SER - Getting Started

ONsip.org
Paul Hazlett <phazlett@gmail.com>
Simon Miles <simon@SystemsRM.co.uk>
Greger V. Teigre <greger@teigre.com>

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

SER - Getting Started

by ONsip.org, Paul Hazlett, Simon Miles, and Greger V. Teigre

The guide that tells you all about getting SER to work.Based upon real life examples, this guide leads you through the steps necessary to build a fully working SIP environment. A Step by Step approach is taken to help the reader understand the intricacies of SER.

Disclaimer :-

The authors of this document do not warrant or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed. Any consequences or results achieved directly or indirectly by this documents configuration files or information are entirely your responsibility.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table of Contents

1. About This Issue ... 1
2. About This Document ... 2
3. SER or OpenSER - A Brief Comparison ... 3
    - History of SER ... 3
    - OpenSER ... 3
    - SER and OpenSER development ... 3
        - Understanding naming of versions ... 4
        - Compatibility now and in the future ... 4
    - Choosing between SER and OpenSER ... 5
    - ONsip.org Getting Started and SER vs OpenSER ... 5
4. Getting Started What is SIP and how does it work? ... 7
    - SER Architecture and ser.cfg ... 7
        - Core and Modules ... 7
        - ser.cfgs Seven Sections ... 7
        - Transactions, Dialogs, and Sessions ... 7
        - Understanding Message Processing in ser.cfg ... 8
    - Stateful vs. stateless ... 9
    - Understanding SIP and RTP ... 9
    - Back-end applications and B2BUA ... 9
    - NAT, STUN, and RTP proxy ... 10
        - Registration behind NAT ... 10
        - INVITEs behind NAT ... 11
        - STUN ... 12
        - Other non-ser NAT traversal techniques ... 12
    - URI, R-URI, and Branches ... 12
5. Reference Design ... 13
6. Hello World ser.cfg ... 15
    - ser.cfg Listing ... 15
    - Hello World ser.cfg Analysis ... 17
    - Using The Hello World ser.cfg Configuration ... 21
7. SER - Adding Authentication and MySQL ... 23
    - MySQL ser.cfg Listing ... 24
    - Authenticating ser.cfg Analysis ... 26
    - Using the Authenticating ser.cfg Proxy ... 28
8. Handling of NAT ... 30
    - Handling of NAT using Mediaproxy ... 30
        - Mediaproxy Transparent NAT Traversal ser.cfg Analysis ... 35
        - Using the Mediaproxy Transparent NAT Traversal ser.cfg ... 40
    - Handling of NAT using RTPproxy ... 40
        - RTPproxy Transparent NAT Traversal ser.cfg Analysis ... 45
9. PSTN Gateway Connectivity ... 48
    - PSTN Gateway Connectivity ser.cfg Analysis ... 54
    - Using the PSTN Gateway Connectivity ser.cfg Example ... 56
10. Call Forwarding ser.cfg ... 58
    - Call Forwarding ser.cfg Analysis ... 66
    - Using the Call Forwarding ser.cfg Example ... 67
11. Appendix - How to download and configure the latest version of SER ... 68
    - Downloading the Latest SER Source Code ... 68
    - Making the SER Binaries and installing ... 68
    - Installing MediaProxy ... 69
    - Configuring the system ... 69
        - Init.d/ser ... 69
        - Init.d/mediaproxy ... 71
    - Supporting MySQL ... 72
    - Debugging Tips ... 73

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Capture of SIP Messages 73
Generate Debug Information 73
12. Appendix The Call Processing Language (CPL) 74

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

List of Figures

5.1. Complete Reference Design 13
5.2. 'Hello World' Reference Design 14
7.1. Reference Design Plus MySQL 23
8.1. Reference Design Plus NAT 30
9.1. Reference Design - PSTN 48

render X
XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 1. About This Issue

This issue is primarily a bug-fix issue. The format of the document has been changed to docbook in order to simplify maintainance by several authors, as well as to support auto-generation of html and Adobe Acrobat (pdf) formats of the document. NAT-handling has been improved with a test around RTP proxy enforcement to avoid doing it twice and the handling of an issue with ACKs (without Route header) when using this setup together with Asterisk. Also, the handling of loose routed messages has fixed a security issued where pre-setting a Route header pointing to the PSTN gateway would bypass authentication for INVITE and REFER messages.

Note that this issue is compatible with version 0.9.x of SER and is most likely NOT to work for the latest OpenSER (but probably the 0.9.x of OpenSER). The next issue will be compatible with 0.10.x and new functionality from this forth-coming release will be introduced.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

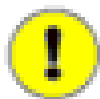
Chapter 2. About This Document

This document will not tell you the basics about SIP, IP Telephony, how SER relates to other open source software, and why you should use SER. We assume you have decided to use SER for your SIP-related needs, and you now need to get up to speed on the concepts and how to realize most of the core functionality that SER can offer. This document describes how to set up SER to function as a SIP Proxy, SIP Registrar and SIP Redirect Server. It is based upon version 0.9.3 of SER and section 10 Appendix - How to download and configure the latest version of tells you how to download this version and install.

This section covers the most important concepts that you must understand in order to get SER up and running and start adapting it to your needs and that from experience many people have problems understanding. However, this document is NOT a manual. The official SER manual can be found at http://www.iptel.org/ser/admin.html

All ser.cfg configuration files can be downloaded from http://www.onsip.org/ under the Downloads section. Onsip.org has been started to facilitate a sharing community around SER and was initiated as part of the effort to write the first version of this Getting Started document. Please register and become a contributor to the success of SER!

NOTE: It is important to understand that SER is a SIP router. It only processes SIP messages. All other IP telephony functionality, such as voice mail, can only be implemented by using external applications.



Important

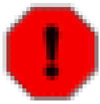
Feedback from the community is an essential element in development of open projects. To facilitate this, readers are encouraged to provide feedback and contributions to the following address :-

Feedback: gettingstarted@onsip.org [mailto:gettingstarted@onsip.org]

Contributions:contributions@onsip.org [mailto:contributions@onsip.org]

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 3. SER or OpenSER - A Brief Comparison



Warning

DISCLAIMER: This section is written by piecing together information from various sources and may be inaccurate. However, the authors have tried to verify the accuracy to the extent possible and encourage feedback to gettingstarted@onsip.org.

History of SER

SER was developed by a team of developers employed by Fraunhofer Fokus, a German research institute. The iptel.org project was to build a website for Voice over IP information and a free Voice over IP service. SIP Express Router (SER) was developed as a part of this effort, lead by Jiri Kuthan. SER was offered as open source (GNU Public License, GPL) and the iptel.org website is still the entry point with SER information, as well as SIP tutorials and other related resources (though now not very actively maintained).

As another result of the iptel.org project, Fraunhofer Fokus spun off iptelorg.com as a commercial venture to further develop SER (for both open source and commercial purposes) and to offer services and software packages and support based on the iptel.org projects developed code (which by the way are more than just SER. See http://www.iptel.org/products/) Iptelorg.com got the main control and responsibility for developing SER, the open source SIP server. Jiri Kuthan and Jan Janak (Chief Software Architect) are both a part of this group. Andrei Pelinescu Onciul is another name that you can see on the serusers mailing list.

Some of the other former employees in the Fraunhofer iptel.org project launched another commercial SER-based venture called Voice System (http://www.voice-system.ro/). Daniel-Constantin Mierla and Bogdan-Andrei Iancu are two well-known names on the serusers mailing list.

This is how those known as the core developers were split in two groups. However, they all participated in the development of what we know as SER. In addition, other companies (like AG Projects responsible for mediaproxy) and individual developers joinded the group of developers. At the time of writing this, SER has 25 registered developers.

OpenSER

On June 14, 2005, Voice System announced that they had created OpenSER. This is the reason stated on the OpenSER website (http://www.openser.org/): The reason for this new venture is the lack of progressing and contributions to the SER project from the other SER team members as well as the reticience to new contributions from project's community members.

The Voice System SER developers felt that iptelorg.com, who made the final decisions on releases and code, had failed to take in new code contributions and also been too slow in releasing new versions of SER. The announcement made a stir on the serusers mailing list. The general feeling was that new features are good, it had been difficult to get code into the code versioning system (CVS) due to slow response from developers who had to accept the code, but most people felt that it would be easier for the community if both the stability and quality assurance found in SER could be combined in the same project with the OpenSER goals.

