_user_in("credentials", "group")` function to check if a user belongs to a specified group. In the example, we have created three groups: `local` for local calls, `ld` for long distance, and `int` for international calls. In the last script, we used regular expressions to check if the call belongs to some of these groups.

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

```
opensipsctl acl show [<username>]
opensipsctl acl grant <username> <group>
opensipsctl acl revoke <username> [<group>]
```

Requests coming from the gateways

Incoming calls are usually handled by the `ALIAS_DB` module. You may map each incoming DID to subscriber using the `aliases` table. Most gateways don't support authentication, so it is important to bypass the user and password authentication. We are going to use the module permissions to authorize calls coming from the PSTN gateway without the digest authentication process.

The module permissions

This module can be employed in several scenarios such as:

- **Route permissions:** Using the `permissions.allow` and `permissions.deny` files, and the `allow_routing()` function. In this case, the file is populated with pairs (from `r-uri`). Check the format in the `/usr/src/opensips-1.6.x/modules/permissions/config/permissions.allow` source.
- **Register permissions:** According to the `register.allow` and `register.deny` files. To check the permissions use the `allow_register()` function.

- **Uri permissions:** According to the `permissions.allow` and `permissions.deny` files. Use the `allow_uri()` function to verify the rules. It is the same as route permissions. However, you will provide a variable as an argument for the function. For example:

```
permissions.allow
ALL : "^sip:361[0-9]*@abc\.com$" EXCEPT "^sip:361[0-9]*3@abc\.com$", "^sip:361[0-9]*4@abc\.com$"
```

This code would allow calls to a number starting with 361 with any number of digits in the domain `abc.com`, except where the final digit is 3.

- **Address permission:** Uses a database table named `address` and two new functions—`check_address(group_id, ip, port, proto [, context_info [, pattern]])` and `check_source_address(group_id, [, context_info [, pattern]])`. These functions are new to version 1.6 and replace the old `allow_trusted()` function. The `check_source_address()` function is a special case of the first function and is equivalent to `check_address(group_id, "$si", "$sp", "$proto", context_info, pattern)`. The following is a description of the parameters (extracted from the documentation):
 - `group_id`: This argument represents the group ID to be matched. The group can be used for the purpose of having different sets serving different scenarios. It can be an integer string or a `pvar` string. If the `group_id` argument is 0, the query can match any group in the cached address table.
 - `ip`: This argument represents the IP address to be matched. It can be given as a string or as a `pvar` string. This argument cannot be null/empty.
 - `port`: This argument represents the port to be matched. It can be given as an integer string or as a `pvar` string. Cached address table entry containing port value 0 matches any port. Also, a 0 value for the argument can match any port in the address table.
 - `proto`: This argument represents the protocol used for transport. It can be given as a string or as a `pvar` string. Transport protocol is either any or any valid transport protocol value such as, `udp`, `tcp`, `tls`, and `sctp`.
 - `context_info`: This argument represents the `pvar` in which the `context_info` field from the cached address table will be stored in case of match. This argument is optional.
 - `Pattern`: This argument is a string to be matched with the regular expression pattern field from the cached address table. This argument is optional.

See the following code example:

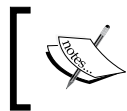
```
if (!check_source_address("0") {
    if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "0");
        exit;
    }
    else if (!check_from()) {
        sl_send_reply("403", "Use From=ID");
        exit;
    };
};
```

The `check_source_address("0")` function is exported by the `PERMISSIONS` module. When called, the function tries to find a rule matching the request.

To add a gateway to the address database, use `osipsconsole`.

```
osipsconsole address add 0 192.168.1.192 255.255.255.255 5060 UDP
osipsconsole address reload
```

It is common for gateways to not register with SIP proxy. Thus, requests coming from the gateway should not receive an error message "407-Proxy Authentication Required". 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 so the call will fail. To fix this, we will check the source IP address using the `check_source_address()` function instead of checking the credentials.



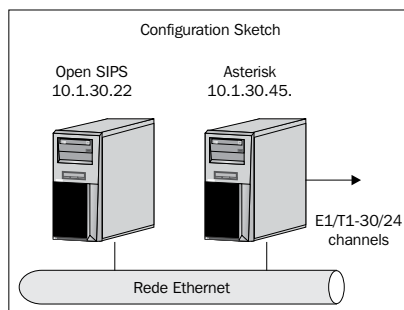
Don't forget to insert the gateway's address in the address table of your MySQL database for this script to work. Also, don't forget to reload the table in the cache.

To forward the call to the PSTN gateway, simply use the `rewritehostport()` function as follows:

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

Example

Here is a complete example of how to make calls to the PSTN using the concepts learned until this point.



This script is named `0745_07_01.cfg` in the code bundle. The complete code is as shown and the changes from previous scripts are highlighted. This will help you to understand where to insert the new code.

```
route{
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483", "Too Many Hops");
        exit;
    }
}

#---- Sequential requests section ----#
if (has_totag()) {
    if (loose_route()) {
        if (is_method("BYE")) {
            setflag(1); # do accounting
            setflag(3); # even if the transaction fails
        }
        else
            if (is_method("INVITE")) {
                record_route();
            }
        route(1);
    }
    else {
        if ( is_method("ACK") ) {
            if ( t_check_trans() ) {
                t_relay();
                exit;
            }
        }
    }
}
```



```
        else {
            exit;
        }
    }
    sl_send_reply("404","Not here");
}
exit;
}

#---- initial requests section ----#
if (is_method("CANCEL"))
{
    if (t_check_trans())
    {
        t_relay();
        exit;
    }
    t_check_trans();
    if (!(method=="REGISTER") && is_from_local())
    {
if(!check_source_address("0")){
        if (!proxy_authorize("", "subscriber")) {
            proxy_challenge("", "0");
            exit;
        }
        if (!idb_check_from()) {
            sl_send_reply("403","Forbidden auth ID");
            exit;
        }
        consume_credentials();
        # caller authenticated
    }
}

# preloaded route checking
if (loose_route()) {
    xlog("L_ERR",
        "Attempt to route with preloaded Route's [$fu/$tu/$ru/$ci]");
    if (!is_method("ACK"))
        sl_send_reply("403","Preload Route denied");
    exit;
}

# record routing
```

```

    if (!is_method("REGISTER|MESSAGE"))
        record_route();

# account only INVITES
    if (is_method("INVITE")) {
        setflag(1); # do accounting
    }

#---- Routing to external domains ----#
    if (!is_uri_host_local())
    {
        append_hf("P-hint: outbound\r\n");
        if(is_uri_host_local()) {
            route(1);
        }
        else {
            sl_send_reply("403","Not here");
        }
    }
    if (is_method("PUBLISH"))
    {
        sl_send_reply("503", "Service Unavailable");
        exit;
    }
    if (is_method("REGISTER"))
    {
        # authenticate the REGISTER requests (uncomment to enable auth)
        if (!www_authorize("", "subscriber"))
        {
            www_challenge("", "0");
            exit;
        }
        if (!db_check_to())
        {
            sl_send_reply("403","Forbidden auth ID");
            exit;
        }
        if (!save("location")){
            sl_reply_error();
            exit;
        }
        if ($rU==NULL) {
            # request with no Username in RURI
            sl_send_reply("484","Address Incomplete");

```

```
        exit;
    }
    # apply DB based aliases (uncomment to enable)
    alias_db_lookup("dbaliases");

#--- Routing to the PSTN section ---#
if (uri=~"^sip:[2-9][0-9]{6}@") {
    #Normalize the local number e.164-Miami(1305)
    if (db_is_user_in("credentials","local")) {
        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 (db_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]*@") {
    #Normalize for e164
    if (db_is_user_in("credentials","int")) {
        Strip(3);
        route(4);
        exit;
    }
    else {
        sl_send_reply("403", "No permissions for internat. calls");
        exit;
    };
};
```

```

};

#---- Routing to local users (urloc table) ----#
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;
    }
}

# when routing via usrloc, log the missed calls also
setflag(2);
route(1);
}

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

route[1] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {
        t_on_branch("2");
        t_on_reply("2");
        t_on_failure("1");
    }
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

branch_route[2] {
    xlog("new branch at $ru\n");
}

```

```
onreply_route[2] {
    xlog("incoming reply\n");
}

failure_route[1] {
    if (t_was_cancelled()) {
        exit;
    }
}
```

Inspection of the opensips.cfg file

The PERMISSIONS module gives OpenSIPS the ability to access the MySQL database to search for permissions. The `address` table is the repository for the trusted addresses. You should insert the IP address and transport protocol of each gateway into this database. This will allow the requests coming from the gateway to avoid the standard digest authentication.

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

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

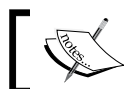
The first line in the following code snippet informs the module where to find the database, passing the required credentials. The second line instructs the modules to use caching on the database access to enhance performance.

```
# ----- Group -----
modparam("group", "db_url", "mysql://opensips:opensipsrw@localhost/opensips")

# ----- Permissions -----
modparam("permissions", "db_mode", 1)
modparam("permissions", "db_url", "mysql://opensips:opensipsrw@localhost/opensips")
```

When your proxy server receives an INVITE request, the normal behavior is to challenge the **user agent clients (UACs)** for their credentials. However, the PSTN gateways usually do not respond to the authentication. Therefore, you need to adopt a special procedure. The `check_source_address()` function will check the source IP address of the INVITE request against the address table of our database. If it matches, the request will be allowed. If it doesn't match, the requester will be challenged for their credentials.

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



Before trying this example, please insert the gateway's IP address into the address table.

You can use utilities such as OpenSIPS Control Panel or phpMyAdmin to maintain the database. It is easier than doing it manually in the MySQL **command-line interface (CLI)**. In the address table, insert the gateway's IP address and transport protocol (udp, tcp, tls, and so on).

The routing of calls according to the regular expressions is as shown:

```
#--- Routing to the PSTN section ---#
if (uri=~"^sip:[2-9][0-9]{6}@") {
    #Normalize the local number e.164-Miami(1305)
    if (db_is_user_in("credentials", "local")) {
        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 (db_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]*@") {
    #Normalize for e164
    if (db_is_user_in("credentials","int")) {
        Strip(3);
        route(4);
        exit;
    }
    else {
        sl_send_reply("403", "No permissions for internat. calls");
        exit;
    };
};
};
```

Local calls are identified by the number of digits (seven), and starting with a number in the range of two to nine ("^sip:[2-9][0-9]{6}@"). Before sending the number to the gateway, let's normalize the number using the local prefix. Long-distance numbers will match the regular expression "^sip:1[2-9][0-9]{9}@". The 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. Before sending it to the gateway, let's normalize the number by stripping the first three digits. 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 `opensipsctl` utility or SerMyAdmin.

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

```
LAB- route[4] {
    rewritehostport("10.1.30.45");
    route(1);
}
```

Using Asterisk as a PSTN gateway

Use the **Configuration Sketch** image provided previously in this chapter and write a script to send PSTN calls to a PSTN gateway. We are going to use the new command-line tool `osipsconsole` to complete these labs. While `opensipsctl` still works, its functionality is gradually being migrated to `osipsconsole`. The steps are as follows:

Step 1: Add the gateway address in the address table using `osipsconsole`.

```
osipsconsole
OpenSIPS$:address add 0 192.168.1.192 255.255.255.255 5060
OpenSIPS$:address reload
```

These records tell the `opensips` script to allow requests coming from the IP address 10.1.30.22 with UDP transport protocol.

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

```
osipsconsole domain add sermyadmin.org
```

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

```
OpenSIPS$: acl grant 1000@sermyadmin.org local
OpenSIPS$: acl grant 1000@sermyadmin.org ld
OpenSIPS$: acl grant 1000@sermyadmin.org int
OpenSIPS$: acl grant 1001@sermyadmin.org local
```

Step 4: Configuring Asterisk as a gateway.

Two very popular gateways for OpenSIPS 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 2 FXO interfaces and an Asterisk with an E1 PSTN card.



Warning:

It is important to prevent the direct sending of SIP packets to the gateways. The SIP proxy should be in front of the gateways 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 DIALPLAN (`extensions.conf`). We will configure Asterisk to send each call coming from the PSTN to the proxy and vice versa.

Asterisk gateway (sip.conf)

The code below is required to receive the calls in the Asterisk server. You have to change the file `sip.conf` to define parameters such as the landing context (`sipincoming`), and the allowed codecs (`ulaw` and `alaw`). In the file `extensions.conf`, you need to define the integrated DIALPLAN. I did it in a very simple way, any call coming from an FXO port will be directed to the SIP proxy and any call starting with a number coming from the SIP proxy will be sent to the PSTN.

```
[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
context=sipoutgoing
host=10.1.30.22
insecure=invite
disallow=all
allow=ulaw
```

Asterisk (`extensions.conf`):

```
root [sipincoming]
exten=>_ [0-9] .,1,Dial(DAHDI/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/fxoid@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 gateway 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 1/0 or 1/1 port as instructed by the dial peer voice 1 and 2 **Plain Old Telephone Service (POTS)** lines. 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 by making and receiving calls.

Dynamic routing

The last script is fine if you have just a few gateways. However, the VoIP providers usually have hundreds or thousands of routes. Dynamic routing, or DROUTING, is a module capable of efficiently routing a large number of routes—more than 3,00,000 according to the documentation—using criteria such as prefix, time, and group of users. The features of DROUTING are as follows:

- **Rule selection:**
 - Prefix based
 - Caller/group based
 - Time based
 - Priority based
 - Blacklisting
- **Processing:**
 - Stripping and prefixing
 - Default rules
 - Inbound and outbound processing
 - Script route triggering
- **Failure handling:**
 - Serial forking
 - Weight-based gateway selection
 - Random gateway selection

Most relevant parameters

Some parameters have a strong influence on the behavior of the DROUTING module.

Sort order

This defines the selection order of the gateways (default 0):

- **0:** All the destinations are tried in the given order.
- **1:** Destinations are grouped using the ; character. Destinations are selected randomly inside each group. For example, 1,2;3,4;5,6. Some results possible are 1-2-3-4-5-6, 2-1-4-3-6-5, and so on.
- **2:** Destinations are grouped using the ; character. Destinations are selected randomly, one for each group. For example, 1,2;3,4;5, 6. Some results possible are 1-3-5, 1-4-5, 1-3-6, and so on.

Blacklist

This defines a blacklist based on a list of gateways. Multiple instances of this parameter are allowed. For example:

```
modparam("drouting", "define_blacklist", 'bl_name= 20,21')
modparam("drouting", "define_blacklist", 'list= 1,2')
```

Force_dns

If set, this resolves DNS names of the gateways during the startup. The default value is 1, enabled.

Drouting tables

This module uses four database tables to store rules and gateways. These tables are shown as follows:

DR_GATEWAYS: Table used to define the gateways.

Column name	Type	Default value	Description
gwid	integer	Auto increment	Unique identifier for the gateway
Type	unsigned int	0	Type/class of the gateway
Address	varchar(128)		Address of the gateway
Strip	unsigned int	0	No of digits to strip
pri_prefix	varchar(255)		PRI prefix of the gateway
Description	varchar(128)		Description of the gateway

Example:

gwid	Type	Address	Strip	pri_prefix	Description
1	10	10.10.10.10:5080	0	2222	Gateway 1
2	10	10.10.10.10	2	3333	Gateway 2
3	20	10.10.10.11	0		Gateway 3

DR_RULES: Table for the definition of routing rules based on prefixes, time, and priority.

Column name	Type	Default	Description
ruleid	integer	auto	UID of the rule
Groupid	varchar(255)		List of routing group IDs
Prefix	varchar(64)		Destination prefix for this route
Timerec	varchar(255)		Time recurrence for this rule
Priority	integer	0	Priority of the rule
Routeid	integer	0	Route block to be executed
Gwlist	varchar(255)		The list of gateways to be used
description	varchar(128)		Description of this rule

Example:

ruleid	Group	Prefix	Timerec	Priority	routeid	Gwlist	Description
1	6	0049	20040101T083000 10H weekly M O,TU,WE,TH,FR	5	23	1,2	Rule 1
2	8	0049	20040101T083000	0	0	1,2	Rule 2
3	7,8,9	0049	20040101T083000				

DR_GROUPS: Table used to associate routes to specific users.

Name	Type	Description
id	unsigned int	Unique ID
username	string	Username part of user
domain	string	Domain part of user
groupid	integer	The ID of the routing group the user belongs to
description	string	Text description of the group/user

DR_GW_LISTS: This is a table created to associate a list of gateways. You can specify a list of gateways in the rules table using the # character before the list number.

Name	Type	Description
id	unsigned int	Unique ID
gwlist	string	Reference to the gateways/destinations from the list
description	string	Text description of the gateway list

Case study for dynamic routing

To better understand how drouting works, let's simulate a real situation. Suppose you have the following routing specifications:

1. One group of users:
 - 0: No routes
 - 1: All routes during work hours (9-5, Monday-Friday)
 - 2: All routes during the whole day
2. We will have two gateways:
 - USA: 192.168.0.200
 - Europe: 192.168.0.201
 - Asia: 192.168.0.202
3. Users:
 - 1000 and 1001 in group 2
 - 1002 in group 1
4. Rules:
 - Destination prefix 1: USA, failover in Europe
 - Destination prefix 40: Europe, failover in USA
 - For international numbers, the user should use the 011 prefix
 - For long distance numbers, 1 + area code + number
 - For local numbers, any 7 numbers starting from 2-9

To implement these features, follow these steps:

Step 1: Add the following code to the proper sections of the `opensips.cfg` file.

```
##### Modules Section #####
loadmodule "drouting.so"

# ----- setting module-specific parameters -----
# ----- drouting params -----
modparam("drouting", "sort_order", 0)
modparam("drouting", "use_domain", 1)
modparam("drouting", "db_url", "mysql://opensips:opensipsrw@localhost/
opensips")

route[4] {
    #---- PSTN route ----#
    if(!do_routing()){
        send_reply("503", "No rules found matching the URI prefix");
        exit;
    }
    # flag 10 - flag the transaction to handle the failure route
    setflag(10);
    route(1);
}

failure_route[1] {
    if (t_was_cancelled()) {
        exit;
    }
    if(isflagset(10)){
        if (use_next_gw()) {
            #xlog ("next gateway $ru \n");
            t_on_failure("1");
            t_relay();
            exit;
        }
        else {
            t_reply("503", "Service not available, no more gateways");
            exit;
        }
    }
}
```

Step 2: Now, you will have to populate the DROUTING tables according to the scenario. Use `opensips-cp` to populate the tables.

Rules table:

ID	Group ID	Prefix	Priority	Route ID	GW List	Description	Details	Edit	Delete
1	0	1	1	0	1	Routes to USA			
No Time Recurrence.									
2	0	40	1	0	2	Routes to Romania			
No Time Recurrence.									
Page: 1									
Total Records: 2									

Gateways table:

ID	Type	Address	Strip	PRI Prefix	Description	Status	Details	Edit	Delete
1	2	192.168.1.201:5061	0		Gateway USA	● / 1			
2	2	192.168.1.192:5061	0		Gateway Romania	● / 1			
Page: 1							Total Records: 2		

DIALPLAN transformations

The module DIALPLAN implements generic string translations based on matching and replacement rules. The module uses a set of transformation rules stored in a database, each rule describing how a matching should be made. Usually, you will set an attribute or apply a transformation to the `r-uri` or a pseudo variable.

There are two types of matching rules: string and regular expressions, and overlapping expressions. Overlapping matching expressions can be controlled using priorities. The first matching rule is the one to be processed.

One of the goals for a good script is to be almost static. Transferring the DIALPLAN from the code to the database makes your implementation more flexible. The module DIALPLAN will help you to implement generic dial plans converting numbers and detecting ranges.

In the following example, we will use the DIALPLAN module for auto completion and range detection. We will be able to accommodate different dial plans per country.

DIALPLAN example

I always like to explain code by example, so let's assume the requirements. Before starting, please download a quick reference card for regular expressions. This is going to make your work a lot easier. Search for the phrase "Regular Expression Quick Reference Card" in your favorite search engine and you will find lots of cards. I'll not provide a specific URL because they change very often. All numbers will be normalized to the E.164 format to simplify routing and billing.

1. In this case, we are going to use a fixed dialplan number "0". You could use different dial plans for different users. To do this, you could include an extra field in the subscriber table and load it into an **attribute value pair (AVP)**.
2. In the dialplan ID 0, we are going to have the following rules:
 - Numbers starting with 6, plus 5 digits, will receive the `usrloc` attribute and will require no further processing. Regular expression: `^6[0-9]{5}`.
 - Numbers with 7 digits starting with 2 to 9 will receive the `pstn` attribute, and will require the appending of the country code and area code. Regular expression: `^[2-9][0-9]{6}`.
 - Numbers starting with 1, plus 7 digits, will receive the `pstn` attribute. Regular expression: `^1[2-9][0-9][0-9][2-9][0-9]{6}`.
 - Numbers starting with 011 or + will receive the `pstn` attribute. Regular expression: `^(011|\+)[1-9][0-9]`.
 - We will not use group checking anymore. If you don't want a class of users to use a specific range of calls, create a dialplan that excludes these destinations and assign this dialplan to your users.

Step 1: Change the `opensips.cfg` file. Add the following excerpts in the proper sections:

```
##### Modules Section #####
loadmodule "avpops.so"
loadmodule "dialplan.so"

#----- avpops params -----
modparam("avpops", "db_url", "mysql://opensips:opensipsrw@localhost/opensips")
modparam("avpops", "avp_table", "usr_preferences")

#----- load the dpid field to a pseudo-variable $avp(s:dpid) -----
modparam("auth_db", "load_credentials", "$avp(s:rpip)=rpip ;$avp(s:countrycode)=countrycode;$avp(s:areacode)=areacode")
```

```

modparam("dialplan", "db_url", "mysql://opensips:opensipsrw@localhost/
opensips")
## attribute of the matched line will be store in the $avp(s:dest)
modparam("dialplan", "attrs_pvar", "$avp(s:dest)")

```

In the routing section, just after the alias lookup, remove the old code and insert the following code. Use the `0745_07_03.cfg` file in the code bundle as a reference, if needed.