SER and OpenSER development

We now know that both projects continued separately. What does it mean to us, the users? First of all, you need to decide whether to go for SER or OpenSER. Deciding is based on knowledge, so first a few words about the organization of SER and OpenSER developments.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Understanding naming of versions

SERs source code is hosted in a CVS at cvs.berlios.de. The versioning of SER has confused many. The reason is that the versioning is based on CVS terminology. For example, the latest stable release of SER is 0.9.3, but in the CVS, this stable release has the branch name rel_0_9_0. A branch can be named anything, for example rel_stable_to_be. rel_0_9_0 means loosely "the branch that will be the next stable releasewith version 0.9.something". A less confusing branch name would probably berel_0_9_x.

Once a branch has been created it has a life on its own and the developers can continue to develop the main code branch, also called the HEAD or the trunk. This is important to know, because once a new branch has been created and is destined to become a stable release, no new features will be added, only bugs will be fixed. Any new features are added to the trunk. Currently, the trunk is also referred to as 0.10.x or 0.10.0 because the next main release will be 0.10.something.

This means that even though you have downloaded a package of 0.9.3, the rel_0_9_0 branch in the CVS may later be updated, but only with bug fixes. In order to get these bug fixes, you either have to download the code from the CVS repository or you have to wait until a new updated package is created.

The source package called 0.9.3 at http://ONsip.org/ is a package with the source files from the stable branch rel_0_9_0 at the date specified in the description field and in the README.ONSIP file. You can run the update_from_cvs script to get the latest changes from CVS.We also update the package once in a while where this update has been done.

If you are adventurous, you can download the trunk/head, but it can at times be impossible to compile, because developers add new functionality that may break things they didn't foresee (even though it compiled on their machine, other things may influence compilation). Also, added functionality may break other functionality, so even though SER compiles, you may end up with problems with your ser.cfg because the developer haven't tested exactly that scenario. These problems should be reported at http://bugs.sip-router.org [http://bugs.sip-router.org/]

When Voice System announced OpenSER, they took SER 0.9.3, added (backported) many of the features found in the SER CVS trunk (0.10.x), and added additional features not found in SER CVS trunk. They called this release OpenSER 0.9.4, which was the first OpenSER release. Since then, they have released OpenSER 0.9.5.

Compatibility now and in the future

Currently, OpenSER can use SER configuration files (ser.cfg), but not the other way around. This is because OpenSER have added new commands and in some cases new syntax. Now, that sounds good. The safe bet would be to go for OpenSER, right? Well, for now. The development and maintenance of SER and OpenSER can become complex. First of all, the developers of OpenSER have stated that they will continue to contribute to the SER development in addition to OpenSER. We still dont know what this means. Will all the OpenSER features be introduced in SERs CVS? This means a lot of double work and other SER developers may have introduced features that are not compatible with OpenSERs.

Also, it is highly likely that features found in OpenSER will find its way into SER. SERs developers may for some technical reason decide to implement the feature in a different way. This means that the ser.cfg used for SER can no longer be used by OpenSER.

Already, OpenSER has a different database format than SER. This means that migrating user data from one to another can be a problematic task. These differences are likely to increase.

And finally, it seems that most of the developments that are done by SER developers are ported to OpenSER by the OpenSER developers. At the time of writing this, OpenSER has six (6) registered developers, while SER has 25. Only time will show how interoperable the two projects will be.

As a side note: One of the authors of this document is the maintainer of a relatively new SER module called experimental. This CVS module is currently only available in the trunk/HEAD and contains SER modules and code that has not yet found its way into the CVS. The modules can thus be tried out and feedback gathered before they

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

are introduced into the CVS trunk. The current modules are TLS support, SIP Path extension, and an Oracle database back-end.

Choosing between SER and OpenSER

Many of you have probably skipped right to this section. Well, you can still read the above when you are finished with this.

The table below has listed some criteria and how SER and OpenSER address each. You should find the criteria important for you and decide based on that. Please contact gettingstarted@onsip.org [mailto:gettingstarted@onsip.org] if you have other criteria you feel should be present.

Criteria	SER	OpenSER
Release frequency/new features	Long release cycles, you need to use the experimental module or the HEAD version to get new features	Frequently minor releases (0.a.x) with new features available. Major changes available in major releases (0.x), but no history of this yet
Large user community	Well-established community with the serusers@iptel.org [mailto:serusers@iptel.org] mailing list as the main exchange. Many active contributors. High volume and questions must be well writton out to receive an answer	Voice System/OpenSER core developers are very active and most questions are answered.
Release quality/stability	The long release cycles and large community reduces the risk of running into bugs if you use a stable release.	The high focus on new features and frequent releases increases the likeliness of bugs in a stable release. The officially stated policy for stability is release often
Documentation	Administrators manual at iptel.org is outdated, but still useful. The main source of module documentation is in the README files of each module. The OpenSER 0.9.x documentation can mostly be used. And this document is a good starter!	Very high focus on documentation. The modules README files, as well as some tutorials can be found at http://openser.org/docs/ A WIKI has also been created. And this document is a good starter!
Support for various architectures	The person responsible for compiling on architectures works on SER	Hard to say, probably the OpenSER developers will try to merge in architecture changes from SER CVS
Development participation	Code repository at:http://developer.berlios.de/projects/ser/ Patches (and bugs) to existing code should be submitted to http://bugs.sip-router.org/ or directly to the developer d in README). Modules or larger code extensions can be submitted to greger@teigre.com [mailto:greger@teigre.com] (maintainer of experimental module)	Code repository at http://sourceforge.net/projects/openser/ Send email to devel@openser.org [mailto:devel@openser.org] (note! mailing list)
Commercial support	Iptelorg.com sells commercial licenses of enhanced SER. Many consultants are present on the serusers@iptel.org [mailto:serusers@iptel.org] mailing list	Voice System offers software and consulting. Availability of other consultants is unknown.

ONsip.org Getting Started and SER vs OpenSER

Maintaining and further developing the Getting Started document and the corresponding configuration files is a major undertaking. We simply do not have the capacity to track the differences between SER and OpenSER and

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

update the documentation accordingly. Thus, the most rationale decision is to focus on SER only and just hope that OpenSER will continue to be able to use SER configuration files

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 4. Getting Started What is SIP and how does it work?

SER Architecture and ser.cfg

Core and Modules

SER is built around a processing core that receives SIP messages and enables the basic functionality of handling SIP messages. Most of SERs functionality is offered through its modules, much like the Apache web server. By having a modular architecture, SER is able to have a core that is very small, fast, and stable. SER modules expose functionality that can be utilized in the SER configuration file, ser.cfg. The ser.cfg configuration file controls which modules shall be loaded and defines how the modules shall behave by setting module variables. You can think of the ser.cfg file as the brains of the SIP router.

ser.cfgs Seven Sections

ser.cfg has seven main logical sections:

1. Global Definitions Section. This portion of ser.cfg usually contains the IP address and port to listen on, debug level, etc. Settings in this section affect the SER daemon itself;

2. Modules Section. This section contains a list of external libraries that are needed to expose functionality not provided by the core as noted above. These modules are shared object .so files and are loaded with the load-module command;

3. Module Configuration Section. Many of the external libraries specified in the Modules Section need to have parameters set for the module to function properly. These module parameters are set by use of the modparam command, which takes this form: modparam(module_name, module_parameter, parameter_value)

4. Main Route Block. The main route block is analogous to a C programs main function. This is the entry point of processing a SIP message and controls how each received message is handled;

5. Secondary Route Blocks. In addition to the main route block, ser.cfg may contain additional route blocks that can be called from the main route block or from other secondary route blocks. A secondary route block is analogous to a subroutine;

6. Reply Route Block. Optional reply route blocks may be utilized to handle replies to SIP messages. Most often these are OK messages;

7. Failure Route Block. Optional failure route blocks may be used when special processing is needed to handle failure conditions such as a busy or timeout;

It is important to understand the SIP protocol and the various types of messages that are used in SIP signalling. Of course, by following the instructions in this document, you will get a working setup. However, before starting to tweak and adapt to your needs, we recommend that you do yourself a favour and read up on SIP. Please refer to http://www.iptel.org/ser/doc/sip_intro/sip_introduction.html as a good introduction. http://www.iptel.org/sip/sip-tutorial.pdf provides more depth. The official SIP RFC can be found at http://www.ietf.org/rfc/rfc3261.txt for those interested.

Transactions, Dialogs, and Sessions

In order to understand ser.cfg properly, you need to understand three SIP concepts:

render X

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

1. SIP transaction: A SIP message (and any resends) and its direct (most often immediate) response (ex. User agent sends REGISTER to SER and receives OK);

2. SIP dialog: A relationship between (at least) two SIP phones that exists for some time (ex. Dialog is established with an INVITE message and ended by a BYE message);

3. Session: An actual media stream of audio between the SIP phones;

These concepts are hard to understand and you may want to revisit this section when you have read later sections or studied the ser.cfg in this document. The concepts are used in the sections below to explain some things commonly confused.

NOTE: If you look at a SIP message, you can identify messages in a particular SIP transaction by looking at the Cseq number in the header. Each SIP dialog will have a Call-Id header (as well as one ID called a tag for each peer in the dialog)

Understanding Message Processing in ser.cfg

You can think of ser.cfg as a script that is executed every time a new SIP message is received. For example, a user agent (UA) (Johns SIP phone) wanting to INVITE another UA (Joans SIP phone) to a conversation (John makes a call to Joan). John sends an INVITE SIP message to SER and ser.cfgs main route{} block will start at the top and execute through the commands found there.

The processing continues until it reaches a point where processing is finished and a decision is made whether to send the INVITE to Joan (using the t_relay() command), to send John a reply with an error (using sl_send_reply()), or just drop the whole INVITE (by reaching the end of the main route or break;), which, of course, is not recommended.

Joan will answer the INVITE with an OK message. The OK is a direct response to the initial INVITE and this message is handled by the last section in an on_reply_route[x]. If Joan didnt respond, or responded with an error (busy, etc), the failure_route[x] is called.

Finally, John will send an ACK to tell Joan that everything was received and accepted.

NOTE 1: The described behaviour is dependent on using t_relay() in ser.cfg. Your SER is then said to be transaction stateful (see also next section)

NOTE 2: An INVITE dialogue also includes provisional responses (trying, your call is important to us) before the OK, but we will not concern ourselves with these for simplicity.

So, how is all this handled in ser.cfg? All SIP messages starting a new SIP transaction will enter the top of the main route{}. In the above, Johns INVITE starts a transaction that is answered with OK from Joan.

You have a lot of freedom in how SIP messages are handled in ser.cfg. For example, to record that Joan is online, you use the save(location) function for all REGISTER messages from Joans IP phone. A call to lookup(location) will retrieve where Joans IP phone can be found so that that a call can be made. Also, very limited info about Joans phone can be stored in the form of flags using the setflags() function. (From version 0.9.0 there is also support for attribute-value pairs that can be loaded/stored for a given subscriber, but more on this later).

The consequence of ser.cfg as a routing logic script is that you have to make sure that each SIP message type is handled correctly (flows through the script in a correct fashion) and that each possible response in a transaction is appropriately handled by reply or failure routes to realize what you want (forward on busy etc). This can be quite complex and opens up for many possible errors. Especially when changes to ser.cfg easily affect more than the messages you intended to target. This is usually the root cause of SER administrators that question whether SER is RFC3261 compliant or not. SER is RFC3261 compliant from point of view of can it properly process any particular SIP message, however any seemingly harmless error in ser.cfg can have dramatic impact on the SIP router and cause SER to deviate from RFC3261 compliance.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This document presents a reference design with a corresponding ser.cfg to enable you to quickly set up SER to do what most people would like SER to do.

Stateful vs. stateless

An often misunderstood concept is stateful vs. stateless processing. The description of the INVITE SIP transaction above is an example of stateful processing. This means that SER will know that the OK belongs to the initial INVITE and you will be able to handle the OK in an onreply_route[x] block. With stateless processing (or simply forwarding), each message in the dialogue is handled with no context. Stateless forwarding is used for simple processing of SIP messages like load distribution.

In order to implement any advanced functionality like call accounting, forward on busy, voicemail, and other functionality in this documents reference setup, you will need to use stateful processing. Each SIP transaction will be kept in SERs memory so that any replies, failures, or retransmissions can be recognized. One consequence is that when using t_relay() for a SIP message, SER will recognize if a new INVITE message is a resend and will act accordingly. If stateless processing is used, the resent INVITE will just be forwarded as if it was the first.

The confusion arises because this stateful processing is per SIP transaction, not per SIP dialog (or an actual phone call)! A phone call (SIP dialog) consists of several transactions, and SER does not keep information about transactions throughout a particular phone call. The consequence is that SER cannot terminate an on-going call (it doesnt know that the call is on-going), nor can SER calculate the length of an ended call (accounting). However, SER can store when an INVITE (or ACK) and a BYE message are received and record this info together with the Call-Id. A billing application can then match the INVITE with the BYE and calculate the length of the call.

Understanding SIP and RTP

In order to understand the subsequent sections, you must understand a few things about SIP and RTP. First of all, SIP is a signalling protocol handling the call control like inviting to a call, cancel (hang up while ringing), hanging up after ended call and so on. SIP messaging can be done directly from user agent to user agent, but often a SIP server is needed for one user agent to find another.

When a SIP server like SER receives a message, it can decide that it wants to stay in the loop or not. If not, SER will provide the user agents with the information they need to contact each other and then SIP messages will go directly between the two user agents.

If SER wants to stay in the loop, for example to make sure that a BYE message is received for accounting, SER must insert a Route header in the SIP message (using the record_route() function) to tell everybody that it wants to participate. In order for this to work, SER and potentially other SIP servers who participate must do what is called loose routing. A bit simplified, it means that SIP messages should not be sent directly to the user agent, but rather indirectly via all who have put a Route header in the SIP message (to check for such a recorded route, the loose_route() function is used).

SIP can also include additional information, for example related to how to set up a call with an audio or video stream (called SDP, Session Description Protocol). The SDP information will result in one or more RTP streams (or sessions) to be set up, normally directly between the two user agents. SER will NEVER participate in the RTP stream. RTP streams are by nature bandwidth and processing intensive. However, as we will describe later, SER can make sure that a third party application like a B2BUA or RTP Proxy can become the middle man.

Finally, the Real-Time Control Protocol (RTCP) communicates information about the RTP streams between the user agents (RTCP will either use the specified RTP port + 1 or the port indicated in the SDP message).

Back-end applications and B2BUA

We have learned that SER does not keep the state of a dialog, but just transfers SIP messages (as part of transactions) between two user agents. The consequence is that SER cannot be a peer in a dialog.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

But, if you want to play a voice prompt saying that the user is not available or provide voicemail capabilities, you will need something that can act as a user agent and be a peer in a SIP dialog, so that a session is actually set up between the calling user agent and that something playing the voice prompt. Modules or third party applications can provide this back-end user agent functionality, and SER will be in the middle. Again, SER is used in its stateful processing to enable these back-end applications.

NOTE: To add to the confusion, when SER is used in stateful processing to enable back-end user agent functionality, it is said to act as a stateful user agent server.

Also, if you want to implement a pre-paid service, you have a problem because SER cannot terminate the call when no money is left! In this scenario you need a third party in the SIP dialog (and quite possibly in the session). This third party will act as the middle man both in the SIP messages and in the audio. Thus, each user agent will only talk to the middle man and not know about the remote user agent. This middle man is what you probably have seen referenced as a B2BUA (Back-to-Back User Agent) on the serusers mailing list. A B2BUA can handle SIP messaging only or both SIP and RTP.

NAT, STUN, and RTP proxy

One of the things causing the most problems is one-way audio or no audio when dealing with user agents behind NAT devices that attempt to communicate with user agents on the public internet or behind a different NAT device. These NAT devices can be ADSL routers, firewalls, wireless LAN routers, etc. In order to understand NAT and RTP proxying, you must understand what happens when a user agent registers with a SIP Registrar and when a call is made.

In the following sections, functions from the nathelper module will be used as examples. Mediaproxy will be covered later in this document.

NOTE: SER has two different offerings for dealing with NAT and RTP proxying, namely rtpproxy and mediaproxy. These two solutions work independently of each other and are covered else where in this document. Much confusion has been caused by these two NAT solutions. Also, the nathelper module noted above is generally considered part of the rtpproxy solution, although you can use its exposed functions with mediaproxy.