```

# apply DB based aliases (uncomment to enable)
alias_db_lookup("dbaliases");
if(!dp_translate("0", "$rU/$rU")){
    send_reply("420", "Invalid Destination");
    exit;
}
xlog("$avp(s:dest)");
if ($avp(s:dest)=="usrloc") {
    #Route to usrloc
    route(3);
}
if ($avp(s:dest)=="pstn") {
    #route to pstn
    route(4);
}
if ($avp(s:dest)=="media") {
    #route to media server
    route(5);
}
send_reply("420", "Invalid Extension");
exit;
}

route[1] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {
        t_on_branch("2");
        t_on_reply("2");
        t_on_failure("1");
    }
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

#Route for user lookups

```

```
route[3]{
  if (!lookup("location", "m")) {
    switch ($retcode) {
      case -1:
      case -3:
        t_newtran();
        t_reply("404", "Not Found");
        exit;
      case -2:
        sl_send_reply("405", "Method Not Allowed");
        exit;
    }
  }
  # when routing via usrloc, log the missed calls also
  setflag(2);
  route(1);
}

route[4] {
  #---- PSTN route ----#
  if(!do_routing()){
    send_reply("503", "No rules found matching the URI prefix");
    exit;
  }
  # flag 10 - route to pstn
  setflag(10);
  route(1);
}

route[5] {
  #---- Route to media servers ----#
  xlog("route to media servers");
}

branch_route[2] {
  xlog("new branch at $ru\n");
}

onreply_route[2] {
  xlog("incoming reply\n");
}

failure_route[1] {
  if (t_was_cancelled()) {
    exit;
  }
}

if(isflagset(10)){
```

```

    if (use_next_gw()) {
        xlog ("next gateway $ru \n");
        t_on_failure("1");
        t_relay();
        exit;
    }
    else {
        t_reply("503", "Service not available, no more gateways");
        exit;
    }
}
}
}

```

Inspection of the file opensips.cfg

We've changed a lot of things in this script. Let's examine these changes carefully:

```
modparam("dialplan", "attrs_pvar", "$avp(s:dest)")
```

We are going to use the DIALPLAN module to classify and transform the calls. The DIALPLAN will be able to automatically detect what kind of number was dialed. The `avp(s:dest)` will receive the attribute associated with the pattern.

```

if(!dp_translate("0","$rU/$rU")){
    send_reply("420", "Invalid Destination");
    exit;
}

```

The `dp_translate()` function searches the `dialplan` table to find a pattern and apply the translations needed. Beyond this, you can assign an attribute to the `avp` named `s:dest` depending on the match.

Step 2: Customize the DIALPLAN control panel

```
vi /var/www/opensips-cp/config/tools/dialplan/local.inc.php
```

Also, change the following lines:

```

$config->attrs_cb=array(
    // name , description
    array("usrloc","Send to user location"),
    array("pstn","Send to PSTN"),
    array("media","Send to media server"),
    #array("d","Descr d"),
    #array("e","Descr e"),
    #array("f","Descr f"),
);

```

Step 3: Insert the data from the following table:

Dialplan ID	Rule Priority	Matching Operator	Matching Regular Expression	Matching String Length	Substitution Regular Expression	Replacement Expression	Attributes	Edit	Delete	Clone
0	1	1	^(011 +)[0-9].*	0	^(011 +)(([0-9].*)	\2	pstn			
0	1	1	^/*	0			media			
0	1	1	^1[2-9][0-9][0-9][2-9][0-9]{6}.*	0			pstn			
0	1	1	^6[0-9]{5}	0			usrloc			
0	1	1	^[2-9][0-9]{6}.*	0	(^[2-9][0-9]{6}).*	1305\1	pstn			
Page: 1								Total Records: 5		

Before interpreting the previous table, have a regular expression quick reference card in hand. In this plan, when we have matches for internal numbers starting with 6 followed by 5 digits (`^6[0-9]{5}`), the `dp_translate` command will assign the `usrloc` attribute to the `avp(s:dest)`, signaling that this number should be handled by the user location table. In the second line, we are catching local numbers starting with a number from 2 to 9 followed by 6 digits (`^[2-9][0-9]{6}`). Pay attention to the fourth line. We are catching international numbers starting with 011 or +, and we are changing the number with a replacement expression. This will help to normalize the number before sending it to the routing table. With the DIALPLAN module, it is possible to handle many situations such as common user mistakes.

```

if ($avp(s:dest)=="usrloc") {
    #Route to usrloc
    route(3);
}

if ($avp(s:dest)=="pstn") {
    #route to pstn
    route(4);
}

if ($avp(s:dest)=="media") {
    #route to media server
    route(5);
}

send_reply("420", "Invalid Extension");
exit;

```

In the last code snippet, we check the value of the `avp(s:dest)`, and check how to handle and normalize the call. Local calls are prefixed with country code and area code.

Step 4: Restart the server. Now try calling some destinations and check if your code is working fine.

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 (local generated 408 timeout) or is responding with a response code type 503, OpenSIPS uses a resource called **DNS blacklist** and inserts your gateway in the blacklist. Your gateway will stay in the blacklist for four 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 make 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 gateways that are permanently or temporarily out of service.

Summary

In this chapter, you have learned how to configure OpenSIPS to forward calls to a gateway. It is important to take care of the security. Using the **PERMISSIONS** module, you can allow the gateways to bypass the digest authentication and permit them by validating only their IP addresses. The **GROUP** module is important for controlling the access from the UACs. It is also interesting 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. Analyze your call detail records frequently to detect abnormal activity.

8

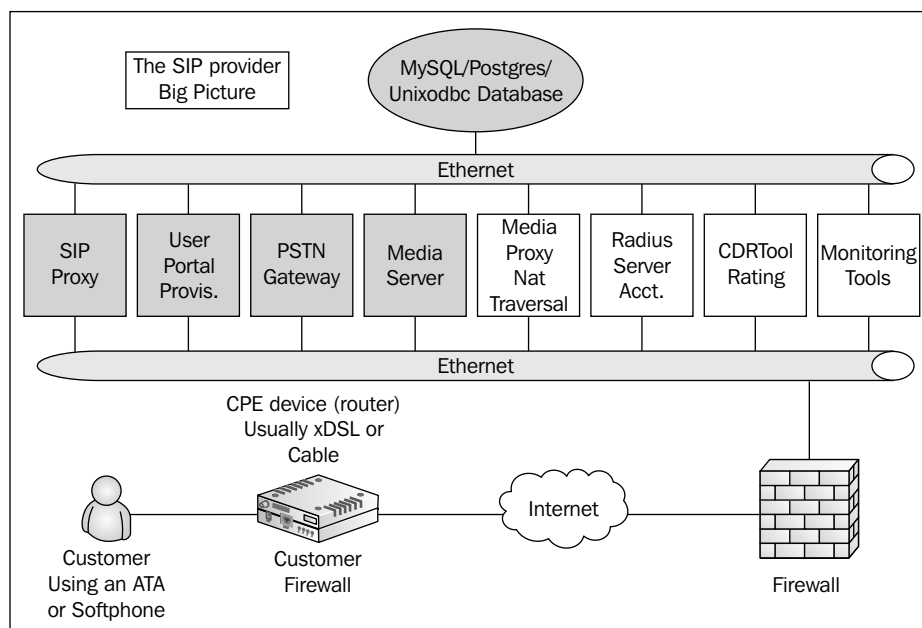
Media Services Integration

One important aspect of OpenSIPS is that it does not handle media or RTP. Therefore, even for simple services such as playing an announcement you will have to rely on an external media server. The most common media server in use with OpenSIPS is Asterisk. Recently, Freeswitch™ and Yate™ are gaining traction to offer media services such as conference, voicemail, announcements, IVR, and so on. In this chapter, we will learn about parallel and serial forking and how to integrate OpenSIPS with a media server. We've chosen Asterisk because of its popularity and simplicity.

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

- Describe concepts such as parallel and serial forking
- Play announcements from the Asterisk server
- Integrate the database of Asterisk and OpenSIPS
- Implement call forward to a voicemail
- Implement call forward on busy and unanswered calls to a voicemail
- Implement miscellaneous services such as conference

Verifying our progress, the VoIP provider solution has many components. To avoid losing perspective, we will show this picture in every chapter. In this chapter, we are working with the Media Server component (Asterisk). The call forwarding feature will help us to understand important concepts such as serial forking and failure routing. The situation covered in this chapter is voicemail. A media server might be used for several applications such as IVRs, to play prompts, text-to-speech, and voice recognition.



Well, our provider is evolving; after installing the SIP proxy and the user portal, it is time to connect to the PSTN for advanced services such as voicemail and conference. These services are provided by a component called media server. Some important concepts such as serial and parallel forking will be covered too.

Playing announcements

Sending calls to a media server is not "rocket science". All you have to do is rewrite the host and port of the R-URI with the IP address of the media server using the `sethostport` or the `rewritehostport` commands.

Example: playing demo-thanks

The `demo-thanks.gsm` file is a recording present on each Asterisk server installed. Let's create a number like `*100` to send a call to the Asterisk server. From now on, each number prefixed by the `*` character will be sent to the media server. In the route section of the file, just after the alias processing, insert:

```
if($rU=~"^\\*") {
    rewritehostport("ip_address_of_the_media_server");
    route(1);
}
```

It is very simple, any number starting with `*` will be sent to the media server. Now in the Asterisk server, we have to receive the call. In order to do this, we will have to edit the `sip.conf` and `extensions.conf` files in the `/etc/asterisk` directory. Usually, we install both Asterisk and OpenSIPS in the same server for testing and educational purposes, and OpenSIPS in the port 5060 and Asterisk in the port 5062.

The file `sip.conf`:

```
[general]
context=from-sip
bindaddr=0.0.0.0
bindport=5062
allowguest=no

[opensips]
type=peer
host=192.168.1.201
context=from-sip
insecure=invite
allow=ulaw;alaw;gsm
deny=0.0.0.0/0.0.0.0
permit=192.168.1.201/255.255.255.255
```

Although we are allowing calls without authentication (`insecure=invite`), we are preventing any other user, except the SIP proxy itself to send calls using IPTABLES.



Do not allow unauthenticated calls to your Asterisk server from any IP address, except from the SIP server

Unauthorized calls are very dangerous. The average spending for victims of toll-fraud is beyond tens of thousands of dollars. You can further improve your security by configuring IPTABLES in your Asterisk server to receive calls only from the OpenSIPS server where users are being authenticated. The IPTABLES configuration is beyond the scope of this book.

The `extensions.conf` file:

```
[from-sip]
exten=*100,1,answer()
exten=*100,n,playback(demo-thanks)
exten=*100,n,hangup()
```

In the `extensions.conf` file, we just have the basics — redirecting calls starting with `*100` to a playback message.

Voicemail

Voicemail is a bit more complicated, as you need to define the users in the Asterisk server. If you have only five or ten users, you could go directly to the file `/etc/asterisk/voicemail.conf` and create the same users there manually. However, VoIP providers usually have tens of thousands of users and the manual configuration option might become very troublesome. Well, it is possible to integrate Asterisk using the real-time tables. Asterisk Real Time was created for Version 1.2. It allows you to pull the configuration for users, peers, voicemail, queues, and other information from a database. Here is tutorial on the subject.

Another important issue is where to store the voicemail messages. Suppose you want to use more than one voicemail server for scalability purposes. In this case, you will need to use a centralized storage. Fortunately, Asterisk allows the use of a voicemail-centralized storage using **Internet Message Access Protocol (IMAP)** or **Open Database Connectivity (ODBC)**. I've chosen ODBC because we are already using it for Asterisk Real Time.

How to integrate Asterisk Real Time with OpenSIPS

Step 1: First let's install the ODBC driver for MySQL. You could configure Asterisk Real Time with native MySQL, but the voicemail storage is made always using ODBC. Another advantage for ODBC is the possibility to avoid the installation of the `asterisk-addons` package. Please, install the `unixodbc` driver if it is not already installed.

```
apt-get install unixodbc libmyodbc
```

Step 2: Configure the ODBC driver for MySQL using the following commands:

```
vi odbcinst.ini
[MySQL]
Description           = MySQL driver for Linux & Win32
Driver                = /usr/lib/odbc/libmyodbc.so
Setup                 = /usr/lib/odbc/libodbcmyS.so
FileUsage              = 1
UsageCount             = 2
cd /etc
odbcinst -i -d -f odbcinst.ini
vi odbc.ini
[asterisk]
Description = MySQL Asterisk
Driver      = MySQL
SERVER      = localhost
USER        = opensips
PASSWORD    = opensipsrw
PORT        = 3306
DATABASE    = opensips
Option      = 3
```

Step 3: Test your connectivity using `isql`:

```
isql asterisk
```

Step 4: Create the views and tables in the OpenSIPS database for Asterisk (This is already made by the `serMyAdmin` utility. If you have installed it, you may skip this step—if you haven't installed it, please add the `vmail_password` column to the subscriber's table).

```
mysql -u root -p
use opensips;
CREATE VIEW vmusers AS
SELECT id as uniqueid,
username as customer_id,
'default' as context,
username as mailbox,
vmail_password as password,
username as fullname,
email_address as email,
NULL as pager,
datetime_created as stamp
FROM opensips.subscriber;
CREATE VIEW sipusers AS
SELECT username as name,
username as username,
```

```
'friend' as type,
NULL as secret,
domain as host,
rpid as callerid,
'from-sip' as context,
username as mailbox,
'yes' as nat,
'no' as qualify,
username as fromuser,
NULL as authuser,
domain as fromdomain,
NULL as insecure,
'no' as canreinvite,
NULL as disallow,
NULL as allow,
NULL as restrictcid,
domain as defaultip,
domain as ipaddr,
'5060' as port,
NULL as regseconds
FROM opensips.subscriber;

CREATE TABLE `voicemessages` (
  `id` int(11) NOT NULL auto_increment,
  `msgnum` int(11) NOT NULL default '0',
  `dir` varchar(80) default '',
  `context` varchar(80) default '',
  `macrocontext` varchar(80) default '',
  `callerid` varchar(40) default '',
  `origtime` varchar(40) default '',
  `duration` varchar(20) default '',
  `mailboxuser` varchar(80) default '',
  `mailboxcontext` varchar(80) default '',
  `recording` longblob,
  PRIMARY KEY (`id`),
  KEY `dir` (`dir`)
) ENGINE=InnoDB;
```

Step 5: Configure Asterisk to use the OpenSIPS database.

```
cd /etc/asterisk
```

```
vi res_odbc.conf
```

```
[asterisk]
enabled => yes
dsn => asterisk
username => opensips
password => opensipsrw
pre-connect => yes
```

Check if the ODBC driver is connected using:

```
asterisk -r
debian*CLI> odbc show
Name: asterisk
DSN: asterisk
Pooled: no
Connected: yes
```

Step 6: Configure the Asterisk Real Time config file.

```
vi extconfig.conf
; Static and realtime external configuration
; engine configuration
;
; Please read doc/extconfig.txt for basic table
; formatting information.
;
[settings]
sipusers => odbc, opensips, sipusers
sippeers => odbc, opensips, sipusers
voicemail => odbc, opensips, vmusers
```

Step 7: Configure the voicemail.conf file and uncomment the lines related to ODBC storage.

```
; Voicemail can be stored in a database using the ODBC driver.
; The value of odbcstorage is the database connection configured
; in res_odbc.conf.
odbcstorage=asterisk
; The default table for ODBC voicemail storage is voicemailmessages.
odbcstorage=voicemailmessages
```



Troubleshooting:

Use the `modules show` command to check if the `res_config_odbc.so` module is loaded on the Asterisk server. If it is not loaded, please check the file `/etc/asterisk/modules.conf` to verify if you have any line preventing the load. Additionally, please check the Asterisk compilation options using `make menuselect`, and check if this module was compiled during the Asterisk installation. Check specifically the `voicemail build options` option for ODBC support.

Step 8: Configure the Asterisk dialplan in the `extensions.conf` file.

In the VIEW definition I have used "from-sip" as the context for calls incoming from OpenSIPs.

vi extensions.conf

```
[from-sip]
exten=_u.,1,voicemail(${EXTEN:1},u)
exten=_u.,n,hangup()
exten=_b.,1,voicemail(i${EXTEN:1},b)
exten=_b.,n,hangup()
exten=*98,1,meetme(d)
exten=*98,2,hangup()
exten=*99,1,voicemailmain(${CALLERID(num)},s)
exten=*99,2,hangup()
exten=*100,1,answer()
exten=*100,2,playback(demo-thanks)
exten=*100,3,hangup()
```

Step 9: Test the access to the conference by dialing *98 or to the voicemail menu using *99.

Call forwarding

Now we are going to work with call forwarding. There are usually three kinds of call forwarding. 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 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, whether it's 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 by sending the INVITE request to the final destination.
- **Forward on no answer:** If a phone replies to an INVITE request with a "480 Temporarily unavailable" message, or the SIP proxy generates a timeout with a "408 request Timeout" message, OpenSIPS again will use `failure_route` to intercept the message and create a new leg, sending the INVITE to the final destination.

All call forwarding destinations are stored in the `user_preferences` table. We are going to revisit some concepts in this chapter such as AVPs (attribute value pairs) and pseudo-variables. AVPs are made available by the OpenSIPS core. The **Attribute-Value Pairs Operations (AVPOPS)** module provides several functions for manipulating AVPs, such as interaction with SIP requests and the database, operations with strings, and operations with regular expressions.

Implementing blind call forwarding

For the blind call forwarding, we are going to use an AVP named `callfwd` to store the URI of the call forward destination. In the `avp_db_load` function, the first parameter is the source and the second is `avp_name`. So, this function will load the value of the attribute `callfwd` in the `callfwd` string AVP for the user, matching the requested uri (`$ruri`) in the columns `username` and `domain`.

```
avp_db_load("$ru", "$avp(s:callfwd) ")
```

Later on, we will push this AVP to the SIP packet, changing the original `$ruri` to the new one.

```
$ru = $avp(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 `s:callfwd` AVP. The magic with call forwarding 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", "db_url", "mysql://opensips:opensipsrw@localhost/opensips")
modparam("avpops", "avp_table", "usr_preferences")
```

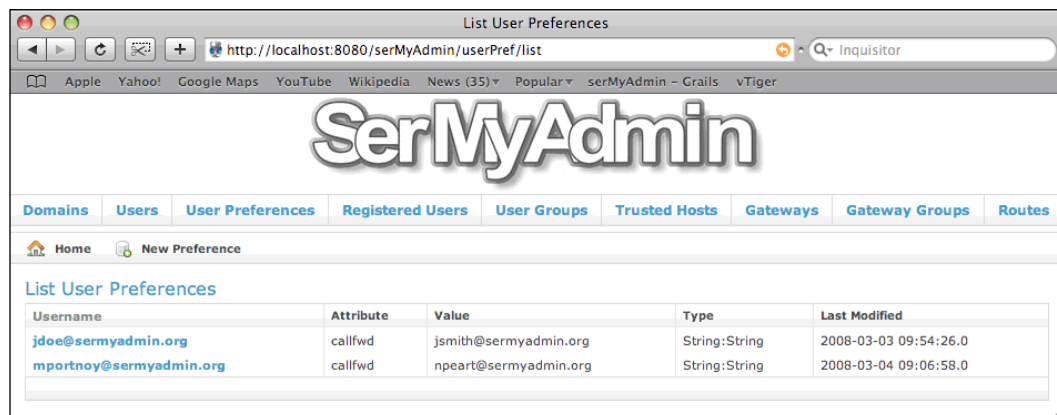
In the first place, let's implement the blind call forward service. In the INVITE processing, we will load the AVP named `callfwd` from the user preference table in the database. If the `callfwd` preference was set to this specific user, we will push it to the R-URI before forwarding the request.

```
if(avp_db_load("$ru", "$avp(s:callfwd)")) {
    $ru = $avp(s:callfwd);
    route(1);
    exit;
}.
```


In order to make this feature work, it is important to insert the correct entries in the database. The table used by the AVPs is `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 the **User Preferences** tab on the menu. There you can view all user preferences, edit them, and create new ones. Use `fwdbusy` for Forward on Busy and `fwdnoansw` for Forward on no answer.



With the previous record, we are telling the system to forward any call made to 1001 to the URI `sip:1004@yourdomain`.

Lab—implementing blind call forwarding

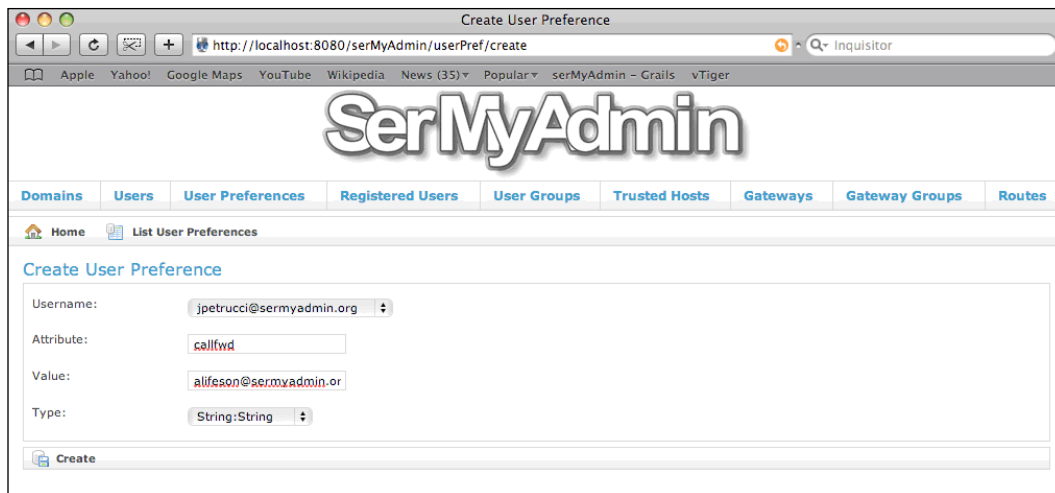
Step 1: Let's insert the AVP pairs using the `serMyAdmin` interface first seen in Chapter 6, *Graphical user interfaces for OpenSIPS*.

In your browser, access the SerMyadmin interface:

`http://<your-ip-server-address>:8080/serMyAdmin`

Step 2: Login 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 `opensips.cfg` file to include the instructions explained above. The file should appear as shown here. Include the following lines in the `opensips.cfg` file, or simply use the file `0745_08_01.cfg` provided in the code bundle and copy it to `opensips.cfg`.

In the module loading section, add:

```
loadmodule "avpops.so"
loadmodule "xlog.so"
```

Add the following in the module parameters section:

```
modparam("avpops", "db_url", "mysql://openser:openser@localhost/
openser")
modparam("avpops", "avp_table", "usr_preferences")
```

Just after the `alias_db_lookup` section, insert the following excerpt:

```
if(avp_db_load("$ru", "$avp(s:callfwd)")) {
    $ru = $avp(s:callfwd);
    xlog("forwarded to: $avp(s:callfwd)");
    route(1);
    exit;
}
```

Step 5: Register the phones 1001 and 1004. Call from the 1001 phone to the 1000 phone. It should forward the call to the 1004 phone 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 a new important concept called `failure_route`. We will handle the following failure situations:

- 408 – Request Timeout (phone not registered or disconnected)
- 480 – Temporarily Unavailable (no answer)
- 486 – Busy Here
- 487 – Request Cancelled

In order to implement "call forward on busy" and "call forward when unanswered", we will use the concept of failure route. In the logic that follows, we will call the function `t_on_failure("1")` just before sending the INVITE to the standard processing. This allows us to handle the SIP failure messages (with reply codes higher than 299 – also called negative replies).

We could use the same logic used for blind call forwarding – setting an AVP to redirect call forward to a URI pointing to the destination. To simplify, when receiving a call in this situation, we will forward it to a voicemail system. If you want to implement call forwarding to a specific destination, you could use something like the following code in the failure route.