Registration behind NAT

When a user agent contacts SER with a REGISTER message, the user agent will only see its private IP address behind the NAT (ex. 192.0.2.13). Thus, it will believe that the correct contact information is myself@192.0.2.13 [mailto:myself@192.168.1.13]:5060 and will include that in the REGISTER message (port 5060 is assumed to be the port the user agent is listening on). Of course, nobody can reach myself@192.0.2.13 [mailto:myself@192.168.1.13]:5060 on the Internet as this address is valid only behind the users NAT device.

SER will receive a REGISTER message from a.b.c.d:f where a.b.c.d is the public IP address of the NAT and f is the to port allocated by the NAT to this communication just initiated. SER will be able to respond to a.b.c.d:f and reach the user agent and must therefore register this address instead of the one presented by the user agent. Nathelpers function for testing if the user agent is behind NAT is called nat_uac_test(). A parameter can be specified to indicate which tests should be done. We recommend using 19 (all tests). A description of the various tests is beyond the scope of this document.

Also, SER must record whether the user agent is behind NAT in order to properly process later messages. This is done by setting a flag for the UA using the setflag() function. This flag will be available for testing for both caller and callee.

Nathelper provides the fix_nated_contact() function for re-writing the Contact header. This action is not strictly according to the RFC, so nathelper (>0.9.0) has another function fix_nated_register() that will only register the correct address and port without rewriting the Contact header.

NOTE: Section 10.3 of RFC3261 says you must not alter the contact header while processing REGISTER messages, therefore, you should never use fix_nated_contact() while processing REGISTER requests. Instead, always use fix_nated_register() and use fix_nated_contact() for all other message types.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Also, the core function force_rport() is often used for all NATed messages to ensure that the Via headers (used for matching transactions etc) include the remote port number. This way the response in the transaction is sent back to the correct port. This only works for clients that support symmetric signalling (listen on the same port as they send). However, most clients are now symmetric.

An extra problem is that the NAT device will only reserve port f for a certain time, so if nothing more happens, the NAT will close the port and ser will not be able to contact the user agent. This is solved by either the user agent sending regular messages to SER or SER must send regular messages to a.b.c.d:f (keep-alive).

INVITEs behind NAT

When an INVITE message is sent from a user agent, it will contain an SDP (Session Description Protocol) payload (read attachment or content). This payload describes various things about the user agent, ex. what type of sessions it supports, and where the party being called can reach the caller. As above, this information will for example be 192.0.2.13:23767. 23767 is the port the user agent has allocated and where it will listen for the actual sound (Real Time Protocol/RTP) for the session. At the time of the INVITE, there is no corresponding port on the NAT (as no RTP has yet been sent). The related SDP lines in the INVITE would look like this (c=contact, m=media):

c=IN IP4 192.0.2.13.

m=audio 23767 RTP/AVP 0 101.

In addition, the contact information will, as for the REGISTER message, be wrong. SER must change the contact information to the public address a.b.c.d:f as for the REGISTER message. With nathelper, this is done by calling fix_nated_contact(). Other transaction-starting messages like ACK, CANCEL, and BYE should also have the contact header fixed.

For the audio, SER can only do three things with the INVITE before forwarding to the user agent being called:

1. Add an SDP command direction:active to the SDP content
2. Change the c= line to a.b.c.d
3. Force RTP to go through a proxy by changing the c-line to c=IN IP4 address-of-proxy and the m-line to m=audio port-on-proxy RTP/AVP 0 101.

#1 is a way to tell the called user agent that it should wait until it receives audio and then just assume that it should send its own audio back to the same address (this is called symmetric RTP). This only works for user agents with a public IP or with a known RTP port on the NAT (for example by using STUN). If both are behind NAT, both will wait for the other to start and nothing will happen. However, this can be a good method for PSTN gateways with public addresses. You do this with nathelper by calling fix_nated_sdp("1").

#2 is basically the same as #1, but the called user agent gets some more information. You can do this with nathelper by calling fix_nated_sdp("2"). You can do both #1 and #2 at the same time by specifying 3.

#3 means that you need to set up a separate daemon called an RTP proxy (with a public IP address) that both user agents can send their audio to. You thus add an extra step (which can add latency in the audio) and for each call you need to handle 2 x required bandwidth for the codec the user agents are using (ex. 88 Kbps for G.711). With nathelper, you call force_rtp_proxy().

Please note that the calling user agent can have a public IP address, while the called user agent is behind a NAT. The OK returned by the called user agent will have to be manipulated as above and thus NAT handling code must also be included in the onreply_route[x] part of ser.cfg. The called user agent also has the flag set as described in the REGISTER section above (setflag). So, in the onreply route one can check whether the called user agent is behind NAT. If so, the contact header should be changed and (if desired) RTP proxying enforced.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

STUN

Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs) also known as STUN…

STUN is a protocol for the user agent to discover its own public IP address. You will need to install a STUN server (not part of SER). The user agent will then try to change the contact and SDP information itself. When STUN is used by a user agent, SER does not have to do any rewriting as described in previous section. The exception is when a user agent is behind a symmetric NAT (the different types of NAT is beyond the scope of this document). In this situation the user agent should NOT try to rewrite as the STUN protocol will provide the wrong port. SER can then be used to rewrite contact and SDP information for these situations. For outgoing calls from a symmetric NAT, the direction:active rewrite can be used, but calls between user agents behind symmetric NATs must always be handled with an RTP Proxy server.

WARNING: Some user agents have faulty implementations of STUN. If STUN then has a wrong conclusion, you may end up with one-way audio (or no audio) or calls that are disconnected. Some situations can actually be detected by SERs NAT detection tests (by comparing the senders source address/port with the contact header in the INVITE, nathelper test flag 16), but it is recommended to test new user agents in different NAT scenarios to make sure that a combined.

Other non-ser NAT traversal techniques

Some NATs and firewalls have a built-in SIP Traversal mechanism, such as Cisco 3600 routers with IOS version 12.3(9). Such a capability is often referred to as an Application Level Gateway (ALG). A SIP ALG will rewrite all SIP messages transparently (both ways). If this is done correctly, both the user agent and SER will be happy. If not, nobody is happy and you either have to turn off the ALG in the router or you have to use SER on another SIP port (like 5065). However, you will then get other NAT-related problems. The fix-faulty-ALG-problem is not really possible to do in SER 0.9.0, but the next version will introduce new features designed to address this issue.

A second variant is Session Border Controllers (SBC). These are difficult to define as a group as they can do many things, but most often they will act as a full B2BUA (terminate the SIP and RTP sessions for both user agents so that both only see the SBC). SBCs are most often used by service providers to control the edge of their networks.

URI, R-URI, and Branches

The URI, Uniform Resource Identifier, is used to identify a user uniquely. The URI is used as an address when INVITEing a party to a call. A typical SIP URI is in the form sip:username@domain.com. The original URI found in the first line of a SIP message is called the request-URI (R-URI). This uri can be referred to in ser.cfg using uri (ex. if(uri=~sip:username@.*))

The uri can be manipulated throughout ser.cfg using many different functions from different modules. For example, lookup(location) will take the uri as changed through uri manipulations, look it up in the location database and rewrite the uri to the correct contact information (including @ipaddress:port). When t_relay() is called, the URI in it final form after transformations will be used as the target URI.

Some functions, like lookup(location) can append branches to the target set of URIs. This means that when t_relay() is called, the INVITE message will be duplicated to all the URIs in the target set. Note that the uri command will still refer to the request-URI in its (potentially) transformed form. The core command revert_uri() will replace the current target URI back to the original request-URI.

NOTE: A backported version of xlog (to 0.9.x) has new parameters that can be used to print the full target set. The backported version can be found on http://www.onsip.org/

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

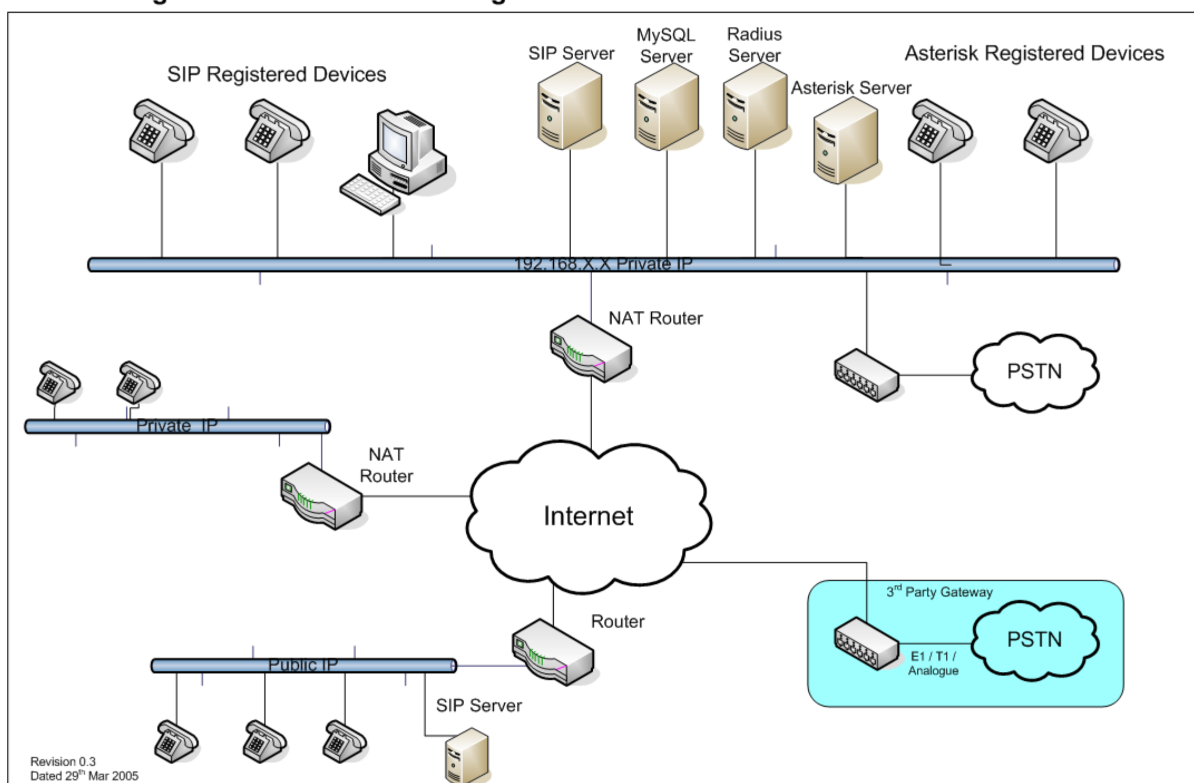
Chapter 5. Reference Design

This document will help you understand how SER can be configured for a number of network designs and functional examples. We will start with a very simple hello world example and then build up to a complex network with both SER and asterisk servers together with firewalls and NAT routers.

To help give you an idea of the type of network we will eventually cover and therefore understand the power and flexibility of SER, the diagram below shows our complete reference design.

Figure 5.1. Complete Reference Design

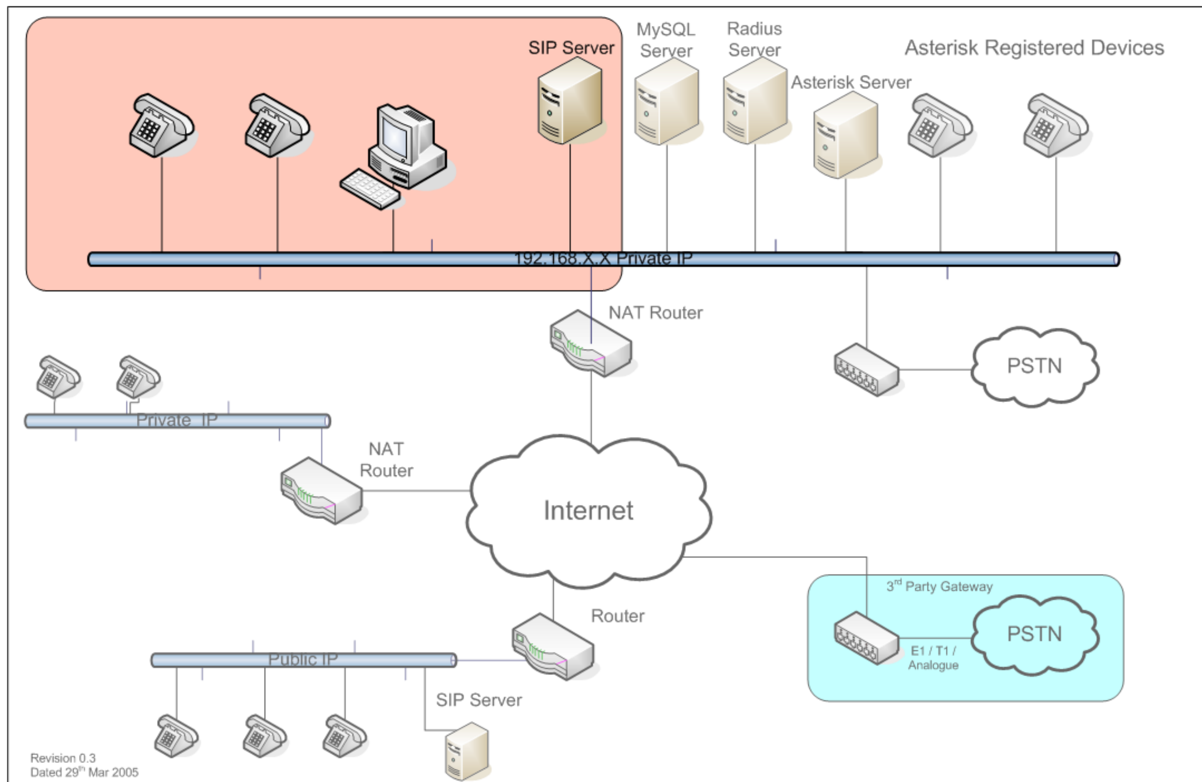
SIP Getting Started – Reference Design



But first let us start with our hello world design. Based upon our reference design, we will just take a standalone LAN segment that has a single SIP server together with a few SIP phones. .

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Figure 5.2. 'Hello World' Reference Design



The area in orange highlights this Hello World design and we will show how to configure SER to perform the following functions:

1. Registration of devices (UAs) to the sip server without any authentication
2. Establishment of a sip to sip call using INVITE messages

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 6. Hello World ser.cfg

It is a very good idea to start SER with a basic Hello World ser.cfg to ensure that the SIP server is working.

What this ser.cfg will do:

1. Establish a SIP server on your internal LAN
2. Allow you to connect IP phones on your internal LAN to SER
3. Make calls between the IP phones on the LAN

What this ser.cfg will NOT do:

1. There is no authentication for the IP telephones nor any database support. These will be added later.
2. there is no support for PSTN, the phones connected to the LAN can only communicate between themselves.

ser.cfg Listing

Listed below is the Hello Word configuration to support the simple configuration above. You can download a version of this file from www.ONsip.org [http://www.ONsip.org/]. A detailed analysis of this ser.cfg follows the example.

```
debug=3❶
fork=no❷
log_stderror=yes❸

listen=192.0.2.13❹        # INSERT YOUR IP ADDRESS HERE
port=5060❺
children=4❻

dns=no❼
rev_dns=no
fifo="/tmp/ser_fifo"❽

❾
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"

modparam("usrloc", "db_mode", 0)❿
modparam("rr", "enable_full_lr", 1)[11]

[12]route {

  # -----------------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) { [13]
    sl_send_reply("483", "Too Many Hops");[14]
    break;[15]
  };
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    if (msg:len > max_len) { 16
      sl_send_reply("513", "Message Overflow");
      break;
    };

    # -----------------------------------------------------------------------
    # Record Route Section
    # -----------------------------------------------------------------------
    if (method!="REGISTER") { 17
      record_route();18
    };

    # -----------------------------------------------------------------------
    # Loose Route Section
    # -----------------------------------------------------------------------
    if (loose_route()) { 19
      route(1);20
      break;21
    };

    # -----------------------------------------------------------------------
    # Call Type Processing Section
    # -----------------------------------------------------------------------
    if (uri!=myself) { 22
      route(1);23
      break;24
    };

    if (method=="ACK") {25
      route(1);26
      break;27
    } if (method=="REGISTER") { 28
      route(2);29
      break;30
    };

    lookup("aliases");31
    32if (uri!=myself) {
      route(1);
      break;
    };

    if (!lookup("location")) { 33
      sl_send_reply("404", "User Not Found");34
      break;35
    };

    route(1);36
  }

  37route[1] {
    # -----------------------------------------------------------------------
      # Default Message Handler
    # -----------------------------------------------------------------------
    if (!t_relay()) {38
      sl_reply_error();39
    };
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  }

40route[2] {
  # -----------------------------------------------------------------------
  # REGISTER Message Handler
  # -----------------------------------------------------------------------
  if (!save("location")) { 41
    sl_reply_error();42
  };
}
```

Hello World ser.cfg Analysis

Now that you have seen our 'Hello World' SIP proxy configuration, we'll explain each line in the file so that you can begin to understand the basics of SER.