```
if(avp_db_load("$ru", "$avp(s:fwdbusy)")) {
    $ru = $avp(s:fwdbusy);
    xlog("forwarded to: $avp(s:fwdbusy)");
}
```

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. The Asterisk server will process the voicemail requests using the application `voicemail(${EXTEN},b)` for busy messages and `voicemail(${EXTEN},u)` for unanswered messages.

Here is an excerpt of the route section with the changes highlighted:

```
##### Routing Logic #####
# main request routing logic
route{
...
...

    # apply DB based aliases (uncomment to enable)
    alias_db_lookup("dbaliases");

    # Blind call forward
```

```

if(avp_db_load("$ru","$avp(s:callfwd)")) {
    $ru = $avp(s:callfwd);
    #xlog("$avp(s:callfwd)");
    route(1);
    exit;
}
if($rU=~"^\*") {
    # Route to media servers
    route(5);
    route(1);
}

#Dial plan processing
#xlog("$avp(s:country)");
$var(sdpid)=$avp(s:country);
$var(dpid)=$(var(sdpid){s.int});
xlog("$var(dpid)");
if(!dp_translate("$var(dpid)","$ruri.user/$ruri.user")){
    send_reply("420", "Invalid Destination");
}
xlog("$avp(s:dest)");
if ($avp(s:dest)=="usrloc") {
    #Route to usrloc
    route(3);
}
if ($avp(s:dest)=="local") {
    #route to pstn
    $var(v_pr)=$avp(s:country)+$avp(s:area);
    $ru="sip:"+$var(v_pr)+$rU+"@"+$rd;
    xlog("$ru");
    route(4);
}
if ($avp(s:dest)=="ld" || $avp(s:dest)=="int") {
    #route to pstn
    route(4);
}
if ($avp(s:dest)=="media") {
    #route to media server
    route(5);
}
send_reply("420", "Invalid Extension");
exit;
}

```

```
route[1] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {
        t_on_branch("2");
        t_on_reply("2");
        t_on_failure("1");
    }
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

#Route for user lookups
route[3]{
    if (!lookup("location", "m")) {
        switch ($retcode) {
            case -1:
            case -3:
                t_newtran();
                t_reply("404", "Not Found");
                exit;
            case -2:
                sl_send_reply("405", "Method Not Allowed");
                exit;
        }
    }
    # when routing via usrloc, log the missed calls also
    setflag(2);
    route(1);
}

route[4] {
    #---- PSTN route ----#
    if(!do_routing()){
        send_reply("503", "No rules found matching the URI
prefix");
        exit;
    }
    # flag 10 - route to pstn
    setflag(10);
    route(1);
}

route[5] {
```

```

        #---- Route to media servers ----#
        #xlog("route to media servers");
        sethostport("192.168.1.202:5062");
        route(1);
    }

    branch_route[2] {
        xlog("new branch at $ru\n");
    }

    onreply_route[2] {
        xlog("incoming reply\n");
    }

    failure_route[1] {
        if (t_was_cancelled()) {
            exit;
        }

        if(isflagset(10)){
            if (!t_check_status("408|[56][0-9][0-9]")) {
                # this is not a GW failure
                exit;
            }

            if (use_next_gw()) {
                xlog ("next gateway $ru \n");
                t_relay();
                exit;
            } else {
                t_reply("503", "Service not available, no more
gateways");
                exit;
            }
        }
    }

    # Redirect busy calls to a media server
    if (t_check_status("486")) {
        revert_uri();
        #If there is an AVP called fwdbusy send to it
        if(avp_db_load("$ru","$avp(s:fwdbusy)")) {
            $ru = $avp(s:fwdbusy);
            xlog("forwarded to: $avp(s:fwdbusy)");
        } else {
            # If call forward on no answer is not set send to VM
            sethostport("MEDIA_SERVER_IP:MEDIA_SERVER_PORT");
            prefix("u");
        }
    }

```

```
        t_relay();
    }

    # Redirect unanswered calls to the media server
    if (t_check_status("480|408")) {
        revert_uri();
        #If there is an AVP called fwdbusy send to it
        if(avp_db_load("$ru","$avp(s:fwdnoansw)") {
            $ru = $avp(s:fwdbusy);
            xlog("forwarded to: $avp(s:fwdnoansw)");
        } else {
            # If call forward on busy is not set send to VM
            sethostport("MEDIA_SERVER_IP:MEDIA_SERVER_PORT");
            prefix("u");
        }
        t_relay();
    }
}
```

Inspecting the configuration file

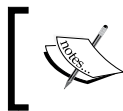
Our script is becoming very hard to debug. Now let's introduce the `xlog` module. It implements the `XLOG()` function. It is similar to the `LOG()` function, but it allows you to use pseudo-variables such as the request-uri (`$ru`) inside the message. Here is an example of the XLOG usage:

```
loadmodule "xlog.so"
xlog("L_ERR","Marker 480 ruri=<$ru>");
```

You can check the latest XLOG messages with the following command:

```
tail /var/log/syslog
```

The `t_on_failure()` function tells OpenSIPS to handle SIP failure (negative/unsuccessful replies) conditions. 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 will tell OpenSIPS to transfer the control to the `failure_route[1]` function when a failure message is detected.



IMPORTANT: The failure route is not executed when you do `t_on_failure`, but sometime in the future, when a SIP failure (if any) is detected.

```
t_on_failure("1");
```

The first part of the `failure_route` section handles cancelled messages (487). The script simply terminates the processing for this kind of a message. In the following code, we will process busy messages.

```
failure_route[1] {
    ##--
    ##-- If cancelled, exit.
    ##--
    if (t_was_cancelled()) {
        exit;
    };
};
```

If the status is equal to 486 (busy here), the action is to revert the `uri` (486 is a failure message in the reverse direction to the INVITE request), prefix the `uri` with `b` using the `prefix()` core function (indicating the voicemail system to play the busy message), and rewrite the host to send the message to the voicemail. The `revert_uri()` command reverts any transformations applied to the original R-URI, before sending it to the media server.

```
# Redirect busy calls to a media server
if (t_check_status("486")) {
    revert_uri();
    sethostport("MEDIA_SERVER_IP:MEDIA_SERVER_PORT");
    prefix("b");
    t_relay();
}

# Redirect unanswered calls to the media server
if (t_check_status("480|408")) {
    revert_uri();
    sethostport("MEDIA_SERVER_IP:MEDIA_SERVER_PORT");
    prefix("u");
    t_relay();
}
```


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 send the busy message, because they have more than a single line. It is important to use all the lines before getting the "486 busy" message. I like to test using two softphones in the same computer—Xlite and Zoiper. When you reject a call in Zoiper, it sends a "486 busy" message to the phone, which is much easier than occupying all of the lines. We are going to 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", 10)
```

Step 1: Test the call forwarding for unanswered calls:

Call from the extension 1000 to the extension 1002. The call should go to the voicemail system with the unavailable message.

Step 2: Test the call forward on busy:

Take the 1003 extension off-hook. Call the 1003 extension from the 1000 extension. It should go to the voicemail system with the busy message.

Summary

In this chapter, we learned how to use AVPs to handle user preferences such as call forward. Using `failure_route` has allowed us to implement three common situations—call forward on busy and call forward on no answer. Finally, we learned how to send these kinds of messages to an external voicemail system like an Asterisk server and how to integrate the databases for simplified administration.

9

SIP NAT Traversal

NAT, also known as **network address translation**, was the solution found to solve the shortage of IP addresses forecasted in mid 90's. The solution consisted of using a small range of IP addresses (in most cases a single IP address) on the outside port of the firewall and a range of invalid addresses (non-registered addresses defined in the RFC1918) on the inside port of the firewall.

Unfortunately, NAT breaks the 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 the SIP communication
- Describe the different types of NAT and their implications
- Describe the main mechanisms available for NAT traversal
- Implement a NAT traversal mechanism called TURN
- Install and configure the RTP Proxy server and the NATHELPER modules

NAT is usually implemented on routers and firewalls. A NAT router maps the internal address to an external address by maintaining an address mapping table. Sometimes NAT is also referred to as **PAT (port address translation)**. PAT maintains a mapping table of `IP:PORT` pairs allowing that a single external address be used to several internal addresses. You can find more information in the RFC1631.

The RFC1918 defines the address allocation for private networks. The private address space can be defined as follows:

- 10.0.0.0 – 0.255.255.255 (10/8 prefix)
- 172.16.0.0 – 172.31.255.255 (172.16/12 prefix)
- 192.168.0.0 – 192.168.255.255 (192.168/16 prefix)

Why NAT breaks SIP

According to Wikipedia, *"In computer networking, network address translation (NAT) is the process of modifying network address information in datagram packet headers while in transit across a traffic routing device for the purpose of remapping a given address space into another"*.

It has been used since the early 90's to overcome IP exhaustion. Since then, it has become a de facto standard for majority of the internet connections and is present in almost any router.

NAT breaks SIP because SIP is a session establishment protocol and thus, belongs to the session layer of the OSI model. However, it includes network layer addresses in their headers. The NAT present in most routers, process only network (layer 3) headers and leaves the SIP headers unchanged.

Where NAT breaks SIP

NAT breaks SIP in the **Contact**, **Via**, **Route**, **Record-Route**, and **SDP** headers. In the following code, the highlighted sections show some layer 3 addresses that appear in an SIP request.

```
U 189.101.207.211:11266 -> 208.109.122.193:5060
INVITE sip:8580@sipmyadmin.com SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.160:11266;branch=z9hG4bK-d8754z-1f2cd50938585f4a-1-
--
d8754z-;rport.
Max-Forwards: 70.
Contact: <sip:flavio@192.168.1.160:11266>.
To: "8580"<sip:8580@sipmyadmin.com>.
From: "flavio"<sip:flavio@sipmyadmin.com>;tag=99494a4b.
Call-ID: NmYwNjAzMDE3MTE0YWM5MWIxNjNiMWNjZDY3NjI0MWQ..
CSeq: 1 INVITE.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE,
INF
O.
Content-Type: application/sdp.
User-Agent: X-Lite release 1103k stamp 53621.
Content-Length: 188.
.
v=0.
o=- 4 2 IN IP4 192.168.1.160.
s=CounterPath X-Lite 3.0.
c=IN IP4 192.168.1.160.
t=0 0.
m=audio 8616 RTP/AVP 0 8 3 101.
a=fmtp:101 0-15.
a=rtpmap:101 telephone-event/8000.
a=sendrecv
```

Without a mechanism to help SIP to traverse NAT, the communication does not happen. Before explaining the mechanisms to traverse NAT, let's explain the existent types of NAT. Some mechanisms will work only for specific types of NAT. There is no silver bullet for the NAT traversal challenge.

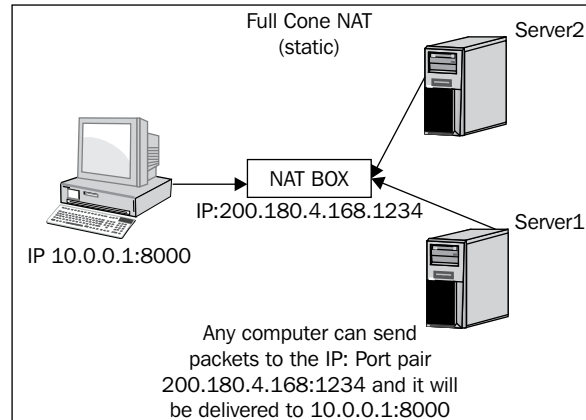
NAT types

There are four kinds of NAT:

- Full cone
- Restricted cone
- Port restricted cone
- Symmetric

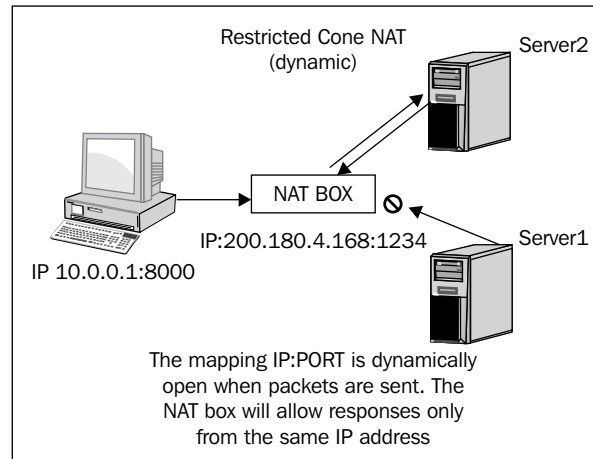
Full cone

The first kind 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.



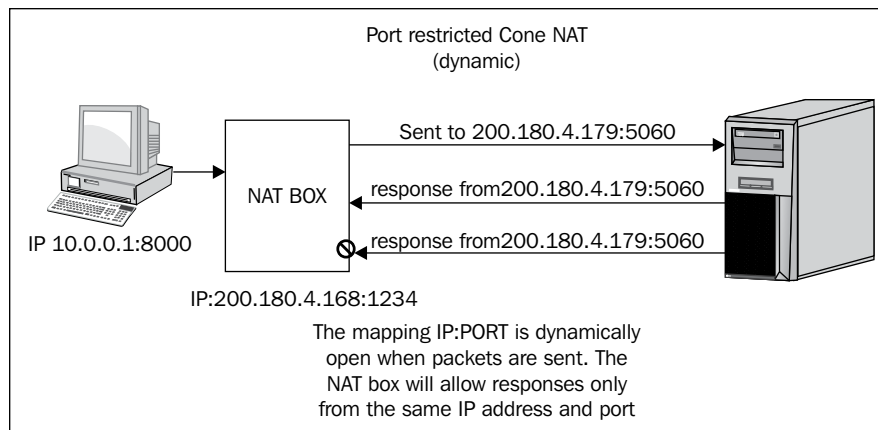
Restricted cone

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 the data back.



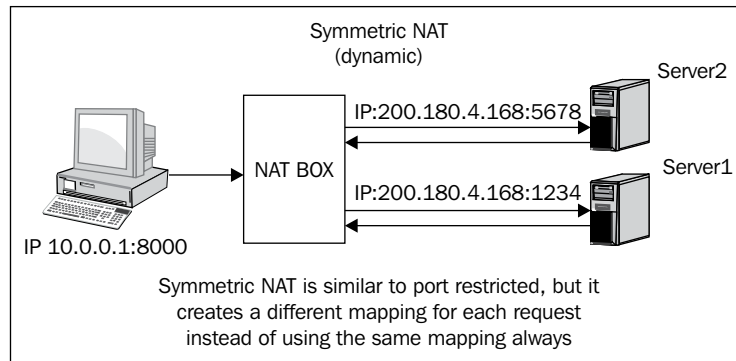
Port restricted cone

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 sent packet.



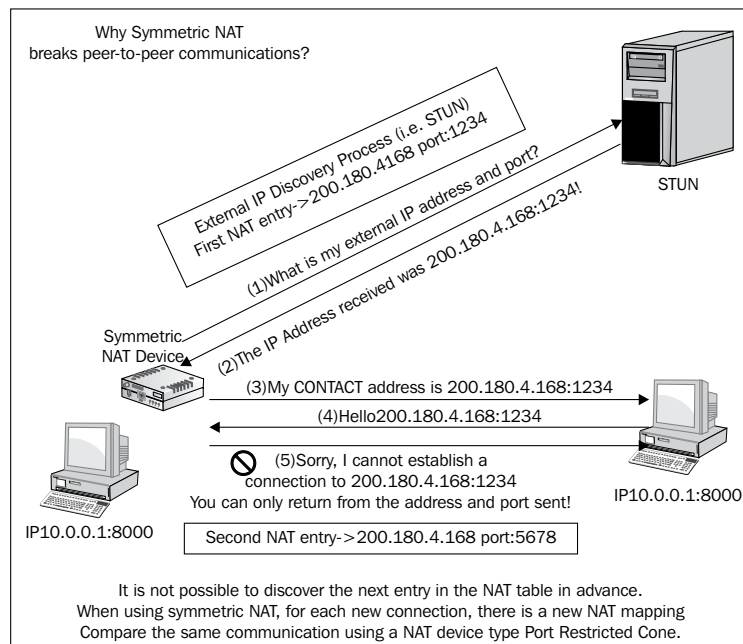
Symmetric

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.



Why symmetric NAT is hard to traverse

With the other three types of NAT, it is possible to use an external server to discover the external IP address used.



However, with symmetric NAT, even if you are able to connect to an external server, the discovered address cannot be used to connect to any other server.

NAT firewall table

The following table is a summary of all types of NAT. This table is useful to help you 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 could be classified as **near-end**, such as **Simple Traversal of UDP through NAT (STUN)**, for solutions implemented on the client-side and **far-end**, such as **Traversal of UDP over Relay NAT (TURN)**, 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, thus forcing the RTP media flow into media relay servers. Near-end solutions are harder to manage and can solve only 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 (symmetric NAT devices don't) and use far-end NAT traversal for the SIP devices behind a symmetric firewall. In this book, we are going to use a far-end NAT solution using a media relay server known as RTP Proxy.

Implementing a near-end NAT solution

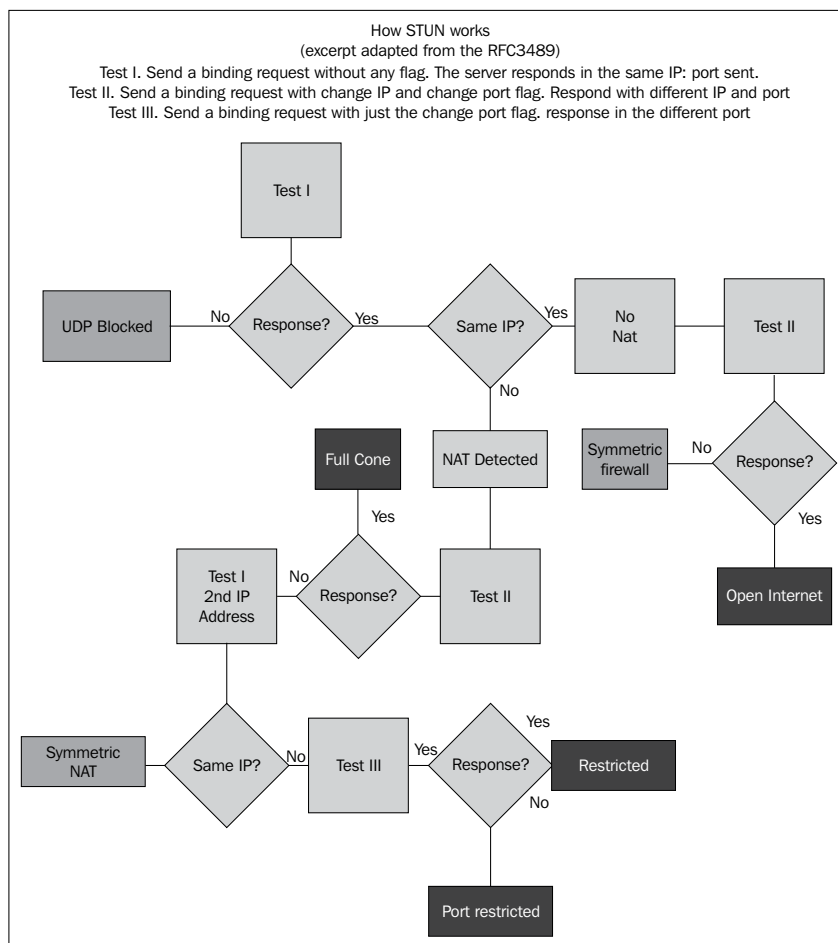
STUN is the most common method for near-end NAT traversal. STUN is based on the RFC3489 and is considered a near-end NAT traversal solution. The biggest advantage of using STUN is that the client appears to be in the Open Internet to the proxy. You don't require any configuration on the server for NAT traversal. The biggest disadvantage is that it does not work with symmetric NAT devices.

The protocol STUN permits that IP endpoints behind NAT device discover their external IP address and ports. With this information, the device can inform another party what address it can be contacted at.

It 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
- If it is behind a NAT device, what kind of device it is

In the following figure, all the results in dark boxes can be handled by STUN. The other situations can only be traversed with the media relay solution (TURN).



STUN is now a module in version 1.6. You can implement STUN using OpenSIPS in the same port as the proxy server. You will need two public IP addresses for this server. To implement STUN, you need to load the STUN module and set some specific parameters. The biggest advantage for STUN running in the same port as the SIP proxy is that it can handle the SIP signaling even for symmetric NATs. See the following configuration example:

```
listen=udp:IP1:5060
listen=udp:IP2:5060
loadmodule "stun.so"
# ---- Stun ----
modparam("stun","primary_ip","IP1")
modparam("stun","primary_port","5060")
modparam("stun","alternate_ip","IP2")
modparam("stun","alternate_port","5060")
```

The implementation of STUN in OpenSIPS allows you to run the STUN service on the same port as the SIP proxy service. Two IP addresses are mandatory and both need to be bound to OpenSIPS.