This configuration is a complete, albeit, minimal SIP router. SIP clients can register with the SIP proxy and can call other registered users. This configuration however is far from complete in terms of functionality. For one thing, if the SIP proxy is restarted then all client registration information is lost because there was no mechanism for persisting registration information to disk. Call features are also absent as well as NAT traversal and voice mail. But don't worry as we will cover all these topics later on. For now, lets just concentrate on a bare bones SIP router.

❶ SER has debug information that can be enabled or suppressed using the debug directive. A nominal value of 3 is usually specified to obtain enough debug information when errors occur. The debug directive specifies how much information to write to syslog. The higher the number, the more verbose SER becomes. The most verbose debug level is 9. When the debug level is set higher than 3 SER becomes very verbose and the start up time can take a bit longer.

❷ The fork directive tells the SER daemon to run in the foreground or the background. When you are ready to operate SER as a system service, you must set this directive to yes. For now we will just operate SER as a foreground process.

NOTE: See the appendix for a SER init.d start script.

❸ Since we are running SER as a foreground process we must set the log_stderror directive equal to yes in order to see the output

❹ The listen directive instructs SER to listen for SIP traffic on a specific IP address. Your server must physically listen on the IP address you enter here. If you omit this directive then SER will listen on all interfaces.

NOTE: When you start SER, it will report all the interfaces that it is listening on.

❺ In addition to specifying the IP address to listen on you can also specify a port. The default port used on most SIP routers is 5060. If you omit this directive then SER assumes port 5060.

❻ The children directive tells SER how many processes to spawn upon server start up. A good number here is 4, however, in a production environment you may need to increase this number.

❼ These lines are really to prevent SER from attempting to lookup its IP address in DNS. By adding these two lines to ser.cfg we suppress any warnings if your IP is not in your DNS server.

❽ The fifo directive specifies the location of the SER FIFO. The FIFO is sort of like the Linux proc file system and can be used to examine the current SIP proxy activity. The FIFO can also be used if you want to inject a new SIP message directly in to the SIP proxy with an external application. A good example of this is the serctl utility that is usually located in /usr/local/sbin/. With this utility you can manage users, ping SIP URIs, and even email a SIP user. Serctl does all of this internally using the FIFO.

The FIFO can be located just about anywhere on disk, however, the user account that SER runs under must be able to create the FIFO in the specified directory.

❾ Here we have external modules that are necessary for our 'hello world' SIP proxy to function. SER modules may be located anywhere on the disk system, but /usr/local/lib/ser/modules is the default location. Loading a module is as simple as specifying in a loadmodule directive as shown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

If you're wondering how you know which module or modules you need, well there is no clear cut way to know. In general the modules shown here will be needed by every SIP proxy configuration. When you need to add additional functionality, such as MySQL connectivity, then you will load the appropriate module, in this case the mysql.so module.

All SER modules are distributed with the source code and located at <ser-source>/modules. Every SER module has a README file that explains all the items that it exposes for use in a ser.cfg file. It is highly recommended that you spend some time familiarizing your self with the available SER modules. This document makes a valid attempt at explaining the most important SER modules and how to use them, but there is no substitute for reading the actual module README files.

Some modules have parameters that need to be set in ser.cfg in order for the module to function properly. Other modules, however, operate normally in most cases without any parameter adjustments. The next section will show our 'hello world' requirements.

10 The usrloc module is responsible for keeping track of SIP client registration locations. In other words, when a SIP client registers with the SIP proxy, SER will store the contact information, also known as the Address Of Record (AOR), in a table. This table is then queried when another SIP client makes a call. The location of this table can vary depending on the value of the usrloc db_mode parameter.

Our SIP proxy sets db_mode to zero to indicate that we do not want to persist registration data to a database. Instead we will only store this information in memory.

We will show how to persist AORs in a later example.

11 The rr parameter called enable_full_lr is really a work-around for older SIP clients that don't properly handle SIP record-route headers. SIP RFC3261 says that specifying loose routing by including a ;lr in the record-route header is the correct way to tell other SIP proxies that our SIP proxy is a loose router. However, some older and/or broken SIP clients incorrectly dropped the ;lr tag because it didn't have value associated with it (ie, lr=true).

Therefore, by setting enable_full_lr to one, SER will write all ;lr tags as ;lr=on to avoid this problem.

12 This is the beginning of the SIP processing logic. This line defines the main route block. A route block must have a beginning and ending curly bracket.

The main route block is where all received SIP messages are sent. From the main route block you can call other route blocks, test the message for certain conditions, reject the message, relay the message, and basically do anything you need to fit your business needs.

Here is a main overview of what happens:

1. A message enters the main route and we do some checks
2. We determine if the message is for us, if not we just send it to where it belongs (route[1])
3. If its for us, we explicitly handle REGISTER messages (a message from a phone asking to register itself). Route[2] handles REGISTERs by saving where we can reach the phone, while route[1] is a default handler for all other messages.

NOTE: It may appear odd to present such simple functionality this way, but we will use this structure as the basis for a much more complex ser.cfg which is presented step-by-step in following sections.

The details are below.

13 mf_process_maxfwd_header is a safety check that you should always include as the first line of your main route block. This function is exposed in the mf.so module and is used to keep track of how many times a SIP message has passed through SER. Erroneous ser.cfg files can send a SIP message to the wrong location and cause looping conditions. Also other SIP proxies that SER interacts with can do the same thing.

The basic rule here is that if this function ever returns 'true' then you need to stop processing the problematic message to avoid endless looping.

14 If a looping situation is detected then SER needs a way to tell the SIP client that an error has occurred. The sl_send_reply() function performs this job. sl_send_reply() is exposed in the sl.so module and all it does is

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

sends a stateless message to the SIP client. This means that SER will send the message and forget about it. It will not try to resend if it does not reach the recipient or expect a reply.

You can specify an appropriate error message as shown. The error message can only selected by the defined SIP error codes and messages. Many IP phones will show the text message to the user.

15 The break statement tells SER to stop processing the SIP message and exit from the route block that it is currently executing. Since we are calling break in the main route block, SER will completely stop processing the current message.

16 msg:len is a core SER function that returns the length in bytes of the current SIP message. This, like mf_process_maxfwd_header() should be called at the beginning of the main route block in all ser.cfg files.

This statement simply tests the length of the SIP message against the maximum length allowed. If the SIP message is too large, then we stop processing because a potential buffer overflow has been detected.

17 Here we look to see if the SIP message that was received is a REGISTER message. If it is not a register message then we must record-route the message to ensure that upstream and/or downstream SIP proxies that we may interact with keep our SIP proxy informed of all SIP state changes. By doing so, we can be certain that SER has a chance to process all SIP messages for the conversation.

The keyword 'method' is provided by the SER core and allows you to find out what type of SIP message you are dealing with.

18 The record_route() function simply adds a Record-Route header field to the current SIP message. The inserted field will be inserted before any other Record-Route headers that may already be present in the SIP message. Other SIP servers or clients will use this header to know where to send an answer or a new message in a SIP dialog.

19 loose_route() tests to see if the current SIP message should be loose routed or not. If the message should be loose routed then SER should simply relay the message to the next destination as specified in the top-most Record-Route header field.

In all ser.cfg files you should call loose_route() after the record_route() function.

This is the basic definition of loose routing. Refer to RFC3261 for a complete explanation of loose routing.

Note for interested readers:

RFC3261 states that a SIP proxy is said to be "loose routing" if it

follows the procedures defined in this specification for processing of

the Route header field. These procedures separate the destination of the

request (present in the Request-URI) from the set of proxies that need to

be visited along the way (present in the Route header field). A proxy

compliant to these mechanisms is also known as a "loose router".

#

SER is a loose router, therefore you do not need to do anything aside

from calling loose_route() in order to implement this functionality.

#

A note on SIP message dialogs (contributed by Jan Janak):

#When it comes to SIP messages, you can classify them as those that

#create a dialog and those that are within a dialog. A message within

#a dialog (ACK, BYE, NOTIFY) typically requires no processing on the

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

#server and thus should be relayed at the beginning of [your ser.cfg]

#right after loose_route function.