It is possible to test STUN without configuring 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 at the client-side, it won't be necessary to make any changes in the `opensips.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 to create a new mapping for each external device contacted. Therefore, if you contact the STUN device, it will inform you about the IP:PORT pair from whom 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 device (full cone, cone restricted, and port restricted), the mapping created for one device would be exactly the same as the one created to other devices, as the internal IP:PORT was the same.

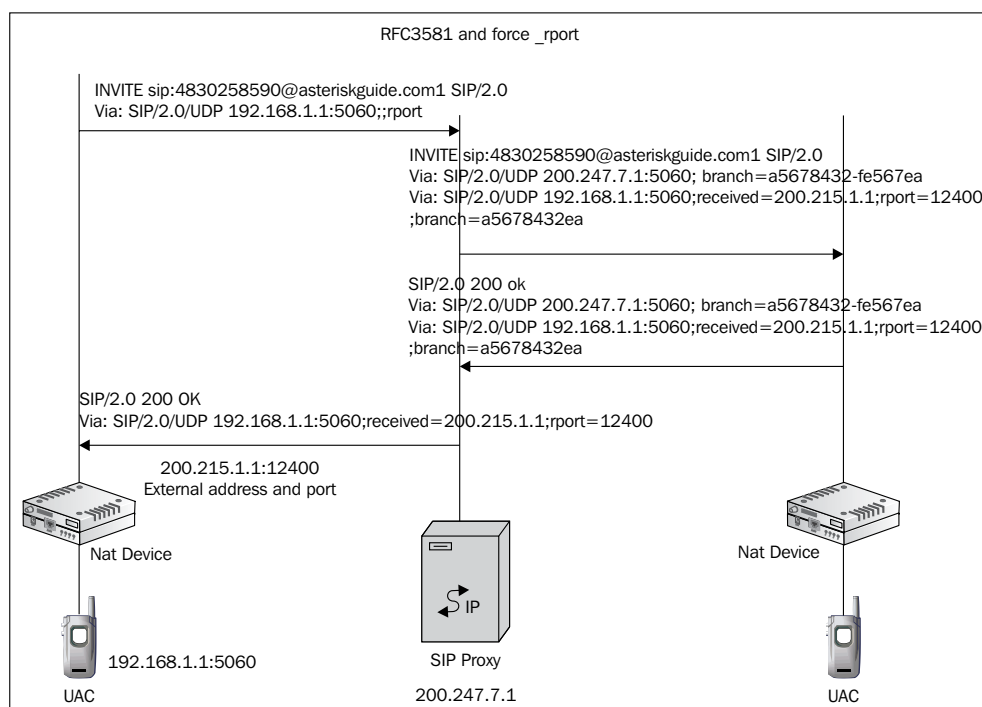
Implementing a far-end NAT solution

Now, let's examine a solution to implement NAT traversal on the servers, without having to make any special configuration changes to the clients. The UAC has to be symmetric, in other words, it should send and receive on the same UDP port for both SIP (5060) and RTP (usually, in the range of 10,000 to 20,000). As of today, most UACs are symmetric, so don't worry about this.

The SIP NAT traversal problem can be classified into two categories. 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 protocol. We will use the RFC3581 to traverse the SIP packets, together with some message treatment and a media relay server known as RTP Proxy, developed by Maxim Sobolev (www.rtpproxy.org) and released under a GPL license. This kind of solution is also known by the acronym TURN. There is another TURN solution for OpenSIPS known as **MediaProxy**.

The RFC3581 and the force_rport() function

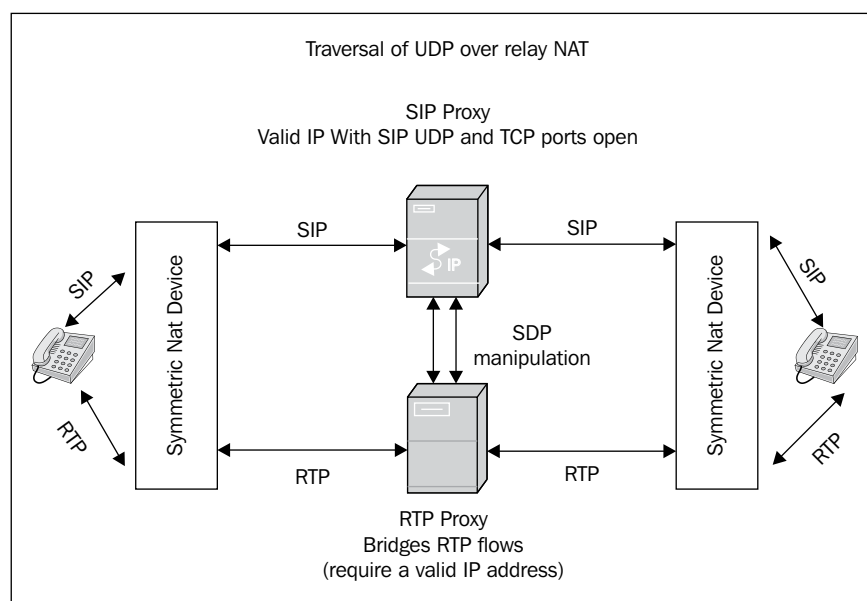
In the following figure, you can see the process described in RFC3581 to solve the NAT traversal for the SIP signaling. The traversal of the SIP signaling can be easily solved using the fields named **rport** and **received** as stated in the RFC.



When a client is compatible with the RFC3581, it inserts a parameter named **rport** in the **Via** field of the SIP request. The proxy will verify the existence of the field and will include the field **received=** and **rport=** with the address from the interface where it received the packet. This makes it 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. The RFC3581 solves the problem with dialing, but not the reception of the calls. To receive calls we are going to tweak the registering process to include the external IP address. You will see this when we explain the script.

Solving the traversal of the RTP packets

The SIP problem was solved with the above solutions. This was possible because the SIP protocol uses a single and previously known port (UDP 5060). On the other hand, the RTP uses dynamic UDP ports described in the **Session Description Protocol (SDP)**. When using symmetric NAT device, a new mapping will be created to each destination. Therefore, it is not possible to inform the UAC about the right UDP port from whom it received the RTP packet to send the response. The solution is to send both RTP sessions directly to a device called Media Proxy with a known IP address and port, and bridge the RTP flow in this box.



When you use an RTP Proxy, 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 the RTP Proxy using the function `force_rtp_proxy()` from the NATHelper module.

I have chosen the RTP Proxy software for this book because it is fast, stable, and easy to use. It has flags that allow the handling of some special cases such as bridging two different network interfaces. Recently, new features were added to RTP Proxy such as call recording and streaming.

RTP Proxy installation and configuration

The RTP Proxy is easy to install. You will have to follow a few steps to install and run the RTP Proxy.

Step 1: Download and install the RTP Proxy:

```
cd /usr/src
wget http://b2bua.org/chrome/site/rtpproxy-1.2.0.tar.gz
tar -xzf rtpproxy-1.2.0.tar.gz
cd rtpproxy-1.2.0
./configure
make
make install
```

Step 2: Start the RTP Proxy:

```
./rtpproxy -l theipaddressofyourserver -s udp:127.0.0.1:7890 -F
```

Step 3: Configure the NATHELPER module to communicate with the RTP Proxy.

```
modparam("nathelper", "rtpproxy_sock", "udp:127.0.0.1:7890")
```

Analysis of the file opensips.cfg

Let's analyze some of the changes to the configuration file created to support NAT traversal.

Modules loading

NATHELPER and RTP Proxy are responsible for handling the NAT traversal. Some functions such as `fix_nated_contact()`, `fix_nated_register()`, and `nat_uac_test()` are made available by the NATHELPER module.

```
loadmodule "/usr/lib/openser/modules/nathelper.so"
```

Modules parameters

The module `nathelper` has some important parameters. Some will be explained in the later sections of this chapter.

```
modparam("nathelper", "rtpproxy_sock", "udp:127.0.0.1:7890")
modparam("nathelper", "natping_interval", 30)
modparam("nathelper", "ping_nated_only", 0)
modparam("nathelper", "sipping_bflag", 7)
modparam("nathelper", "sipping_from", "sip:pinger@PROXY_IP")
```

These parameters play an important role in the NAT traversal process. The first parameter `rtpproxy_sock` establishes the socket where the daemon `rtpproxy` and OpenSIPS will communicate. If you want to load balance several instances of `rtpproxy`, then you can specify several addresses separated by commas. This is recommended if you are using a multiprocessor computer and want to improve the use of the hardware available.

The parameters `natping_interval`, `ping_nated_only`, `sipping_bflag`, and `sipping_from`, controls how to ping the clients to maintain the NAT mapping open. Most of these parameters are self explanatory, but pay attention to the `sipping_bflag`. If the `bflag` is set to 7, the NATHELPER module will ping using an OPTIONS SIP request from the address established in the parameter `sipping_from`. It is much safer to use the OPTIONS request instead of a dummy UDP packet, because some NAT implementations require an outbound packet to keep the NAT mapping open and the client will answer the OPTIONS request with any SIP response, which will do the trick.

Determining if the client is behind NAT

We are going to use the function `nat_uac_test()` to determine early if the client is behind NAT. If the client is actually behind NAT, we are going to mark this request with the flag number 5 for future use. See the following code snippet:

```
modparam("registrar","received_avp", "$avp(i:42)")
modparam("nathelper","received_avp", "$avp(i:42)")

## NAT Detection
#
force_rport();
if (nat_uac_test("19")) {
```

```

    if (method=="REGISTER") is_method() {
        fix_nated_register();
    }
    else {
        fix_nated_contact();
    };
    setflag(5);
};

```

In the last code snippet, we force the `rport` for all requests, because it is harmless and will help us to determine the correct IP address of the request. After testing the request using the function `nat_uac_test()` (see the table that follows), we will issue the command `fix_nated_register()` if the method is `REGISTER`, or `fix_nated_contact()` if any other method. The function `fix_nated_register()` will create a new URI with source IP + port + protocol in an AVP (in our case, `received_avp, i:42`). The URI will be appended with a `received` parameter in the `CONTACT` header field and stored in the user location table. To check if this function is working properly, verify if you have the external address of your client stored in the user location table. The function `fix_nated_contact()` simply rewrites the `CONTACT` header field to contain the request's source IP:PORT.

Parameter	Tests applied by the function <code>nat_uac_test()</code>
1	Tests if the client has an RFC1918 address in the <code>CONTACT</code> header field.
2	Tests if the client has contacted OpenSIPS from an address that is different from the one in the <code>Via</code> field.
4	Tests if the client has an RFC1918 address in the topmost <code>Via</code> header.
8	SDP is searched for RFC1918 addresses.
16	Tests if the source port is different from the port in the <code>Via</code> 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 REGISTER requests behind NAT

It is easy to handle register requests behind NAT. In the first place, you have to set the parameter `nat_bflag` to some number before saving the location of the UAC. The flag 6 will be saved to the user location table and will be recovered when you use the function `lookup("location")`. This will help you to determine if the destination is behind NAT.

```
modparam("usrloc","nat_bflag", 6)

if (isflagset(5)) {
    setbflag(6);
    setbflag(7);
};
if (!save("location")) sl_reply_error();
```

Handling INVITE messages behind NAT

On the 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 protocol. To accomplish this, we will have to make modifications to the SIP and the SDP headers.

Usually, when a request is behind NAT, the contact information is wrong, it points to the private (RFC1918) address. The OpenSIPS 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 fixed too. This was done at the beginning of the script.

```
route[1] {
    if (subst_uri('/(sip:.*);nat=yes/\1/')) {
        setbflag(6);
    };

    if (isflagset(5) || isbflagset(6)) {
        route(6);
    };
    if (isflagset(5)) {
        search_append('Contact:.*sip:[^>[:cntrl:]]*', ';nat=yes');
    }
    route[6] {
        if (is_method("BYE|CANCEL")) {
```

```

    unforce_rtp_proxy();
}
else if (is_method("INVITE")){
    force_rtp_proxy();
    t_on_failure("1");
};
}

```

Now we need to handle the media. When an INVITE message is sent it will contain an SDP payload. This SDP header identifies the session content (audio, video, chat, and named events). The SDP payload describes several things about the UAC, such as type of session it supports, IP address, and UDP port where the other part can be found. When we detect a caller behind NAT (flag 5) or a callee behind NAT (flag 6), we will instruct the script to call the `route(6)`. In this route, we will force the RTP Proxy for INVITE requests using the function `force_rtp_proxy()`. For BYE and CANCEL requests, we will turn off the RTP Proxy session freeing valuable resources using `unforce_rtp_proxy()`.

For 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. The SDP lines describing the IP address and UDP port are shown as follows:

```

c=IN IP4 192.168.0.1.
m=audio 23767 RTP/AVP 0 101.

```

To handle the audio sessions, OpenSIPS will do one thing before forwarding the INVITE to the final user. Force RTP to pass over the RTP Proxy changing the line `c` to `c=<ip-address-of-the-RTP-proxy> RTP/AVP 0 101.`

This option means that you need to configure an RTP Proxy with a public IP address, on which both users can send the RTP traffic adding an additional hop to the RTP connection. This additional hop will have implications on the delay and possibly jitter. However, it is the only way to traverse the symmetrical NAT.

Handling the responses

The "200 OK" message returned from the UAC will also need to be manipulated. Thus a NAT handling code must be included in the section `on_reply_route[]`.

```

onreply_route[2] {
    if ((isflagset(5) || isbflagset(6)) && status=~"(183)|(2[0-9][0-9])"){
        force_rtp_proxy();
        append_hf("P-hint: onreply_route|force_rtp_proxy \r\n");
    }
}

```



```
}

#---- If the CALLEE is behind NAT, fix the CONTACT HF ----#
if (isbflagset(6)) {
    #-- Insert nat=yes at the end of the Contact header --#
    #-- This helps with REINVITES, --#
    #- nat=yes will be included in the R-URI for seq.requests-#
    search_append('Contact:.*sip:[^>[:cntrl:]]*', ';nat=yes');
    append_hf("P-hint: Onreply-route - fixcontact \r\n");
    fix_nated_contact();
}
exit;
}
```

If the flag 5 or 6 is set, in other words, if the caller or callee is behind NAT respectively, and the status of the request is 183 or 2XX, force the use of the RTP Proxy. In the request (`route[6]`), you have changed the SDP to point the IP address of the RTP Proxy to the callee. Now, on the reply (`on_reply_route[2]`), you are pointing the IP address of the RTP Proxy to the caller.

Handling RE-INVITE messages

Handling RE-INVITES is a two step process. In the first step, we append the string `nat=yes` in the CONTACT header field. This string will be added to the routing set of the caller UAC.

```
#--- Insert nat=yes at the end of the Contact header ----#
#----- This helps with REINVITES, ----#
#- nat=yes will be included in the R-URI for sequential requests -#
search_append('Contact:.*sip:[^>[:cntrl:]]*', ';nat=yes');
```

Sequential requests are handled at the beginning of the script in the "Sequential requests section" as shown in the following code snippet. In this section, we use `route[1]` where we check the presence of the string `nat=yes` in the r-uri. If found, it indicates that this request is behind NAT and should be handled properly.

```
#----- This is used to Process REINVITES ----#
if (subst_uri('/(sip:.*);nat=yes/\1/')){
    setbflag(6);
};
```

Routing script

The following is the complete route script. The general and module sections were omitted. The sections highlighted are the ones required for NAT traversal.

```

route{
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
    }
    #---- NAT Detection ----#
    force_rport();
    if (nat_uac_test("18")) {
        if (method=="REGISTER")is_() {
            fix_nated_register();
        }
        else {
            fix_nated_contact();
        }
    }
    setflag(5);
}

#---- Sequential requests section ----#
if (has_totag()) {
    # sequential request withing a dialog should
    # take the path determined by record-routing
    if (loose_route()) {
        if (is_method("BYE")) {
            setflag(1); # do accounting
            setflag(3); # even if the transaction fails
        }
        else if (is_method("INVITE")) {
            record_route();
        }
        route(1);
    }
    else {
        if ( is_method("ACK") ) {
            if ( t_check_trans() ) {
                t_relay();
                exit;
            }
        }
        else {
            exit;
        }
    }
}

```

```
    }
    sl_send_reply("404","Not here");
  }
  exit;
}

    #---- initial requests section ----#
if (is_method("CANCEL")) {
  if (t_check_trans()) {
    t_relay();
    exit;
  }
  t_check_trans();
  if (!(method=="REGISTER")is_() && is_from_local()) {
    if(!allow_trusted()){
      if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "0");
        exit;
      }
      if (!db_check_from()) {
        sl_send_reply("403","Forbidden auth ID");
        exit;
      }
      consume_credentials();
      # caller authenticated
    }
  }
}

#---- preloaded route checking ----#
if (loose_route()) {
  xlog("L_ERR","Attempt to route with preloaded Route's
    [$fu/$tu/$ru/$ci]");
  if (!is_method("ACK")) {
    sl_send_reply("403","Preload Route denied");
    exit;
  }
}

#---- record routing ----#
if (!is_method("REGISTER|MESSAGE"))
  record_route();
# account only INVITEs
if (is_method("INVITE")) {
  setflag(1); # do accounting
}
```

```

#---- Routing to external domains ----#
if (!is_uri_host_local())
{
    append_hf("P-hint: outbound\r\n");
    if(is_uri_host_local()) {
        route(1);
    }
    else {
        sl_send_reply("403", "Not here");
    }
}
if (is_method("PUBLISH"))
{
    sl_send_reply("503", "Service Unavailable");
    exit;
}
if (is_method("REGISTER"))
{
    # authenticate the REGISTER requests (uncomment to enable auth)
    if (!www_authorize("", "subscriber"))
    {
        www_challenge("", "0");
        exit;
    }
    if (!db_check_to())
    {
        sl_send_reply("403", "Forbidden auth ID");
        exit;
    }
}

#-- Request is behind NAT(flag5) save with bflag 6 -#
#---- Use bflag 7 to start SIP ping (Options) ---#
if (isflagset(5)) {
    setbflag(6);
    setbflag(7);
};

if (!save("location")) {
    sl_reply_error();
    exit;
}
if ($rU==NULL) {
    # request with no Username in RURI
    sl_send_reply("484", "Address Incomplete");
}

```

```
        exit;
    }

    # apply DB based aliases (uncomment to enable)
    alias_db_lookup("dbaliases");
    # Blind call forward
    if(avp_db_load("$ru","$avp(s:callfwd)") ) {
        $ru = $avp(s:callfwd);
        #xlog("$avp(s:callfwd)");
        route(1);
        exit;
    }
    if($rU=~"^\\*") {
        # Route to media servers
        route(5);
        route(1);
    }
    #Dial plan processing
    #xlog("$avp(s:country)");
    $var(sdpid)=$avp(s:country);
    $var(dpidd)=$(var(sdpid){s.int});
    xlog("$var(dpidd)");
    if(!dp_translate("$var(dpidd)","$ruri.user/$ruri.user")){
        send_reply("420", "Invalid Destination");
        exit;
    }
    xlog("$avp(s:dest)");
    if ($avp(s:dest)=="usrloc") {
        #Route to usrloc
        route(3);
    }
    if ($avp(s:dest)=="local") {
        #route to pstn
        $var(v_pr)=$avp(s:country)+$avp(s:area);
        $ru="sip:"+$var(v_pr)+$rU+"@"+$rd;
        xlog("$ru");
        route(4);
    }
    if ($avp(s:dest)=="ld" || $avp(s:dest)=="int") {
        #route to pstn
        route(4);
    }
    if ($avp(s:dest)=="media") {
        #route to media server
        route(5);
    }
    send_reply("420", "Invalid Extension");
```

```

    exit;
}
route[1] {
    # for INVITES enable some additional helper routes
#---- Helper route, if nat=yes in the R-URI set flag 6 ----#
#---- This is used to Process REINVITES ----#
if (subst_uri('/(sip:.*);nat=yes/\1/')){
    setbflag(6);
};
#-- If caller(flag 5) or callee(flag 6) are behind NAT --#
#-- Call the route(6) to force the use of the RTP Proxy --#
if (isflagset(5) || isbflagset(6)) {
    route(6);
};
if (isflagset(5)){
    search_append('Contact:.*sip:[^>[:cntrl:]]*', ';nat=yes');
}
if (is_method("INVITE")) {
    t_on_branch("2");
    t_on_reply("2");
    t_on_failure("1");
}
if (!t_relay()) {
    sl_reply_error();
};
exit;
}
#Route for user lookups
route[3]{
    if (!lookup("location", "m")) {
        switch ($retcode) {
            case -1:
            case -3:
                t_newtran();
                t_reply("404", "Not Found");
                exit;
            case -2:
                sl_send_reply("405", "Method Not Allowed");
                exit;
        }
    }
}
# when routing via usrloc, log the missed calls also
setflag(2);
route(1);
}

route[4] {