#

#Messages creating dialogs can be further classified as those that

#belong to the server (they have the domain or IP of the server in

#the Request-URI) and those that do not. Messages that do not have a

#domain or IP of the proxy in the Request-URI should be immediately

#relayed or blocked.

20 If the test for loose routing returns 'true' then we must relay the message without other processing. To do so we pass execution control to route block number one (ie, route[1]). This route block is documented later but it suffices now to say that it is the default message handler.

21 Since loose routing was required we cannot process the message any further so we must use a break command to exit the main route block.

22 Now we have come to a part of the SIP configuration file were we are processing out-of-dialogue messages. In other words, we are processing either a message that will begin a new dialogue, or we are processing a SIP message that is not destined for our sip proxy (but we are relaying or proxying the message).

The keyword 'uri' is defined in the SER core and is a synonym for the request URI or R-URI for short. The 'myself' keyword is also defined in the SER core and is a synonym for the SER proxy itself.

So this line tests to see if the R-URI is the same as the SIP proxy itself. Another way to look at it is by saying that if this statement is TRUE then the SIP message is not being sent to our SIP proxy but rather to some other destination, such as a third party SIP phone.

23 Relay the message to the destination without further interrogation.

24 Stop processing the current message because we have sent it to its destination on the previous line.

25 ACK messages are explicitly handled, so we need to find out if we are processing an ACK request.

26 Relay the message to the destination without further interrogation.

27 Stop processing because the ACK message has already been fully processed.

28 REGISTER messages are explicitly handled, so we need to find out if we are processing a registration request.

29 All registration requests are passed to route[2] for processing. We do this to somewhat segregate the workload of the main route block. By doing so we keep our ser.cfg from becoming unmanageable.

30 Stop processing because the REGISTER message has already been fully processed.

31 Lookup(aliases) attempts to retrieve any aliases for the Requested URI. An alias is just another way to refer to a SIP destination. For example, if you have a SIP user 4075559938 and a toll free number 8885551192 that rings to 4075559938, then you can create the toll free number as an alias for 4075559938. When someone dials the toll free number the SIP client configured as 4075559938 will ring.

The lookup(aliases) function is not strictly needed for this example, however, serctl, which is a command line utility included with SER requires this line for it to operate properly.

32 If an alias was found and the Requested URI is no longer a locally served destination, then we relay the SIP message using our default message handler route[1]. After the message is relayed we stop processing by means of the break command.

33 lookup(location) attempts to retrieve the AOR for the Requested URI. In other words it tries to find out where the person you are calling is physically located. It does this by searching the location table that the save() function updates. If an AOR is located, we can complete the call, otherwise we must return an error to the caller to indicate such a condition.

34 If the party being called cannot be located then SER will reply with a 404 User Not Found error. This is done in a stateless fashion.

35 Since the user cannot be found we stop processing.

36 If the current SIP message is not a REGISTER request, then we just relay it to its destination without further processing. We do this by passing control to route[1], the default message handler.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

37 Route[1] is the default message handler that is used throughout the ser.cfg file to relay messages. Notice how the route block definition uses square brackets, but to call a route block you use parenthesis.

38 t_relay() is a function exposed by the tm.so module and is perhaps one of the most important functions in any ser.cfg file. t_relay() is responsible for sending a message to it's destination and keep track of any re-sends or replies. If the message cannot be sent successfully, then t_relay() will return an error condition.

See section 1 for more in-depth background information on transactions.

39 If t_relay() cannot send the SIP message then sl_reply_error() will send the error back to the SIP client to inform it that a server error has occurred.

40 Route[2] is the registration message handler. This is where we store user contact data.

41 All REGISTER messages are happily honoured because we have not yet implemented authorization checks. The save() function is responsible for storing SIP client registration information in the location table. However, recall that the usrloc db_mode has been set to zero on line 5.1. This means that the save() function will only save the registration information to memory rather than to disk.

42 If SER is unable save the contact information then we just return the error to the client and return to the main route block.

Using The Hello World ser.cfg Configuration

Now that we have described our first SIP proxy configuration it is time to start up the server and configure two SIP phones so that we can make telephone calls between the two registered clients.

To start SER as a non-daemon process that runs in the foreground you need to open up your favourite terminal, such as a BASH shell.

Then issue the following command:

/usr/local/sbin/ser D E

The D parameter tells SER to not run as a daemon (ie, dont run in the background) while the E parameter tells SER to send any errors to stderr.

SER should start up and say that it is listening for TCP and UDP on the IP address of the computer that you are using. The IP must not be 127.0.0.1 because this IP is your loopback interface and SIP phones will not be able to contact your SIP proxy.

Please note that unless SER is compiled with mode=debug, this may not work on all architectures. SER will just print some basic stuff and then be quiet.

Now that SER is running you should configure two (2) or more SIP phones. A good example of a SIP phone is a Grandstream BT100 because they are inexpensive and work well for basic SIP testing.

When configuring your SIP phones you should use the following settings:

SIP Proxy	IP Address of your SER server
SIP Port	5060
Outbound Proxy	Leave blank
User name	Enter a unique numeric value. A good example would be 1000 for the first phone and 1001 for the second. Each phone should have a unique user name, although you can have more than one SIP phone with the same credentials.
Authentication ID	If your SIP phone has such a field, you can use the same value as the User Name field or just leave it blank.
Password	You can leave this field blank for now. If you key a value in it will be ignored because out SIP proxy does not perform any authentication at this point.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

You should now be able to boot up your SIP phones and call each other. The numbers you would dial are whatever values you keyed in to the User Name fields. So in our example SIP phone 1000 can dial 1001 and SIP phone 1001 can dial 1000.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 7. SER - Adding Authentication and MySQL

What this ser.cfg does:

1. Adds authentication of IP phones by using credentials stored in MySQL
2. Contact information is stored persistently in MySQL

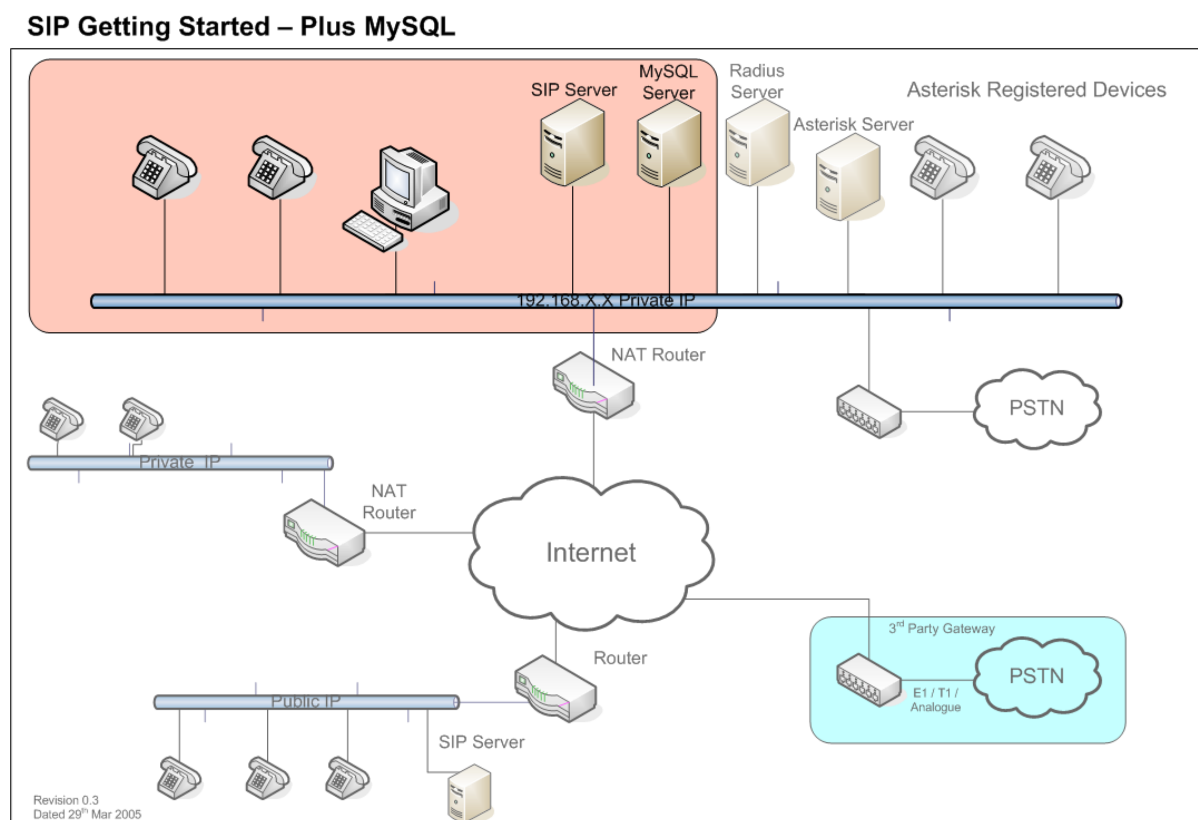
Now that you have tested a basic SIP environment, we need to add more functionality. This section we talk about authentication.