```

```
#---- PSTN route ----#
if(!do_routing()){
    send_reply("503", "No rules found matching the URI prefix");
    exit;
}
#--- mark the transaction with flag 10 - route to pstn
setflag(10);
route(1);
}

route[5] {
    #---- Route to media servers ----#
    #xlog("route to media servers");
    rewritehostport("192.168.1.202:5062");
    route(1);
}

route[6] {
    #---- RTP Proxy handling ---#
    if (is_method("BYE|CANCEL")) {
        unforce_rtp_proxy();
    }
    else if (is_method("INVITE")){
        #---- Activates the RTP Proxy for the CALLEE ---#
        force_rtp_proxy();
        t_on_failure("1");
    };
}

branch_route[2] {
    xlog("new branch at $ru\n");
}

onreply_route[2] {
    #xlog("incoming reply\n");
    #---- Handling of the SDP for the 200 or 183 reply ----#
    #---- If behind nat (flags 5 or 6) start RTP Proxy ----#
    #---- Activates the RTP Proxy for the CALLER ----#
    if ((isflagset(5) || isbflagset(6)) && status=~"(183)|(2[0-9][0-9])"){
        force_rtp_proxy();
        append_hf("P-hint: onreply_route|force_rtp_proxy \r\n");
    }

    #---- If the CALLEE is behind NAT, fix the CONTACT HF ----#
    if (isbflagset(6)) {
        #-- Insert nat=yes at the end of the Contact header --#
        #-- This helps with REINVITES, --#
    }
}
```

```

#- nat=yes will be included in the R-URI for seq.requests-#
search_append('Contact:.*sip:[^>[:cntrl:]]*', ';nat=yes');
append_hf("P-hint: Onreply-route - fixcontact \r\n");
fix_nated_contact();
}
exit;
}

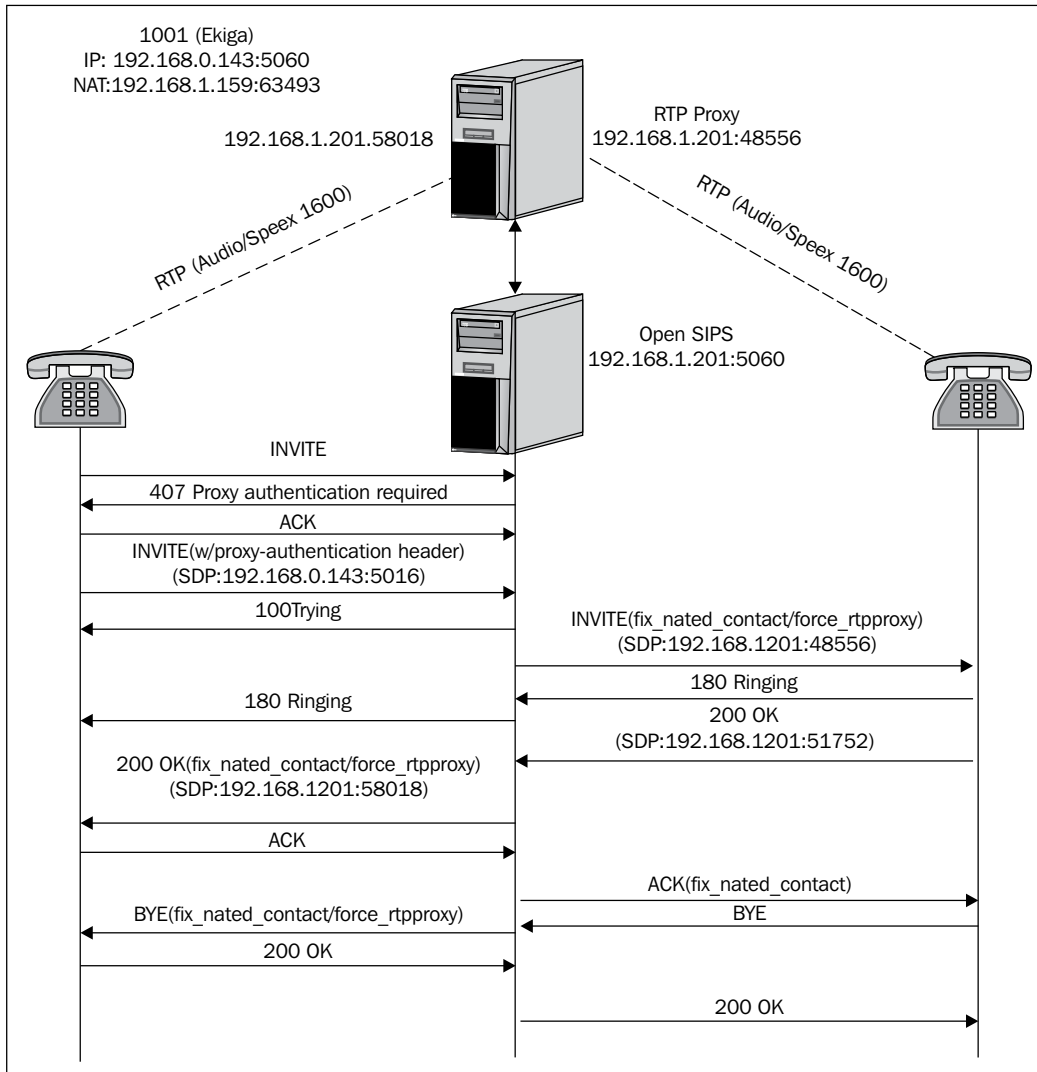
failure_route[1] {

    #---- If a failure has occurred, deactivate the RTP Proxy ----#
    if (isflagset(5) || isbflagset(6)){
        unforce_rtp_proxy();
    }
    if (t_was_cancelled()) {
        exit;
    }
    #- if the failure comes from a PSTN route, handle properly -#
    if(isflagset(10)){
        if (use_next_gw()) {
            xlog ("next gateway $ru \n");
            route(1);
            exit;
        }
        else {
            t_reply("503", "Service not available, no more gws");
            exit;
        }
    }
    # Redirect busy calls to a media server
    if (t_check_status("486")) {
        revert_uri();
        sethostport("192.168.1.202:5062");
        prefix("b");
        t_relay();
    }
    # Redirect unanswered calls to the media server
    if (t_check_status("480|408")) {
        revert_uri;
        sethostport("192.168.1.202:5062");
        prefix("u");
        t_relay();
    }
}
}

```


Invite diagram

The following figure shows us how the packets are handled. We've included some hints in the requests to help you to trace the call in the script.



Packet sequence

The following is the INVITE sequence with the mangled requests highlighted. Pay careful attention to the modified SDP after the relay of the INVITE request and the "200 OK" reply. Some requests were reduced to save space.

```

U 192.168.1.159:63493 -> 192.168.1.201:5060
INVITE sip:1000@192.168.1.201 SIP/2.0.
Date: Tue, 15 Sep 2009 16:44:08 GMT.
CSeq: 1 INVITE.
Via: SIP/2.0/UDP 192.168.0.143:5067;branch=z9hG4bKb224e5a7-84a0-dell-
8a33-000c29254d97;rport.
User-Agent: Ekiga/2.0.12.
From: "flavio goncalves" <sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-
dell-8a33-000c29254d97.
Call-ID: 1894e2a7-84a0-dell-8a33-000c29254d97@debian.
To: <sip:1000@192.168.1.201>.
Contact: <sip:1001@192.168.0.143:5061;transport=udp>.
Allow: INVITE,ACK,OPTIONS,BYE,CANCEL,NOTIFY,REFER,MESSAGE.
Content-Type: application/sdp.
Content-Length: 389.
Max-Forwards: 70.
v=0.
o=- 1253033048 1253033048 IN IP4 192.168.0.143.
s=Opal SIP Session.
c=IN IP4 192.168.0.143.
t=0 0.
m=audio 5016 RTP/AVP 96 3 107 110 0 8 101.
a=rtpmap:96 SPEEX/16000.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.
m=video 5018 RTP/AVP 31.
a=rtpmap:31 H261/90000.

U 192.168.1.201:5060 -> 192.168.1.159:63493
SIP/2.0 407 Proxy Authentication Required.

U 192.168.1.159:63493 -> 192.168.1.201:5060
ACK sip:1000@192.168.1.201 SIP/2.0.

U 192.168.1.159:63493 -> 192.168.1.201:5060
INVITE sip:1000@192.168.1.201 SIP/2.0.
Date: Tue, 15 Sep 2009 16:44:08 GMT.
CSeq: 2 INVITE.

```

Via: SIP/2.0/UDP 192.168.0.143:5067;branch=z9hG4bK64d3e8a7-84a0-de11-8a33-000c29254d97;rport.
User-Agent: Ekiga/2.0.12.
From: "flavio goncalves" <sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-de11-8a33-000c29254d97.
Call-ID: 1894e2a7-84a0-de11-8a33-000c29254d97@debian.
To: <sip:1000@192.168.1.201>.
Contact: <sip:1001@192.168.0.143:5061;transport=udp>.
Proxy-Authorization: Digest username="1001", realm="192.168.1.201", nonce="4aa2b95800000038e0678aab2f10d7305e63dc12bbb0fda3", uri="sip:1000@192.168.1.201", algorithm=md5, response="be3521103f357487cb42a51cf89d1ebf".
Allow: INVITE,ACK,OPTIONS,BYE,CANCEL,NOTIFY,REFER,MESSAGE.
Content-Type: application/sdp.
Content-Length: 389.
Max-Forwards: 70.
.
v=0.
o=- 1253033048 1253033048 IN IP4 192.168.0.143.
s=Opal SIP Session.
c=IN IP4 192.168.0.143.
t=0 0.
m=audio 5016 RTP/AVP 96 3 107 110 0 8 101.
a=rtpmap:96 SPEEX/16000.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.
m=video 5018 RTP/AVP 31.
a=rtpmap:31 H261/90000.

U 192.168.1.201:5060 -> 192.168.1.159:63493
SIP/2.0 100 Giving a try.

U 192.168.1.201:5060 -> 192.168.1.159:4106

INVITE sip:1000@192.168.1.159:4106;rinstance=d22dcb0534217188 SIP/2.0.
Record-Route: <sip:192.168.1.201;lr=on>.
Date: Tue, 15 Sep 2009 16:44:08 GMT.
CSeq: 2 INVITE.
Via: SIP/2.0/UDP 192.168.1.201;branch=z9hG4bKcf89.06472571.1.
Via: SIP/2.0/UDP 192.168.0.143:5067;received=192.168.1.159;branch=z9hG4bK64d3e8a7-84a0-de11-8a33-000c29254d97;rport=63493.
User-Agent: Ekiga/2.0.12.
From: "flavio goncalves" <sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-de11-8a33-000c29254d97.
Call-ID: 1894e2a7-84a0-de11-8a33-000c29254d97@debian.
To: <sip:1000@192.168.1.201>.

```

Contact: <sip:1001@192.168.1.159:63493;transport=udp>.
Allow: INVITE,ACK,OPTIONS,BYE,CANCEL,NOTIFY,REFER,MESSAGE.
Content-Type: application/sdp.
Content-Length: 409.
Max-Forwards: 69.
P-Hint:fix_nated_contact applied.
P-Hint:applied force_rtp_proxy/INVITE.
v=0.
o=- 1253033048 1253033048 IN IP4 192.168.0.143.
s=Opal SIP Session.
c=IN IP4 192.168.1.201.
t=0 0.
m=audio 48556 RTP/AVP 96 3 107 110 0 8 101.
a=rtpmap:96 SPEEX/16000.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.
m=video 38902 RTP/AVP 31.
a=rtpmap:31 H261/90000.
a=nortpproxy:yes.

U 192.168.1.159:4106 -> 192.168.1.201:5060
SIP/2.0 180 Ringing.

U 192.168.1.159:4106 -> 192.168.1.201:5060
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 192.168.1.201;branch=z9hG4bKcf89.06472571.1.
Via: SIP/2.0/UDP 192.168.0.143:5067;received=192.168.1.159;branch=z9hG
4bK64d3e8a7-84a0-de11-8a33-000c29254d97;rport=63493.
Record-Route: <sip:192.168.1.201;lr>.
Contact: <sip:1000@192.168.1.159:4106;rinstance=d22dcb0534217188>.
To: <sip:1000@192.168.1.201>;tag=96694179.
From: "flavio goncalves"<sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-
de11-8a33-000c29254d97.
Call-ID: 1894e2a7-84a0-de11-8a33-000c29254d97@debian.
CSeq: 2 INVITE.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO.
Content-Type: application/sdp.
User-Agent: X-Lite release 1011s stamp 41150.
Content-Length: 237.
.
v=0.
o=- 3 2 IN IP4 192.168.1.159.
s=CounterPath X-Lite 3.0.

```

```
c=IN IP4 192.168.1.159.
t=0 0.
m=audio 51752 RTP/AVP 96 0 8 101.
a=fmtp:101 0-15.
a=rtpmap:96 SPEEX/16000.
a=rtpmap:101 telephone-event/8000.
a=sendrecv.
m=video 0 RTP/AVP 34.

U 192.168.1.201:5060 -> 192.168.1.159:63493
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 192.168.0.143:5067;received=192.168.1.159;branch=z9hG
4bK64d3e8a7-84a0-de11-8a33-000c29254d97;rport=63493.
Record-Route: <sip:192.168.1.201;lr>.
Contact: <sip:1000@192.168.1.159:4106;rinstance=d22dcb0534217188;nat=
yes>.
To: <sip:1000@192.168.1.201>;tag=96694179.
From: "flavio goncalves"<sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-
de11-8a33-000c29254d97.
Call-ID: 1894e2a7-84a0-de11-8a33-000c29254d97@debian.
CSeq: 2 INVITE.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO.
Content-Type: application/sdp.
User-Agent: X-Lite release 1011s stamp 41150.
Content-Length: 255.
P-hint:applied force_rtp_proxy/2000K.
P-hint:applied fix_nated_contact .
.
v=0.
o=- 3 2 IN IP4 192.168.1.159.
s=CounterPath X-Lite 3.0.
c=IN IP4 192.168.1.201.
t=0 0.
m=audio 58018 RTP/AVP 96 0 8 101.
a=fmtp:101 0-15.
a=rtpmap:96 SPEEX/16000.
a=rtpmap:101 telephone-event/8000.
a=sendrecv.
m=video 0 RTP/AVP 34.
a=nortpproxy:yes.

U 192.168.1.159:63493 -> 192.168.1.201:5060
```

ACK sip:1000@192.168.1.159:4106;nat=yes;rinstance=d22dcb0534217188
SIP/2.0.
Route: <sip:192.168.1.201;lr>.

U 192.168.1.201:5060 -> 192.168.1.159:4106

ACK sip:1000@192.168.1.159:4106;rinstance=d22dcb0534217188 SIP/2.0.
CSeq: 2 ACK.
Via: SIP/2.0/UDP 192.168.1.201;branch=z9hG4bKcf89.06472571.3.
Via: SIP/2.0/UDP 192.168.0.143:5067;received=192.168.1.159;branch=z9hG4bK94ce60aa-84a0-de11-8a33-000c29254d97;rport=63493.
From: "flavio goncalves" <sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-de11-8a33-000c29254d97.
Call-ID: 1894e2a7-84a0-de11-8a33-000c29254d97@debian.
To: <sip:1000@192.168.1.201>;tag=96694179.
Contact: <sip:1001@192.168.1.159:63493;transport=udp>.
Proxy-Authorization: Digest username="1001", realm="192.168.1.201", nonce="4aa2b95800000038e0678aab2f10d7305e63dc12bbb0fda3", uri="sip:1000@192.168.1.159:4106", algorithm=md5, response="88d4283b711982d79a9038eac6daf87e".
Allow: INVITE,ACK,OPTIONS,BYE,CANCEL,NOTIFY,REFER,MESSAGE.
Content-Length: 0.
Max-Forwards: 69.
P-Hint:fix_nated_contact applied.
P-Hint:detected reinvoke behind NAT.

U 192.168.1.159:4106 -> 192.168.1.201:5060
BYE sip:1001@192.168.1.159:63493;transport=udp SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.159:4106;branch=z9hG4bK-d87543-

U 192.168.1.201:5060 -> 192.168.1.159:63493

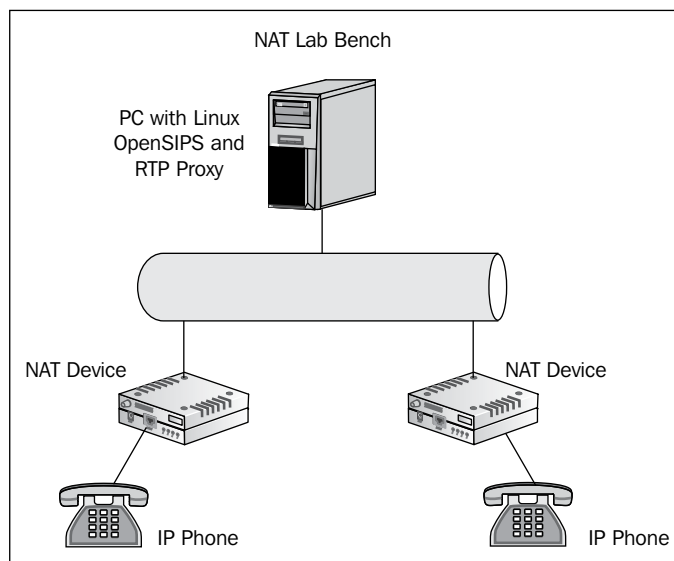
BYE sip:1001@192.168.1.159:63493;transport=udp SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.201;branch=z9hG4bKcf89.26472571.0.
Via: SIP/2.0/UDP 192.168.1.159:4106;received=192.168.1.159;branch=z9hG4bK-d87543-fc00b82bea322863-1--d87543-;rport=4106.
Max-Forwards: 69.
Contact: <sip:1000@192.168.1.159:4106;rinstance=d22dcb0534217188>.
To: "flavio goncalves"<sip:1001@192.168.1.201>;tag=8a9ae2a7-84a0-de11-8a33-000c29254d97.
From: <sip:1000@192.168.1.201>;tag=96694179.
Call-ID: 1894e2a7-84a0-de11-8a33-000c29254d97@debian.
CSeq: 2 BYE.
User-Agent: X-Lite release 1011s stamp 41150.
Reason: SIP;description="User Hung Up".
Content-Length: 0.

P-Hint:fix_nated_contact applied.

```
U 192.168.1.159:63493 -> 192.168.1.201:5060
SIP/2.0 200 OK.
CSeq: 2 BYE.
U 192.168.1.201:5060 -> 192.168.1.159:4106
SIP/2.0 200 OK.
CSeq: 2 BYE.
```

Lab—using the RTP Proxy for NAT traversal

Testing 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: Download and compile RTP Proxy with the instructions provided earlier in this chapter.

Step 2: Start your RTP Proxy.

```
./rtpproxy -l theipaddressofyourserver -s udp:127.0.0.1:7890 -F
```

Step 3: Start making calls from the phones behind the NAT devices. Use `ngrep` to capture the packets and troubleshoot any problems.

Comparing STUN with TURN (MRS)

STUN allows for better scalability and the endpoints can communicate directly. With **Media Relay Server (MRS)**, if a UAC wants to communicate to another UAC, they will have to use your server to relay the RTP packets. This will consume your bandwidth and by consequence, your money. Additionally, 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 in 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 (any NAT device except symmetric NAT). Use the 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>.

Application layer gateways (ALGs)

Another very common solution for near-end NAT traversal is the **application layer gateway (ALG)**. Several NAT devices implement ALG. In this case, the NAT device changes the SIP and SDP headers to make the packets look as if it has 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 an SIP provider. Other implementations change the headers, but not the MD5 digest, giving an authentication error (using the hostname instead of the IP address of the SIP server solves this problem).

It is important to be aware of NAT devices with ALG on your network. When something is not working is an important place to check.

In some cases, you may use OpenSIPS combined with RTP Proxy as an ALG server. In this case, you are going to handle the NAT traversal in a different box. This may simplify the script for the SIP proxy.

Interactive Connectivity Establishment (ICE)

ICE is the newest protocol for NAT traversal. It combines STUN and TURN to choose the best path available. ICE uses all the available methods 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 understood the implications of the symmetrical NAT, and the use of STUN and TURN. You learned how to implement a Media Relay Server, known as RTP Proxy 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, use Media Proxy or RTP Proxy.

10

OpenSIPS Accounting and Billing

In the previous chapter, we learned how to implement NAT traversal, now it is time to focus on one of the most important things for a VoIP provider—revenue. The accounting feature will allow you to determine the exact duration of each call. We will show you two methods. The first one uses a MySQL server while the second uses a RADIUS server. It is common for Internet providers and telcos to have a RADIUS infrastructure in place. RADIUS is a de-facto standard for AAA (Authentication, Authorization, and Accounting).

Objectives

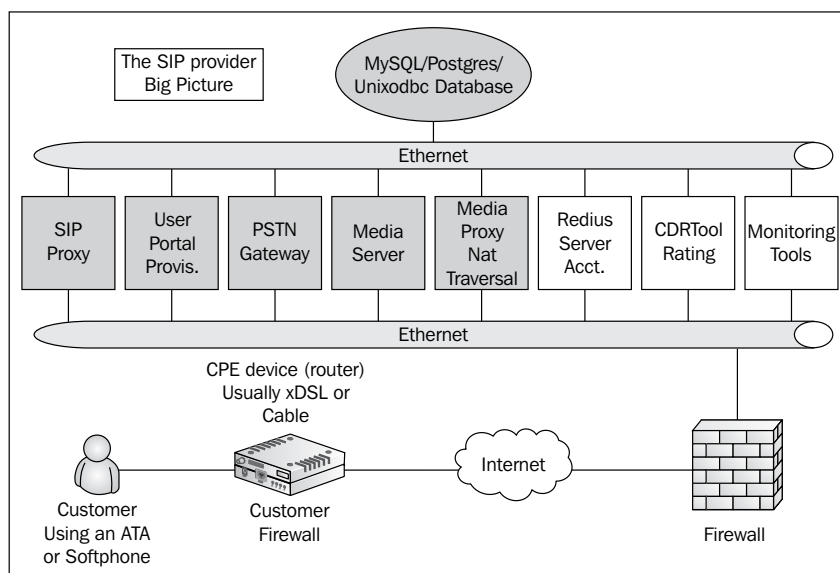
By the end of this chapter, you will be able to:

- Enable automatic accounting to a MySQL server
- Account missed calls
- Account failed calls
- Add extra fields to your accounting table
- Generate a Call Detail Record matching INVITE and BYEs
- Configure `opensips-cp` to see your CDR

Where we are

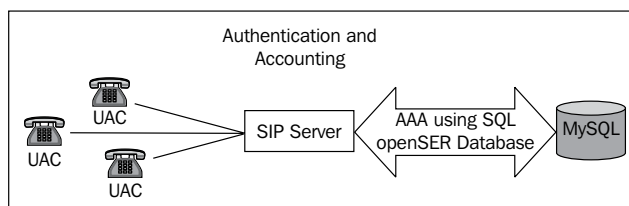
We are going to work on the billing side of the solution. The proxy is working fine, completing calls between users and gateways. However, we are not billing the calls. Billing is a two-step process:

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.



VoIP provider architecture

The VoIP server uses the concept of **Authentication, Authorization, and Accounting (AAA)**. Until now, we have used only MySQL to authenticate and 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 INVITES and BYEs.



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. The best place to bill the calls is on the gateways, because a call can be left open after an INVITE without the respective BYE. Another 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 starts, so the accounting information will be incomplete. In the gateways, it is also 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 `opensips-cp` to check the records. We need to set a flag in the transactions to be accounted. Accounting is activated in the default script but with just some of the fields.

Example:

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 the caller and the callee, you need to add some extra data.

Automatic accounting

OpenSIPS supports automatic accounting. You only need to flag the transaction you want to account. Usually, it makes sense only to account INVITEs and BYEs. To enable automatic accounting, you need only to set the appropriate flag. It is also possible to add extra data to the accounting records. In the next lab, you will see how to do this. Automatic accounting is controlled by the following parameters.

When using parallel forking, the system will account the first successful answer to the request. If you don't have any successful reply (2XX) and the `failed_transaction_flag` is set, the system will account the latest failed transaction.

If a call is not answered and you have the `db_missed_flag` set, the failed call will be accounted to the missed calls table in the database. This is useful when you redirect calls to voicemail, but still want to log the missed calls.

It is also possible to account additional data such as 183 replies (`early_media`), ACK messages (`report_ack`), and canceled transactions (`report_cancels`).

The default accounting does not include the `caller_id` and `callee_id`. You may add additional information with `extra_data`.

```
modparam("acc", "db_extra", "caller_id=$fu; callee_id=$tu")
```

Multi-leg accounting

Accounting might be affected by operations such as call transfer and call forward. You need to take this into consideration. In some cases, it may be useful to account REFER messages. It is possible in OpenSIPS to account *multi-leg* calls. There is the possibility to add additional information about the legs of a call in the accounting data. Let's suppose that you have a call from A to B and this call is forwarded to C. It is up to the script writer to decide if he wants to account the leg B to C only to B. In normal accounting, the call would be accounted from A to C only. To add multi-leg data use:

```
modparam("acc", "multi_leg_info", "leg_src=$avp(src); leg_
dst=$avp(dst) ")
```

Now you have this new information in the `leg_src` and `log_dst` columns.

Lab—accounting using MySQL

In this lab, we are going to enhance the accounting by adding two extra fields.

Step 1: Add the following fields in the ACC table:

```
mysql -u root

USE OPENSIPS;
ALTER TABLE `acc` ADD `caller_id` CHAR( 64 ) NOT NULL ;
ALTER TABLE `acc` ADD `callee_id` CHAR( 64 ) NOT NULL ;
```

Step 2: Make the highlighted changes to the script:

```
# ----- acc params -----
modparam("acc", "early_media", 1)      #Report early media "183" replies
modparam("acc", "report_ack", 1)        #Report acknowledges
modparam("acc", "report_cancels", 1)    #Report CANCELS "487"
modparam("acc", "failed_transaction_flag", 3) #Report failed trans.
#modparam("acc", "log_flag", 1) # Disable log to syslog
#modparam("acc", "log_missed_flag", 2) #No missed calls to syslog
modparam("acc", "db_flag", 1)           # Flag 1 to Account call
to DB
```

```

modparam("acc", "db_missed_flag", 2) # Flag 2 to Account Missed Calls
modparam("acc", "db_url", / "mysql://opensips:opensipsrw@localhost/
opensips") #Pointer to the DB
modparam("acc", "db_extra", "caller_id=$fu; callee_id=$tu")#Extra Data

# account only INVITES
if (is_method("INVITE")) {
    setflag(1); # Do accounting
    setflag(2); # Account Missed Calls
    setflag(3); # Account failed transactions
}

if (is_method("BYE")) {
    setflag(1); # do accounting ...
    setflag(3); # ... even if the transaction fails
}

```

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;

```

Analysis of the opensips.cfg file

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 OpenSIPS to account transactions marked with the flag number 1. The parameter `db_missed_flag` tells OpenSIPS 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 previously created in the database and a value that can be taken from pseudo-variables, AVPs, and headers.

```

# ----- acc params -----
modparam("acc", "early_media", 1)    #Report early media "183" replies
modparam("acc", "report_ack", 1)      #Report acknowledges
modparam("acc", "report_cancels", 1)  #Report CANCELS "487"
modparam("acc", "failed_transaction_flag", 3) #Report failed trans.
#modparam("acc", "log_flag", 1) # Disable log to syslog
#modparam("acc", "log_missed_flag", 2) #No missed calls to syslog
modparam("acc", "db_flag", 1)         # Flag 1 to Account call to DB

```

```
modparam("acc", "db_missed_flag", 2) # Flag 2 to Account Missed Calls
modparam("acc", "db_url", / "mysql://opensips:opensipsrw@localhost/
opensips") #Pointer to the DB
modparam("acc", "db_extra", "from_uri=$fu; to_uri=$tu") #Extra data
```

The default script has two places where it activates accounting. The first one is for INVITEs. Usually, it does not make sense to account other transactions. If you were to allow call transfers, it may be useful to account REFERs.

```
# account only INVITEs
if (is_method("INVITE")) {
    setflag(1); # Do accounting
    setflag(2); # Account Missed Calls
    setflag(3); # Account failed transactions
}
```

BYEs are being flagged in the `loose_route` section, because we are using record routing. INVITEs are being flagged for accounting in the initial requests section. You don't need to flag re-INVITEs.

```
if (is_method("BYE")) {
    setflag(1); # do accounting ...
    setflag(3); # ... even if the transaction fails
}
```

You may now check the accounting table using `serMyAdmin` or `phpMyAdmin`. Look for the ACC table in the OpenSIPS database.

Generating the CDRs

To generate the CDRs, you will have to track the INVITE and BYEs. An INVITE and its respective BYEs belong to the same dialog. Thus, they share the same Call-ID, FROM tag, and TO tag. We are going to use a slightly modified MySQL stored procedure to generate the CDRs.

A MySQL stored procedure is a script that runs directly from the MySQL database. We have to insert the script into the database and call it from time to time from the CRON database.

Lab—generating Call Detail Records

The procedures to generate the CDR from the accounting records are as follows:

Step 1: Install the cdr table schema:

```
cd /var/www/opensips-cp/web/tools/cdrviewer
mysql -D opensips -p < cdrs.sql
mysql -u root -p
mysql> use opensips
mysql> ALTER TABLE acc ADD COLUMN cdr_id bigint(20) DEFAULT 0;
mysql -D opensips -p < opensips_cdrs_1_5.sql
```

Step 2: Edit the cron_job/generate-cdrs.sh file and change the mysql connection data (hostname, username, password, and database) there:

```
cd /var/www/opensips-cp/cron-job
vi generate_cdrs.sh
```

Step 3: To generate the CDR records regularly, insert the shell script in the crontab file. Edit the file /etc/crontab and add the following line for a three-minute interval:

```
vi /etc/crontab
*/3 * * * * root /var/www/opensips-cp/cron_job/generate-cdrs.sh
```

Step 4: Insert the stored procedure /usr/src/opensips_cdrs_bk.sql into the database. This stored procedure will calculate the duration of the call and insert it into the cdrs table. It has been modified to include caller_id, called_id, and leg_type.

```
mysql -D opensips -p < /usr/src/opensips_cdrs_bc.sql
```

Step 5: Execute the CDR generation stored procedure:

```
/var/www/opensips-cp/cron-job/generate-cdrs.sh
```


Step 6: Go to OpenSIPS Control Panel and check the `cdrviewer`. You should see something like:

Search CDRs by:										
☐ CDR field :	Caller									
☐ Start Date :	2009	-	01	-	01	00	:	00	:	00
☐ End Date :	2009	-	06	-	05	23	:	59	:	59
Search Show All Export										
Caller	Callee	Call Start Time	Duration	Leg Type	Trace					
1000	55482345678	2009-06-05 11:06:58	8	pstn						
Page: 1					Total Records: 1					

Accounting using RADIUS

Remote Authentication Dial in User Service (RADIUS) is a kind of AAA service. It is a de-facto standard for Internet access providers. From the last few years, RADIUS is becoming an important security protocol used in several network applications such as **Network Access Control (NAC)** and VoIP accounting. You can implement a radius server using an 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:

1. RFC2865: Authentication
2. RFC2866: Accounting

In this chapter, we will use RADIUS only for accounting. MySQL will be held in the authentication function.

Lab—accounting using a FreeRADIUS server

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 into five steps:

- Package and dependencies installation
- FreeRADIUS and `radiusclient` installations
- OpenSIPS configuration

Package and dependencies

Step 1: For FreeRADIUS, install the following packages:

```
apt-get install freeradius libradiusclient-ng2 libradiusclient-ng-dev
```

Step 2: Compile OpenSIPS to use RADIUS

Go to the source code and edit the `Makefile` file, remove from the excluded the module `aaa_radius`.

Compile the code using:

```
make prefix=/ all && make prefix=/ install
```

FreeRADIUS client and server configuration

Now, let's configure the radius client and server.

Step 1: Configure both client and server to share the same secret.

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=opensips
    shortname=OpenSIPS
    nastype=other
}
```

Edit the servers in the radius client:

```
vi /etc/radiusclient-ng/servers
```

#Server Name or Client/Server pair	Key
127.0.0.1	opensips

Step 2: Configure the sip dictionary for both client and server:

```
cd /usr/share/dictionary
vi dictionary
    $INCLUDE          /etc/freeradius/dictionary.ser
```

 Be sure to include it in the last line.

```
cd /etc/radiusclient-ng/dictionary
vi dictionary
    $INCLUDE          /etc/freeradius/dictionary.ser
```

 Be sure to include it in the last line.

Step 3: Restart the FreeRADIUS server:

```
/etc/init.d/freeradius restart
```

Configure OpenSIPS server

Now let's configure the OpenSIPS server to send the accounting to the FreeRADIUS database.

Step 1: Add the following lines to the system at the proper places:

```
#--- Modules Loading
loadmodule "aaa_radius.so" # Load the Radius Base API
# --- Modules Parameters
modparam("acc", "aaa_url", "radius:/etc/radiusclient-ng/radiusclient.
conf") # Point the configuration file
modparam("acc", "aaa_extra", "Calling-Station-Id=$fU;Called-Station-
Id=$rU") # Add extra data
modparam("acc", "aaa_flag", 1) #Use the same flag for Radius
modparam("acc", "service_type", 15) # Set the service_type to 15

# ----- radius params -----
modparam("aaa_radius", "radius_config", "/etc/radiusclient-ng/
radiusclient.conf")
```

Step 2: Restart OpenSIPS and FreeRADIUS

Step 3: Make some calls

Step 4: Check for RADIUS accounting in the directory `/var/log/freeradius/radacct`

You should see something like:

Fri Oct 9 18:08:49 2009

Acct-Status-Type = Start

Service-Type = SIP

Sip-Response-Code = 200

Sip-Method = Invite

Event-Timestamp = "Oct 9 2009 18:08:49 EDT"

Sip-From-Tag = "a10c246c"

Sip-To-Tag = "as7d5b4474"

Acct-Session-Id =

"N2QwYzZjNmQxYzRjMjg5ZWFiNmUxNWJhNWJhYmUxMGE."

Calling-Station-Id = "1000"

Called-Station-Id = "*98"

NAS-Port = 5060

Acct-Delay-Time = 0

NAS-IP-Address = 127.0.0.1

Acct-Unique-Session-Id = "6d767b7ee4892789"

Timestamp = 1255126129

Request-Authenticator = Verified

Fri Oct 9 18:08:50 2009

Acct-Status-Type = Stop

Service-Type = SIP

Sip-Response-Code = 200

Sip-Method = 8

Event-Timestamp = "Oct 9 2009 18:08:50 EDT"
Sip-From-Tag = "as7d5b4474"
Sip-To-Tag = "a10c246c"
Acct-Session-Id =
"N2QwYzZjNmQxYzRjMjg5ZWFiNmUxNWJhNWJhYmUxMGE."
Calling-Station-Id = "*98"
Called-Station-Id = "1000"
NAS-Port = 5060
Acct-Delay-Time = 0
NAS-IP-Address = 127.0.0.1
Acct-Unique-Session-Id = "6d767b7ee4892789"
Timestamp = 1255126130
Request-Authenticator = Verified

Solving the problem with missing BYEs

One of the biggest issues with SIP accounting is the occurrence of missing BYEs. If one leg of a call is abruptly disconnected from the network, the BYE request is not generated. In this case, it is not possible to generate the BYE event and to determine the duration of the call correctly. There are some approaches to solve this issue:

- Account in the gateway instead of the proxy
- Use SIP session timers
- Use RTP proxy timeout
- Use Media proxy timeout

Account in the gateway instead of the proxy

One of the main differences between the gateway and the proxy is that you have the media passing through the gateway. Using RTP timeout you can detect when a connection gets stuck and close the call. The main disadvantage of this method is when you use third-party gateways where you don't have control over the accounting process.

Use SIP session timers

SIP session timers, described in RFC4028, 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 (in case of a disconnected network), the proxy does not have a mechanism to close this call and generate the CDR precisely. To implement SIP session timers, it is necessary to have support for at least one of the SIP components—the client or the gateway. The advantage of this method is to use only signaling without any control of the media. The disadvantage is the need to depend on the client or the gateway, sometimes a VoIP provider does not have total control over these components.

Use RTP proxy timeout

Recently, a timeout socket was included in RTP proxy. You can use an external program connected to this socket to catch the timeout events and to fix the accounting. You will also need to force all billable connections over the RTP proxy. The main disadvantage is the overhead caused by forcing all the connections through the RTP proxy. The main advantage is that you have complete control over the solution. To activate the socket and the timeout events, use the `-n` option for the `rtpproxy` daemon.

Example:

```
-n unix:/var/run/rtpproxy_timeout.sock
```

Use Media Proxy timeout

Media Proxy is a NAT traversal component provided by AG-Projects. It is used with a rating tool called CDRTools. The combination of Media proxy and CDRTool is very powerful to create a pre and post paid billing system for OpenSIPS. Media Proxy is able to detect sessions without media and to fix the duration of the call directly in a RADIUS server.

Prepaid and postpaid billing

OpenSIPS generates only accounting events in the MySQL database or the RADIUS server. We have used `opensips-cp` to generate the duration of the call from the accounting records generated by OpenSIPS. However, in the daily operation of a VoIP provider, you will need a tool to bill your users. A detailed explanation of these tools is beyond the scope of this book. I will present here an introduction of CDRTool—a popular open source tool for prepaid and postpaid billing.

CDRTool is a mediation and rating engine for Call Detail Records generated by OpenSIPS in combination with a FreeRADIUS server. CDRTool provides accurate accounting when used in combination with Media Proxy. Rating functions are available from the network over a TCP socket and can be used for both postpaid and prepaid applications. The call control prepaid application for OpenSIPS is available to provide session control for prepaid services.

CDRTool provides a set of features that can solve many issues related to accounting for both prepaid and postpaid calls. Some of the most important features supported by the tool are:

- Real-time rating engine for postpaid and prepaid accounting
- Web and CSV file management for rating tables
- CDR search with query criteria saved for later use
- Search results can be grouped by any field available in the CDR
- Multiple data sources with consistent search and export capabilities
- Restrict access to CDRs per subscriber, domain or gateway
- Rating based on the day of the week, time of day, duration, destination, and ENUM
- Manage prepaid cards and accounts for call control
- Supports RADIUS accounting type 15 (FAILED) for storage of missed calls

The complete installation guide for CDRTool can be found at:

<http://cdrtool.ag-projects.com/wiki/Install>

Summary

In this chapter, we learned how to implement one of the most sensitive components of a VoIP provider—accounting. Accounting might be done in MySQL, syslog, or RADIUS. We have installed and tested accounting in MySQL and RADIUS.

11

Monitoring Tools

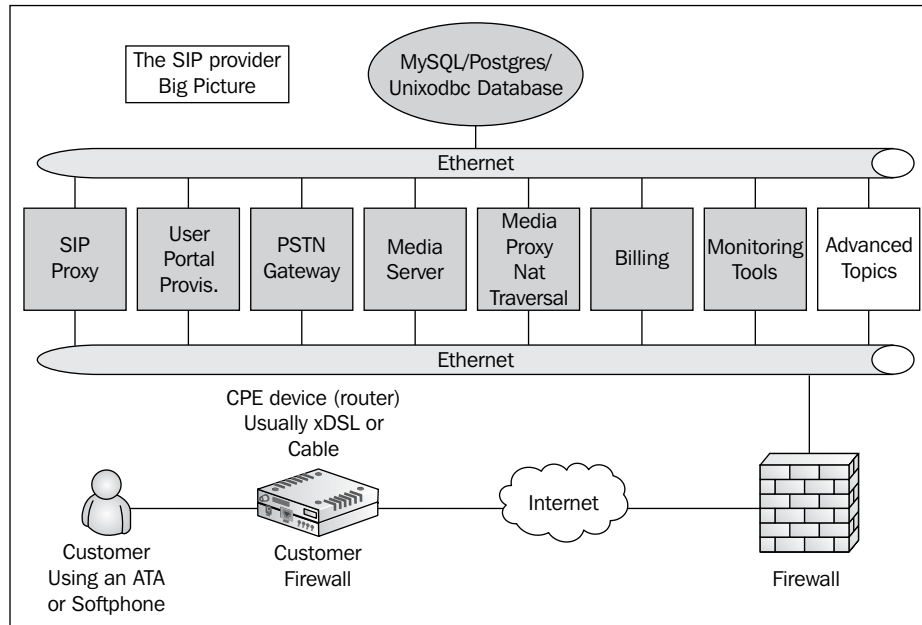
After installing the whole system, comes a new phase—test, operation, and maintenance. In this chapter, we are going to cover some tools and utilities to help you with this task. We will start with built-in monitoring tools such as OpenSIPS statistics and SIPTRACE. Following this, we are going to show testing tools such as SIPp and SIPSAK.

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

- Understand how to use built-in tools such as `opensipsctl`
- Troubleshoot customer signaling using SIPTRACE
- Stress test OpenSIPS using SIPp
- Create automated tests using SIPSAK
- Choose the right tool to help with OpenSIPS monitoring

Where we are

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.



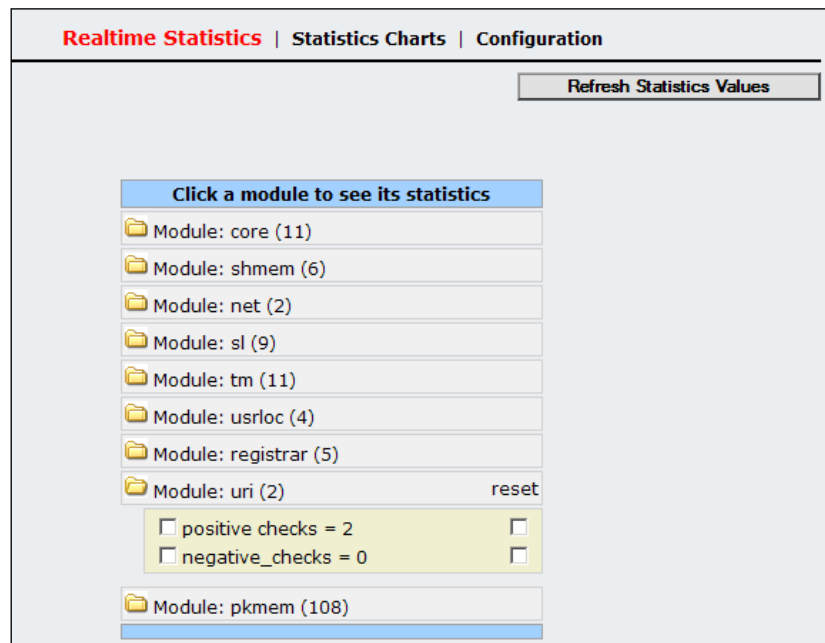
Built-in tools

In Version 1.6, we have a bunch of new tools to manage OpenSIPS. The main tools are `opensipsctl`, the OpenSIPS Control Panel, and SIPTRACE.

The `opensipsctl` shell script has some options to generate statistics for OpenSIPS. Let's expose some FIFO commands to generate statistics about OpenSIPS.

Command	Result
<code>opensipsctl fifo which</code>	Displays all available commands
<code>opensipsctl fifo ps</code>	Displays all OpenSIPS processes
<code>opensipsctl fifo get_statistics core:</code>	Displays statistics about the core
<code>opensipsctl fifo get_statistics net:</code> (new 1.6)	Displays Net sockets
<code>opensipsctl fifo get_statistics pkmem:</code> (new 1.6)	Displays private memory of each process
<code>opensipsctl fifo get_statistics tm:</code>	Displays TM module statistics
<code>opensipsctl fifo get_statistics sl:</code>	Displays SL module statistics
<code>opensipsctl fifo get_statistics shmem:</code>	Displays shared memory statistics
<code>opensipsctl fifo get_statistics usrloc:</code>	Displays user location statistics
<code>opensipsctl fifo get_statistics registrar:</code>	Displays REGISTRAR statistics
<code>opensipsctl fifo get_statistics uri:</code>	Displays URI statistics

All these commands should be graphed using the OpenSIPS Control Panel.

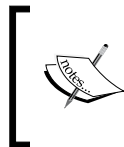


Some statistics to pay attention to are:

- **TM—inuse_transactions**: Number of transactions in use in memory. If this number is increasing, you could have a situation where your transactions are not being released quickly. Each transaction uses a small amount of memory.
- **NET—waiting_udp, waiting_tcp**: If the socket has some packets waiting to be processed, it means that your system is not fast enough to keep up with the requests.
- **SHMEM**: Check shared memory statistics to see if you have any memory leakage.

It is common to receive inquiries on how to obtain the number of simultaneous calls and the number of registered users. I would like to say that the number of simultaneous calls is not the most important statistic for OpenSIPS. For a SIP proxy, a five-minute call consumes exactly the same amount of work as a one-hour call. The proxy only needs to relay INVITE and BYE. So what defines the performance of a SIP proxy is the number of transactions processed. It is important to remember that SIP is not only meant for calls, but it may also be used for presence, instant messaging, video, and other uses. The statistic that defines the real performance of a SIP proxy is the number of transactions per second. However, if you want to determine the number of simultaneous calls (dialogs), you might use:

```
opensipsctl fifo get_statistics dialog:
```

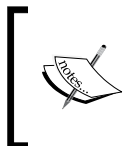


Check for active dialogs

If you are establishing other types of dialogs such as presence, the number of active dialogs will not exactly match the number of calls. Again, SIP is not only about calls.

To show statistics about the modules in an interactive way, you could use:

```
opensipsctl moni
```



Be sure that your `mi_fifo` module is correctly configured, otherwise `opensipsctl` won't work for OpenSIPS statistics. Please check your `opensipsctlrc` file to see if the FIFO is pointing to the file `/tmp/opensips_fifo`.

These 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 completed. The messages sent bring you some information about the errors that occur. Finally, the `usrloc` statistics allow you to check the health of the REGISTER processes.

Trace tools

There are several packet capture and trace tools for OpenSIPS. One of the simplest is `ngrep`, which is used throughout this book. Another important tool is the SIPTRACE module. With this tool, you can trace calls from a specific user in real time. SIPTRACE can impact the performance of your system when enabled. The SIPTRACE module logs the SIP signaling to a database for inbound and outbound traffic.

SIPTRACE

The module is simple to use, you need to load the module and mark the transactions you want to record using a specific flag. This flag is defined in the module parameter `trace_flag`. You probably don't want to record all messages to the database because of the overhead. It is possible to start the trace for a single user.

The module parameter `traced_user_avp` allows you to specify a user to be traced. The name of the user will be shown in the `traced_user` field. You can store multiple values in the AVP if you want to trace more than a single user. The SIPTRACE module can be enabled and disabled using the `fifo` command `sip_trace on/off`. It is possible to visualize the results of the traced call using the OpenSIPS Control Panel.

Configuring the SIPTRACE

In order to enable SIPTRACE, add the following instructions to your script:

```
loadmodule "siptrace.so"
modparam("siptrace", "db_url", "mysql://opensips:opensipsrw@localhost/opensips")
modparam("siptrace", "trace_flag", 22)
modparam("siptrace", "traced_user_avp", "$avp(s:traceuser)")
modparam("siptrace", "trace_local_ip", "ip_address_of_your_server")

if(avp_db_load("$fu", "$avp(s:trace)")) {
    $avp(s:traceuser)=$fu;
    setflag(22);
    sip_trace();
    xlog("L_INFO", "User $fu being traced");
}
```

After restarting the script, add the user to be traced using:

```
opensipsctl avp add -T usr_preferences 1000@youripaddress trace 0 1
```

Now start SIPTRACE using:

```
opensipsctl fifo sip_trace on
```

Once everything is configured correctly, try making a call from the user 1000 to the user 1001. You will see the results in the `sip_trace` table or in the OpenSIPS Control Panel as follows:

Date Time	Method	Status	Path	Details
2009-10-20 23:24:32	BYE	200	 ← 	
2009-10-20 23:24:32	BYE	200	 ← 	
2009-10-20 23:24:32	BYE		 → 	
2009-10-20 23:24:30	INVITE	200	 ← 	
2009-10-20 23:24:30	INVITE	200	 ← 	
2009-10-20 23:24:27	INVITE	180	 ← 	
2009-10-20 23:24:27	INVITE	180	 ← 	
2009-10-20 23:24:27	INVITE		 → 	
 Caller  Proxy  Callee				

Stress testing tools

Now we will present some tools for stress-testing your OpenSIPS server before going to production. The first tool is `sipsak` (www.sipsak.org) and the second is `SIPp` (sipp.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 also good 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 see an example of how to use it. Install it using:

```
apt-get install sipsak
```

Example of use:

You can ping a UAC using the `OPTIONS` method by issuing:

```
sipsak -vv -s sip:1000@opensips.org
```

To register a user and return a completion code you may use:

```
sipsak -U -s sip:1000@192.168.1.185 -a 1000 -W 1 -vvvvv
```

To trace a call, use:

```
sipsak -T -s sip:1000@opensips.org
```

To send a message using the MESSAGE method, use:

```
sipsak -M -s sip:1000@opensips.org -c flavio@opensips.org -B "time for a  
coffee break"
```

You may also use the stress (-F) and the torture (-R) mode. You may find the complete documentation on the sipsak website—www.sipsak.org. The best results are obtained by combining SIPSAK with Nagios (<http://www.nagios.org>) or even a simple shell script.

SIPp

To explain each detail of SIP is beyond the scope of this material. The idea here is to give you an overview of SIP and teach you how to get started. Allow enough time to test your platform; you will need a lot of time to build a test lab with several UACs and UAS and interpret the results.

SIPp is a tool for traffic generation and stress-testing for SIP. It is a good tool to use in order 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 their website: 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

Installing 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
```

Downloading and decompressing the sip source file:

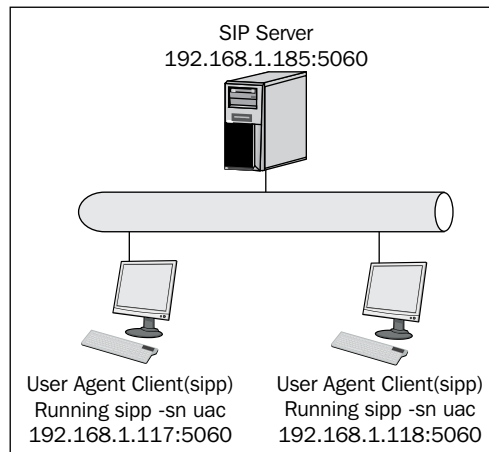
```
wget http://downloads.sourceforge.net/sipp/sipp.3.1.src.tar.gz
tar -xzf sipp.3.1.src.tar.gz
```

Compile it, including the ssl libraries to allow authentication:

```
make ossl
./sipp
```

Stress test—the SIP signaling

I don't want to get into the details of SIPp's scenario customizations, so we are going to use the default uac and uas scenarios. In these scenarios, authentication is not being used and there is no record routing in the client. Please check the SIPp documentation if you intend to test OpenSIPS with authentication. On the OpenSIPS website, there is a good example of how to test the register requests with authentication.



Step 1: You will need to register the user agent server manually, by adding static mapping in the user location table. In the following example, we are saying that the user 1003 is at the address 192.168.1.117 (where we started the UAS).

```
opensipsctl ul add 1003 sip:1003@192.168.1.117:5060
```

Step 2: Add the following addresses as domains (to make sure all calls are handled as intra-domain)

```
opensipsctl domain add 192.168.1.185
opensipsctl domain add 192.168.1.117
opensipsctl domain reload
```

Step 3: Change the script to avoid authentication and loose routing for sipp packets (use the `0745_11_02.cfg` script provided in the code bundle). In the following code, the sections are changed. Notice that I'm skipping the loose routing section where the client is sipp (the default user in the `From` header for SIPp).

```
if (has_totag() && ($fU!="sipp")) {
  #if(!is_from_gw()){
  #    if (!proxy_authorize("", "subscriber")) {
  #        proxy_challenge("", "0");
  #        exit;
  #    }
  #    if (!db_check_from()) {
  #        sl_send_reply("403","Forbidden auth ID");
  #        exit;
  #    }
  #    consume_credentials();
  #    # caller authenticated
  #}
}
```

To start the user agent server, use:

```
./sipp -sn uas -rsa 192.168.1.185 -i 192.168.1.118
```

It will show you a screen like the following:

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 `0745_11_02.cfg` which I have used for these tests. You will have to manually handle the ACKs and BYE as SIP does not support record routing by default.

To start the user agent client, use:

```
./sipp -sn uac 192.168.1.185:5060 -s 1003 -p 5060 -d 1000 -i
192.168.1.117
```

```
----- 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
    180 <-----      4177     78
    183 <-----         0         0
    200 <----- E-RTD1 4511    322
    ACK ----->      4511    322
Pause [      0ms]      4511
    BYE ----->      4511   3074      0
    200 <-----      4389     0
                                122

----- [|-*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Increase the call rate by using the + key until you start seeing retransmissions. In the preceding case, 100 simultaneous calls with rtpproxy support was enough. The previous screen is referent 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 could fill up your hard disk easily.

Stress test—the RTP signaling

It is possible to test the RTP signaling using a combination of the UAS with the `rtp_echo` function combined with a UAC using the `pcap` function. See the details in the SIP documentation.

```

----- Scenario Screen ----- [1-9]: Change Screen --
Port    Total-time  Total-calls  Transport
5061    2500.87 s      1118    UDP

0 new calls during 1.000 s period      3 ms scheduler resolution
16 calls                               Peak was 110 calls, after 387 s
0 Running, 16 Paused, 0 Woken up
1 open sockets
39217 Total echo RTP pkts 1st stream  0.000 last period RTP rate (kB/s)
0 Total echo RTP pkts 2nd stream      0.000 last period RTP rate (kB/s)

-----> INVITE                Messages  Retrans  Timeout  Unexpected-Msg
                                768      2268           350

<----- 180                    768      2268
<----- 200                    768      4875      469
-----> ACK      E-RTD1 81        12           0

-----> BYE                    283      97           0
<----- 200                    283      97
[  4000ms] Pause                283           214
----- Sipp Server Mode -----

```

To start the UAS with RTP echo, use a command similar to the following. Please adapt the scenario to your own situation before testing.

Example:

```
sipp -sf uas.xml -rtp_echo -mi 192.168.1.116 -mp 1000 -rsa 192.168.1.185
-i 192.168.1.117
```

In the previous image, you can see the data related to RTP packets echoed by the system.

To start the UAC with `pcap`, use a command similar to:

```
sipp -sf uac_pcap.xml -s 1003 192.168.1.185:5060 -d 1000 d 1000 -i
192.168.1.117
```

You will see a screen similar to the following:

```

11 new calls during 1.001 s period      2 ms scheduler resolution
60 calls (limit 270)                    Peak was 60 calls, after 6 s
0 Running, 59 Paused, 1 Woken up
0 out-of-call msg (discarded)
1 open sockets
2090 Total RTP pkts sent                133.679 last period RTP rate (kB/s)

      INVITE ----->      Messages  Retrans  Timeout  Unexpected-Msg
      100 <-----      60          69         0           0
      180 <-----      28          11         0           0
      200 <-----      17           0         0           0
      200 <----- E-RTD1 17          13         0           0

      ACK ----->      17          13
      [ NOP ]
      Pause [ 8000ms ]  17
      [ NOP ]
      Pause [ 1000ms ]  0
      BYE ----->      0           0         0           0
      200 <-----      0           0         0           0

----- [+!-!*!/] : Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----

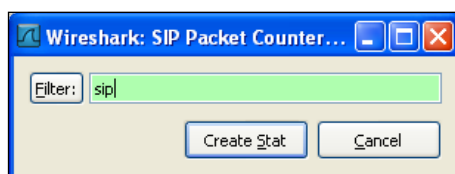
```

Wireshark

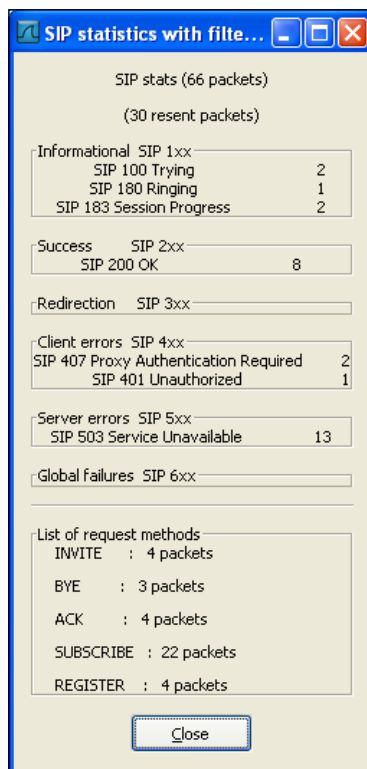
Wireshark (formerly Ethereal) is the most used protocol analyzer available in the market and it is GPL-licensed. To teach you exactly how to use a protocol analyzer is beyond the scope of this material. However, we will give you some tips on analyzing SIP and RTP packets. Wireshark has some special statistics for SIP and RTP. After loading the captured packets, you can start analyzing statistics of 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:

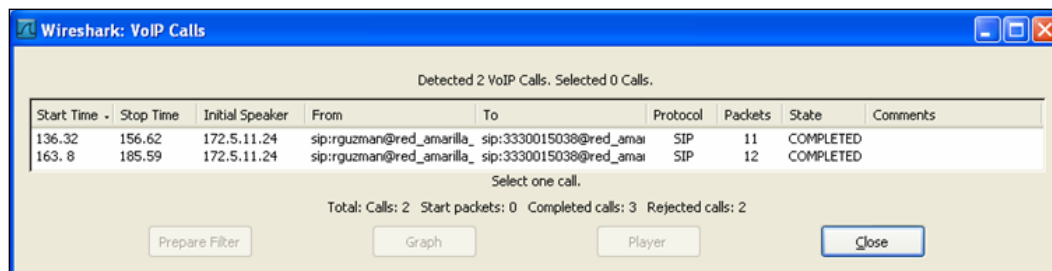


It is a nice trick. You can now check general statistics about your SIP messages. Well, this is not our best trick, but it can help to spot abnormal behavior.

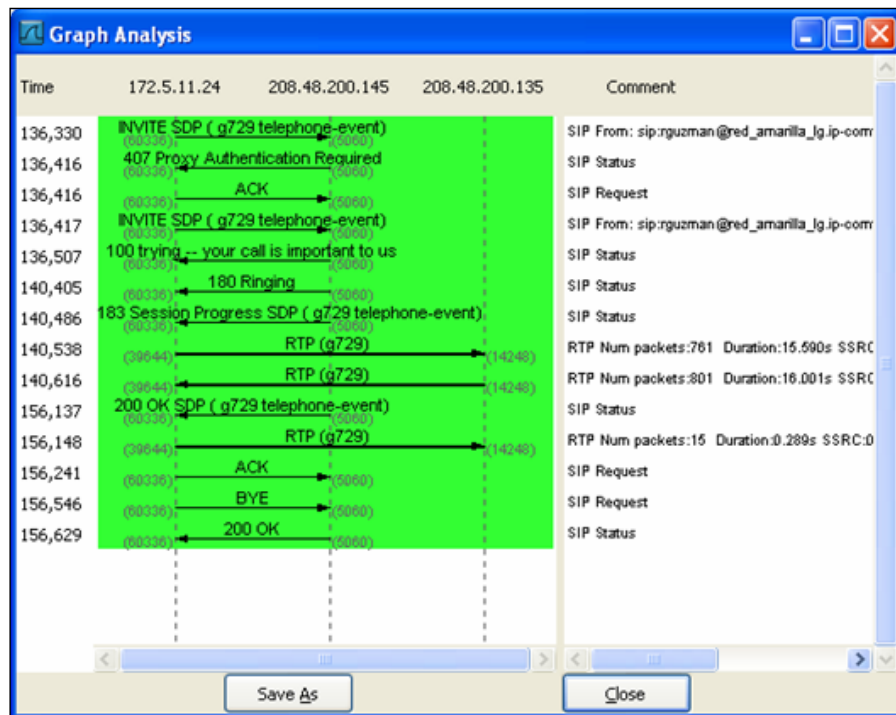
Let's go to the second trick to graph the SIP dialog. In the Wireshark menu, select:

Statistics | VoIP Calls

You will see the following screen:



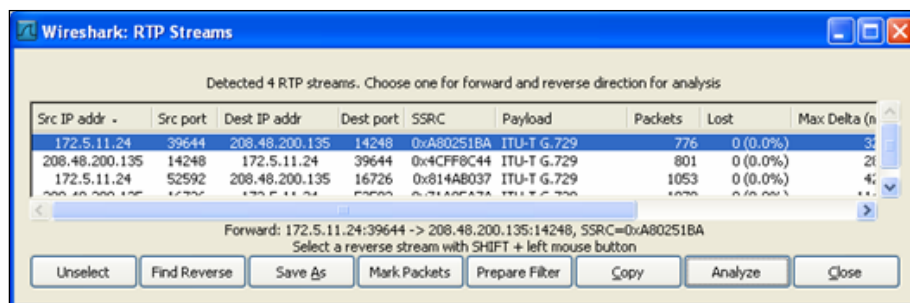
In this screen, you can select the call you want to graph. After selecting the call, press the **Graph** button.



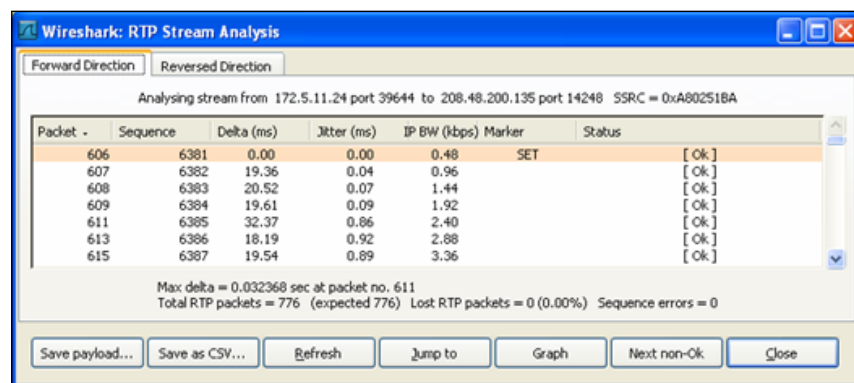
You will see this amazing graph with the SIP dialog. Now you can spot specific problems in a single dialog. You can even play a call in the previous menu if it is coded using g.711 alaw or ulaw.

Well, now let's check the RTP packets. After all, RTP packets will determine the voice quality. There is no single recommendation, we consider a call as having good voice quality when the latency is below 150 ms (the one-way equivalent to a round-trip time of 300 ms), the jitter is below 20 ms, and the packet loss below three percent. You can have 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. 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 your own standards. To help you in this task, let's use the following statistics. In the Wireshark menu, select:

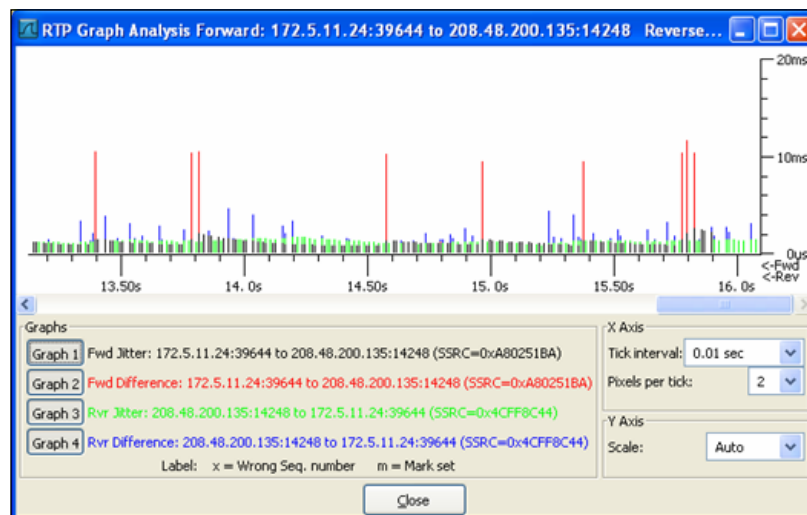
Statistics | RTP | Stream Analysis



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 the packets.

Monitoring tools

To monitor OpenSIPS, you can use a set of utilities along with the network monitoring tools. You can use Nagios along with SIPSAK to monitor real transactions such as REGISTER and INVITE. MONIT (<http://mmonit.com/monit/>) is another tool you can use to monitor OpenSIPS from within. Using MONIT, you can generate alerts about the status of the system and the OpenSIPS daemon. A good tutorial on how to set up MONIT with OpenSIPS can be found at www.voip-info.org/wiki/view/OpenSER+And+Monit.

Summary

In this chapter, we learned about the main tools for testing and monitoring OpenSIPS. It is wise to stress test OpenSIPS 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 with them, because you will certainly need to use them. Finally, MONIT can be used to monitor the processes and help you keep OpenSIPS up and running.

Index

A

AAA 226

Access Control List. *See* **ACL**

accounting, OpenSIPS

configuration 227

MySQL used 228

ACL 156

address-of-record. *See* **AOR**

address permissions 149, 150

af 71

ALGs 223

alias

adding 117, 118

used 125

alias_db_lookup("dbaliases" function) 118

alias parameter, global parameters 67

allow_routing()function 148

allow_trusted() function 149

allow_uri()function 149

AOR 70

append_fromtag parameter 69

Application layer gateways. *See* **ALGs**

arithmetic operations, script variables 73

Asterisk

Asterisk gateway (sip.conf) 160

Cisco 2601 gateway 161, 162

using, as PSTN gateway 159, 160

Asterisk PBX 147

Asterisk Real Time

integrating, with OpenSIPS 178-181

Attribute-Value Pair. *See* **AVP**

Attribute-Value Pairs Operations. *See* **AV-POPS**

attribute value pair. *See* **AVP**

AUTH_DB module

about 92

parameters 93

proxy_authorize(realm, table) function 93

www_authorize(realm, table) function 93

AUTH_DB module, parameters

calculate_ha1 93

db_url 93

domain_column 93

load_credentials 93

password_column 93

password_column2 93

use_domain 93

user_column 93

Authentication, Authorization, and Accounting. *See* **AAA**

AVP

about 74, 168

avp_check 74

avp_copy 74

avp_db_delete 74

avp_db_load 74

avp_db_query 74

avp_db_store 74

avp_delete 74

avp_op 75

avp_print 75

avp_printf 74

avp_push 74

avp_subst 75

functions 74

is_avp_set 75

usr_preference table 75

avp_check 74

avp_copy 74

avp_db_delete 74

- avp_db_load 74
- avp_db_query 74
- avp_db_store 74
- avp_delete 74
- avp_op 75
- avp_print 75
- avp_printf 74
- avp_push 74
- avp_subst 75
- AVPOPS 183

B

- blacklist parameter 163
- blind call forwarding 182, 183
- blind call forwarding, implementing
 - AVPOPS module, loading 183, 184
 - steps 184, 185
- branch flag 76
- branch routing blocks 35

C

- calculate_ha1 parameter 93
- Call Detail Records. *See* CDRs
- call forwarding
 - about 182
 - blind call forwarding 182
 - blind call forwarding, implementing 183
 - call forward on busy or unanswered, implementing 186, 188, 190
 - configuration file, inspecting 190, 191
 - forward on busy 182
 - forward on no answer 182
 - t_on_failure() function 190
 - testing 192
 - XLOG() function 190
- call forwarding, types
 - blind call forwarding 182
 - forward on busy 182
 - forward on no answer 182
- call forward on busy or unanswered
 - implementing 186, 190
- Call Processing Language. *See* CPL
- CANCEL request
 - handling 118
- CDRs
 - generating 231

- CDRTool
 - about 238
 - features 238
- check_source_address("0") function 150
- check_source_address()function 149, 150, 157
- children directive 66
- CLI 157
- command-line interface. *See* CLI
- core functions 71
- core keywords 71, 72
- core values 71
- CPL 11, 32

D

- daemon options, global parameters 66
- db_check_from() function 108
- db_check_to() function 108
- db_get_user_groups()function 148
- db_is_user_in(148
- db_is_user_in()function 147
- db_mode parameter 70
- db_url parameter 93
- dialog flow, SIP
 - about 15, 16
 - header fields 16, 17
 - method name 16
 - SDP header 19
 - secure SIP URI 16
 - SIP proxy, for domain 17, 18
 - SIP URI 16
- dialogs, SIP 21
- DIALPLAN
 - about 167
 - example 168-171
 - opensips.cfg, inspection 171-173
- digest authentication
 - about 101
 - authorization request header 102
 - QOP (Quality Of Protection) parameter 102
 - WWW-Authenticate response header 101
- DNS blacklist 173
- domain_column parameter 93
- dp_translate()function 171
- DR_GATEWAYS 163, 164
- DR_GROUPS 164
- DR_GW_LISTS 165

DR_RULES 164

drop(); 71

Drouting

case study 165-167

features 162

parameters 162

Drouting, parameters

blacklist 163

Force_dns 163

sort order 163

Drouting, tables

DR_GATEWAYS 163

DR_GROUPS 164

DR_GW_LISTS 165

DR_RULES 164

dst_ip 71

dynamic routing. *See* **Drouting**

E

enable_full_lr parameter 69

error routing block 35

exit(); 71

F

failure routing blocks 35

far-end NAT solution, SIP NAT traversal

force_rport() function 201

implementing 201

RFC3581 201

RTP packets, traversal solving 202, 203

RTP packets traversal, solving 202, 203

fix_nated_contact() function 203

fix_nated_register() command 205

fix_nated_register() function 203

flags

module GFLAGS 76

types 76

flags, types

branch flag 76

message flag 76

script flag 76

Force_dns parameter 163

force_rport() function 201, 202

fork directive 66

forward(); 71

forward()command 37

forward on busy 182

forward on no answer 182

FreeRADIUS server installation

about 232

client, configuring 233, 234

dependencies 233

OpenSIPS server, configuring 234, 236

package 233

server, configuring 233, 234

from_uri 71

full cone NAT 195, 198

G

GCJ 135

global definitions 35

global parameters, OpenSIPS

about 65

daemon, options 66

listen interfaces 65

logging 66

miscellaneous 67

processes, number 66

SIP identity 67

standard script 67, 68

GNU Compiler for Java. *See* **GCJ**

Grails

downloading 133

Groovy on Rails. *See* **Grails**

group.so module 156

group module 147, 148

Groovy Server Pages (GSP) 143

GRUB 54

H

hash passwords 103

HTTP 7

Hypertext Transfer Protocol. *See* **HTTP**

I

ICE 224

IETF 7

if-else statement 76

Inbound inter-domain 116

INET/INET6 71

initial requests 79

installation, OpenSIPS

- hardware requirements 41
- Linux, installing 43-55
- Monit, installing 133
- OpenSIPS, running at Linux boot 56, 57
- OpenSIPS Control Panel 131, 132
- OpenSIPS v1.6.x, downloading 55
- OpenSIPS v1.6.x, installation process 55
- SerMyAdmin 136, 137
- software requirements 42
- VMware virtual machine with Debian, installing 42

Interactive Connectivity Establishment. *See* ICE

Internet Engineering Task Force. *See* IETF

Internet Message Access Protocol (IMAP) 178

Intra-domain 116

Invite

- diagram 216
 - packet, sequence 217-222
- INVITE authentication sequence**
- about 97
 - code snippet 100, 101
 - message, authenticating 97
 - packet capture, ngrep used 98, 100

INVITE messages

- handling 206, 207

is_avp_set 75

is_from_local() function 117

is_uri_host_local() function 117

L

LANs 22

Linux

- installing, for OpenSIPS 43-55

Linux boot

- OpenSIPS, running 56, 57

listen interfaces, global parameters 65

load_credentials parameter 93

Local Area Networks. *See* LANs

local routing blocks 35

location server 10

LOG() function 190

log files 59, 60

logging, global parameters

- log levels 66

log levels, global parameters

- L_ALERT (-3) 66
- L_CRIT (-2) 66
- L_DBG (4) 66
- L_ERR (-1) 66
- L_INFO (3) 66
- L_NOTICE (2) 66
- L_WARN (1) 66

lookup("aliases") function 85, 86, 118

lookup() function 86

loose_route() function 80, 83

loose_route function 80

Low-cost rates (LCR) 109

M

main routing block 35

MediaProxy 201

media proxy, VoIP provider

- for Nat traversal 26

Media Proxy timeout 237

media relay server. *See* MRS

media server

- example 177
- voicemail 178

media server, VoIP provider 26

message flag 76

message transfer agent. *See* MTA

method 71

mf_process_maxfwd_header 81

modparam directive 70

modparam statement 68

module GFLAGS 76

module permissions

- about 148
- address permissions 149
- register permissions 148
- route permissions 148
- Uri permissions 149

modules

- about 35
- append_fromtag parameter 69
- db_mode parameter 70
- enable_full_lr parameter 69
- loading, loadmodule used 68

- modparam directive 70
- mpath statement 68
- standard configuration 69
- usrloc module 70
- Monit**
 - installing 131
- mpath statement** 68
- MRS** 223
- MTA** 135
- multi domain support** 125
- myself** 71
- MySQL support**
 - installing 103-106
- N**
- NAC** 232
- NAT**
 - about 193
 - firewall table 198
 - INVITE messages, handling 206, 207
 - RE-INVITE messages, handling 208
 - REGISTER requests, handling 206
 - responses, handling 207, 208
 - SIP, breaking 194, 195
 - types 195
- NAT, types**
 - full cone 195
 - port restricted cone 196
 - restricted cone 196
 - symmetric 197
- nat_uac_test() function** 203-205
- nathelper module**
 - about 204
 - natping_interval parameter 204
 - ping_nated_only parameter 204
 - rtpproxy_sock parameter 204
 - sipping_bflag parameter 204
 - sipping_from parameter 204
- near-end NAT solution, SIP NAT traversal**
 - implementing 198-200
 - STUN 199, 200
- NET** 242
- Network Access Control.** *See* NAC
- network address translation.** *See* NAT
- non-register requests, opensips.cpg file** 110

O

- Open Database Connectivity (ODBC)** 178
- OpenSIPS**
 - 473/Filtered Destination messages 173
 - about 30
 - alias, using 117
 - Asterisk Real Time, integrating with 178-181
 - CANCEL request, handling 118
 - compiling, packages 42
 - drawbacks 65
 - features 31
 - global parameters 65
 - hardware, requisites 41
 - history 31
 - lab, aliases using 125
 - lab, call forward feature testing 192
 - lab, multi domain support 125
 - Linux, installing for 43-55
 - log files 59, 60
 - missing BYEs issue, solving 236
 - post paid billing 237
 - pre paid billing 237
 - restarting 60
 - running, at Linux bootup 56, 57
 - script 119
 - scripting 64
 - server, configuring 234, 236
 - software, requisites 42
 - starting 60
 - startup options 60, 61
 - stateful operation 37, 38, 39
 - stopping 60
- OpenSIPS, configuration file**
 - about 34
 - core and modules 35
 - message, processing in opensips.cfg 36
 - OpenSIPS.cfg file, sections 35
 - SIP dialog 36
 - SIP session 36
 - SIP transaction 36
- OpenSIPS, features**
 - extendable 32
 - flexibility 32
 - portability 32

- small footprint 32
- speed 32
- usage, scenarios 33
- OpenSIPS, usage scenarios 33**
- OpenSIPS-CP. *See* OpenSIPS Control Panel**
- OpenSIPS.cfg file**
 - inspection 156, 157, 158
 - modules, loading 203
 - modules, parameters 204
- OpenSIPS.cfg file, sections**
 - branch routing blocks 35
 - error routing block 35
 - failure routing blocks 35
 - global definitions 35
 - local routing blocks 35
 - main routing block 35
 - modules 35
 - modules configuration 35
 - reply routing blocks 35
 - secondary routing blocks 35
- opensips.cfg file analysis**
 - consume_credentials() function 108
 - db_check_from() function 108
 - db_check_to() function 108
 - modules, loading 106, 107
 - non-register requests 108
 - proxy_authorize() function 108
 - requests, registering 107
- OpenSIPS console 56**
- OpenSIPS Control Panel**
 - about 128
 - and SerMyAdmin, comparing 143
 - configuring 132, 133
 - installing, steps 129, 130
 - modules 128
 - Monit, installing 131
 - tools 128
 - utilities 134
- opensipsctl fifo get_statistics core command 241**
- opensipsctl fifo get_statistics net (new 1.6) command 241**
- opensipsctl fifo get_statistics pkmem (new 1.6) command 241**
- opensipsctl fifo get_statistics sl command 241**
- opensipsctl fifo get_statistics tm command 241**
- opensipsctl fifo get_statistics registrar command 241**
- opensipsctl fifo get_statistics shmем command 241**
- opensipsctl fifo get_statistics uri command 241**
- opensipsctl fifo get_statistics usrloc command 241**
- opensipsctl fifo ps command 241**
- opensipsctl fifo which command 241**
- opensipsctl shell script**
 - about 109
 - authentication, implementing 113, 114
 - enhancing 114-116
 - Intra-domain 116
 - multiple domains, managing 116, 117
 - opensipsctrlrc file 110-112
 - Outbound-to-outbound 116
 - Outbound inter-domain 116
- OpenSIPS missing BYEs issue, solving**
 - gateway and the proxy, differences 236
 - Media Proxy timeout used 237
 - RTP proxy timeout used 237
 - SIP session timers used 237
- OpenSIPS server**
 - configuring 234-236
- OpenSIPS v1.6.x**
 - directory structure 57
 - downloading 55
 - installing 55
- OpenSIPS v1.6.x, directory structure**
 - binaries (/sbin) 58
 - configuration files (etc/opensips) 57
 - modules (/lib/opensips/modules) 58
- OSI model 24**
- osipsconsole 56**
- Outbound inter-domain 116**

P

- parameters, Drouting**
 - blacklist 163
 - Force_dns 163
 - sort order 163

- password_column2 parameter 93
- password_column parameter 93
- PAT 193
- ping_nated_only parameter 204
- plain old telephony system. *See* POTS
- plaintext passwords 103
- port address translation. *See* PAT
- port parameter 65
- post restricted cone NAT 196-198
- POTS 161
- prefix() core function 191
- proto 71
- proxy_authorize(realm, table) function 93
- proxy server 10
- pseudo-variables 72
- PSTN
 - about 145
 - calls, making to 151-155
- PSTN gateway
 - Asterisk, using as 159, 160
- PSTN gateway, VoIP provider 25

Q

- QOP 102
- Quality Of Protection. *See* QOP

R

- RADIUS 232
- RE-INVITE messages
 - handling 208
- Real Time Control Protocol. *See* RTCP
- Real Time Protocol. *See* RTP
- Record-Route header 80
- record routing 79
- redirect server 10
- REGISTER authentication sequence
 - about 94
 - code snippet 96, 97
 - message, authenticating 94
 - packet capturing, ngrep used 94-96
- register permissions 148
- REGISTER requests
 - handling 206
- REGISTRAR 11

- Remote Authentication Dial in User Service.

- See* RADIUS

- reply routing blocks 35
- Request for Comment. *See* RFC
- responses
 - handling 207, 208
 - RE-INVITE messages, handling 208
 - script, routing 209-215
- restricted cone NAT 196-198
- retcode 71
- revert_uri() command 191
- rewritehostport()function 150, 158
- RFC 7
- RFC2865 232
- RFC2866 232
- RFC3261 7, 21
- RFC3581 201, 202
- RFC4028 237
- Route header 80
- route permissions 148
- route set 80
- routing
 - initial requests 79
 - Record-Route header 80
 - record routing 79
 - replies 78
 - requests 78
 - Route header 80
 - route set 80
 - sample 80, 82
 - sequential requests 80
- RTCP 22
- RTP 21
- RTP packets traversal
 - solving, Media Proxy used 202, 203
 - solving, RTP Proxy used 203
- RTP protocol
 - about 21
 - codecs 22
 - DTMF relay 22
 - Real Time Control Protocol (RTCP) 22
- RTP Proxy
 - ALGs 223
 - configuring 203
 - ICE 224
 - installing 203

- using, for Nat traversal 222
- RTP proxy, VoIP provider**
 - for Nat traversal 26
- rtpproxy_sock parameter** 204
- RTP proxy timeout** 237

S

- SBCs** 31
- script flag** 76
- scripting**
 - basics 70
 - core functions 71
 - core keywords 71, 72
 - core values 71
 - OpenSIPS 64
 - pseudo-variables 72
 - script variables 72, 73
- script variables**
 - about 72
 - arithmetic operations 73
 - string transformations 73
- SDP** 202
- SDP (RFC2327)** 17
- SDP protocol** 22, 23
- secondary routing blocks** 35
- secure SIP URI** 16
- sequential requests** 80
- SerMyAdmin**
 - about 133
 - configuring 136, 137
 - features 134
 - installing 134, 135
 - modules 133
- SerMyAdmin, tasks**
 - domain, managing 142
 - interface, customizing 142, 143
 - new user, approving 139, 140
 - new user, registering 138
 - user, managing 140-142
- server**
 - operating, as SIP proxy 13
- session border controllers.** *See* **SBCs**
- Session Description Protocol.** *See* **SDP**
- Session Initiation Protocol.** *See* **SIP**
- SHMEM** 242

- Simple traversal of UDP through NAT.** *See* **STUN**

SIP

- about 7-9
- architecture 8
- basic messages 14
- basics 8, 9
- breaking, by NAT 194
- components 10, 11
- dialog flow 15, 16
- dialogs 21
- operation theory 10
- protocol, features 7, 8
- registering process 11, 12
- trapezoid 8
- transactions 21
- URI 16
- SIP, components**
 - location server 10
 - proxy server 10
 - redirect server 10
 - UA (User Agent) 10
 - UAC (User Agent Client) 10
 - UAS (User Agent Server) 10
- SIP, header fields**
 - CALL-ID 17
 - CONTACT 17
 - CONTENT-LENGTH 17
 - CONTENT-TYPE 17
 - CSEQ 17
 - FROM 17
 - MAX-FORWARDS 17
 - TO 16
 - VIA 16
- SIP dialog** 21, 36
- SIP Express Media Server (SEMS)** 26
- SIP identity, global parameters** 67
- SIP NAT traversal**
 - challenges, solving 198
 - far-end NAT solution, implementing 201
 - issues, classifying 201
 - near-end NAT solution, implementing 198, 200
- SIP NAT traversal issues, classifying**
 - RTP protocol 201
 - SIP protocol 201

- SIPp**
 - about 245
 - installing 245, 246
- sipping_bflag parameter** 204
- sipping_from parameter** 204
- SIP protocol** 24
- SIP protocol, features**
 - call establishment 8
 - call management 8
 - user availability 7
 - user location 7
 - user parameters negotiation 7
- SIP provider big picture** 146, 147
- SIP proxy**
 - about 25
 - basic processing 36
 - server operating as 13
- SIP PSTN gateway** 145
- SIP redirect**
 - server operating as 13
- SIPSAK** 244, 245
- SIP session** 36
- SIP session timers** 237
- SIP trace** 26
- SIPTRACE, trace tools**
 - about 243
 - configuring 243
- SIP transaction** 36
- SIP tutorial**
 - Columbia University, URL 26
 - iptel website, URL 27
- SIP URI** 16
- sl_replay_error() function** 86
- sl_send_reply function** 81
- sort order parameter** 163
- standard configuration, OpenSIPS**
 - uses 88-90
- startup options, OpenSIPS**
 - c 60
 - D -E dddddd 60
 - other options 60, 61
- stateful routing** 78
- statements**
 - if-else 76
 - subroutes 77
 - switch 77
- status** 71

- stress testing tools, trace tools**
 - about 244
 - SIPp 245
 - SIPp, installing 245
 - SIPSAK 244
 - stress test, RTP signaling 249, 250
 - stress test, SIP signaling 246-248
- string transformations, script variables** 73
- STUN**
 - about 198
 - and TURN (media relay server), comparing 223
 - implementing, in OpenSIPS 200
 - symmetric NAT devices 200
 - working 199
- subroutes statement** 77
- switch statement** 77
- symmetric NAT**
 - about 197, 198
 - traversing, issues 197, 198

T

- t_check_trans(); function** 82
- t_check_trans()function** 118
- t_on_failure() function** 190
- t_relay() command** 37
- t_relay()function** 38, 86, 190
- tables, Drouting**
 - DR_GATEWAYS 163
 - DR_GROUPS 164
 - DR_GW_LISTS 165
 - DR_RULES 164
- TCP/TLS/UDP** 71
- TM** 242
- to_uri** 71
- tools**
 - FIFO commands 240, 241
- tools, VoIP provider**
 - monitoring 26
- trace tools**
 - about 243
 - monitoring 254
 - SIPTRACE 243
 - SIPTRACE, configuring 243
 - stress testing tools 244
 - Wireshark 250-254

transactions, SIP 21

Traversal of UDP over Relay NAT). *See*

TURN

troubleshooting

client unable to register 89, 90

daemon does not start 89

too many connections 90

TURN

and STUN, comparing 223

U

UA. *See* **user agent**

UAC 10

UACs 157

UAS 10

uri 71

Uri permissions 149

use_domain parameter 93

user_column parameter 93

user agent

about 10

Contact List, updating 12

current contact list, requesting for 12

new registration 12

registration, cancelling 12

registering, process 12

unsuccessful registration 12

User Agent Client. *See* **UAC**

user agent clients. *See* **UACs**

User Agent Server. *See* **UAS**

usr_preference table 75

usrloc module 70

V

Voicemail 178

VoIP provider

AAA (Authentication, Authorization and
Accounting) 26

about 24

accounting, automating 227

accounting, configuration 227

accounting, MySQL used 228

accounting, RADIUS used 232

administration 25

architecture 226

Call Detail Records (CDRs), generating 231

media proxy, for Nat traversal 26

media server 26

multileg accounting 228

opensips.cfg file analysis 229

provisioning portal 25

PSTN gateway 25

RTP proxy, for Nat traversal 26

SIP Express Media Server (MES), features
26

SIP proxy 25

tools, monitoring 26

user administration 25

W

WANs 22

WeSIP 32

Wide Area Networks. *See* **WANs**

Wireshark 250-254

www_authorize(realm, table) function 93

X

XLOG() function 190



Thank you for buying
**Building Telephony Systems
with OpenSIPS 1.6**

Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing Building Telephony Systems with OpenSIPS 1.6, Packt will have given some of the money received to the OpenSIPS project.

In the long term, we see ourselves and you – customers and readers of our books – as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

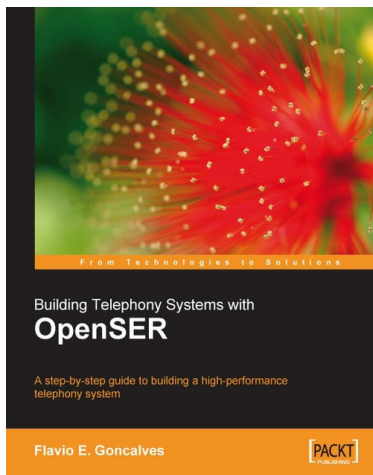
We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.PacktPub.com.



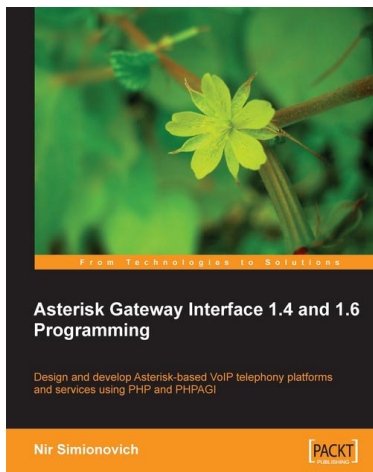
Building Telephony Systems with OpenSER

ISBN: 978-1-847193-73-5

Paperback: 324 pages

A step-by-step guide to building a high performance Telephony System

1. Install, configure, and troubleshoot OpenSER
2. Use OpenSER to build next generation VOIP networks from scratch
3. Learn and understand SIP Protocol and its functionality



Asterisk Gateway Interface 1.4 and 1.6 Programming

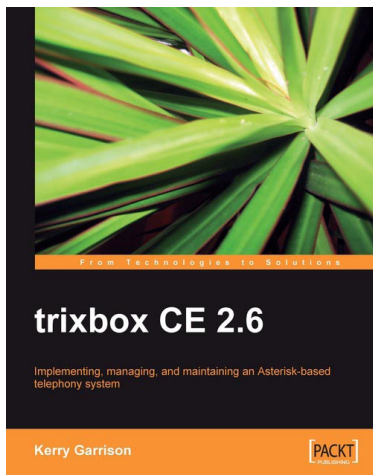
ISBN: 978-1-847194-46-6

Paperback: 220 pages

Design and develop Asterisk-based VoIP telephony platforms and services using PHP and PHPAGI

1. Develop voice-enabled applications utilizing the collective power of Asterisk, PHP, and the PHPAGI class library
2. Learn basic elements of a FastAGI server utilizing PHP and PHPAGI
3. Develop new Voice 2.0 mash ups using the Asterisk Manager

Please check www.PacktPub.com for information on our titles



trixbox CE 2.6

ISBN: 978-1-847192-99-8

Paperback: 344 pages

Implementing, managing, and maintaining an Asterisk-based telephony system

1. Install and configure a complete VoIP and telephonic system of your own; even if this is your first time using trixbox
2. In-depth troubleshooting and maintenance
3. Packed with real-world examples and case studies along with useful screenshots and diagrams



FreePBX 2.5 Powerful Telephony Solutions

ISBN: 978-1-847194-72-5

Paperback: 292 pages

Configure, deploy, and maintain an enterprise-class VoIP PBX

1. Fully configure an Asterisk PBX without editing the individual text-based configuration files
2. Add enterprise-class features such as voicemail, least-cost routing, and digital receptionists to your system
3. Secure your PBX against intrusion by managing MySQL passwords, FreePBX administrative accounts, account permissions, and unauthenticated calls

Please check www.PacktPub.com for information on our titles