In normal circumstances, we must restrict the use of the SIP server to those telephones (i.e., users) that we want. Authentication is about ensuring only those telephones that we have given a password to are allowed to use our sip services.

To support authentication, we need a way to store information that does not get lost when we stop the sip server. We need a database and the most popular is MySQL as it comes with all Linux configurations. Support is provided for other databases, such as PostgreSQL, but in this Quick Start guide we focus on MySQL.

To add support for MySQL you need to go back to the source code and modify a few parameters. In Chapter 11 - Supporting MySQL describes how to do this and re-install the binaries. Once you have updated your SER environment you need to modify your ser.cfg file as described below.

Figure 7.1. Reference Design Plus MySQL



XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

MySQL ser.cfg Listing

Listed below is the SIP proxy configuration to which builds upon subject matter covered in the Hello World section.

```
debug=3
fork=no
log_stderror=yes

listen=192.0.2.13   # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"
fifo_db_url="mysql://ser:heslo@localhost/ser"❶

loadmodule "/usr/local/lib/ser/modules/mysql.so"❷
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"❸
loadmodule "/usr/local/lib/ser/modules/auth_db.so"❹
loadmodule "/usr/local/lib/ser/modules/uri_db.so"❺

modparam("auth_db|uri_db|usrloc", "db_url", "mysql://ser:heslo@localhost/ser")❻
modparam("auth_db", "calculate_ha1", 1)❼
modparam("auth_db", "password_column", "password")❽
modparam("usrloc", "db_mode", 2)❾
modparam("rr", "enable_full_lr", 1)

route {

  # -----------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    break;
  };

  if (msg:len > max_len) {
    sl_send_reply("513", "Message Overflow");
    break;
  };

  # -----------------------------------------------------------------
  # Record Route Section
  # -----------------------------------------------------------------
  if (method!="REGISTER") {
    record_route();
  };
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  # -----------------------------------------------------------------
  # Loose Route Section
  # -----------------------------------------------------------------
  if (loose_route()) {
    route(1);
    break;
  };

  # -----------------------------------------------------------------
  # Call Type Processing Section
  # -----------------------------------------------------------------
  if (uri!=myself) {
    route(1);
    break;
  };

  if (method=="ACK") {
    route(1);
    break;
  } if (method=="INVITE") { ⓾
    route(3);11
    break;
  } else  if (method=="REGISTER") {
    route(2);
    break;
  };

  lookup("aliases");
  if (uri!=myself) {
    route(1);
    break;
  };

  if (!lookup("location")) {
    sl_send_reply("404", "User Not Found");
    break;
  };

  route(1);
}

route[1] {

  # -----------------------------------------------------------------
  # Default Message Handler
  # -----------------------------------------------------------------
  if (!t_relay()) {
    sl_reply_error();
  };
}

route[2] {

  # -----------------------------------------------------------------
  # REGISTER Message Handler
  # ----------------------------------------------------------------
  sl_send_reply("100", "Trying");12
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    if (!www_authorize("","subscriber")) { 13
      www_challenge("","0");14
      break;15
    };

    if (!check_to()) { 16
      sl_send_reply("401", "Unauthorized");17
      break;
    };

    consume_credentials();18

    if (!save("location")) { 19
      sl_reply_error();
    };
  }

20route[3] {
    # -----------------------------------------------------------------
    # INVITE Message Handler
    # -----------------------------------------------------------------
    if (!proxy_authorize("","subscriber")) { 21
      proxy_challenge("","0");22
      break;
    } else if (!check_from()) { 23
      sl_send_reply("403", "Use From=ID");24
      break;
    };

    consume_credentials();25

    26lookup("aliases");
    if (uri!=myself) {
      route(1);
      break;
    };

    27if (!lookup("location")) {
      sl_send_reply("404", "User Not Found");
      break;
    };

    route(1);28
  }
```

Authenticating ser.cfg Analysis

Our new SIP proxy configuration is quickly maturing to meet real world requirements. We now store all user registration data to MySQL which enables us to restart the SIP proxy without affecting user registrations. This means that you can quickly restart SER without the clients knowing about it.

Another very important feature that we have introduced is authentication. Authentication happens during two different times in order to secure our SIP router. The first place we need to look at is the area that handles REGISTER messages because we do not want anonymous users to have the ability to register with our SIP proxy.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The second area we must secure is the handler that processes INVITE messages because we do not want unauthenticated users to make telephone calls. If we allowed this then we would have what is called an open relay and if your SIP proxy is connected to a PSTN gateway you could be responsible for excessive toll charges.

An important thing to note is that comments from the Hello World configuration have been removed for clarity so that you can quickly find new areas in this ser.cfg file, which have been commented.

❶ The fifo_db_url directive is also included to suppress start up warnings that would otherwise appear when adding MySQL support. We do not directly use the fifo_db_url from ser.cfg, however, other ancillary tools, such as serctl use it to add users to the database.

❷ MySQL support is added easily by including the mysql.so module in the loadmodule section. A very important thing to notice is that mysql.so is loaded before all other modules. The reason is that mysql.so does not have any module dependencies, however, other modules, such as uri_db, do depend on the mysql.so module.

❸ The auth.so module is not directly used by ser.cfg, however it is necessary to enable authentication functionality. The authentication functionality in this ser.cfg file is provided by the auth.so and auth_db.so modules.

❹ Auth_db.so is the module that we invoke directly. This module interoperates with auth.so to perform its function.

❺ Uri_db.so is the module that exposes some authentication functions that we will use in this ser.cfg example, namely check_to().

❻ The auth_db module exposes a db_url parameter which is required in order to tell SER where to find a MySQL database for user authentication. If you notice we have included auth_db, uri_db, AND usrloc in a single modparam by using the pipe symbol. We do this as a short cut, however it is perfectly legal in SER syntax to have included these in separate modparam statements.

❼ The auth_db parameter calculate_ha1 tells SER whether or not to use encrypted passwords in the MySQL subscriber table. In a production system you will want to set this parameter to zero, but in our development system we use unencrypted passwords so the value is set to one.

❽ The auth_db module defaults the password column name to ha1, however the MySQL schema that SER uses calls the password column password. Therefore we must inform SER that the column name has changed.

❾ The usrloc parameter db_mode must be changed from zero, used in the Hello World example to 2 in this example to configure SER to use MySQL for storing contact information and authentication data.

❿ We now explicitly define an INVITE handle route[3]. This handler is responsible for setting up a call.

11 Pass control to route[3] for all INVITE messages that have not been loose routed. Invite messages hitting this statement are original INIVITEs rather than re-INVITEs. After an INVITE message is processed we exit by calling break.

12 When we receive a REGISTER message, we immediately send a 100 Trying message back to the SIP client to stop it from retransmitting REGISTER messages. Since SER is UDP based there is no guaranteed delivery of SIP messages, so if the sender does not get a reply back quickly then it will retransmit the message.

By replying with a 100 Trying message we tell the SIP client that we're processing its request.

13 The www_authorize() function is used to check the user's credentials against the values stored in the MySQL subscriber table. If the supplied credentials are correct then this function will return TRUE, otherwise FALSE is returned.

If credentials were not supplied with the REGISTER message, then this function will return FALSE.

The first parameter specifies the realm in which to authenticate the user. Since we are not using realms, we can just use an empty string here. You can think of the realm in exactly the same way you think of a web server realm (domain).

The second value tells SER which MySQL table to use for finding user account credentials. In this case we specified the subscriber table.

14 Here we actually send back a 401 Unauthorized message to the SIP client which tell it to retransmit the request with digest credentials included.

www_challenge() takes two arguments. The first is a realm, which will appear in the WWW-Authorize header that SER sends back to the SIP client. If you put a value in here then that realm will appear to the SIP client when it is challenged for credentials.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The second value affects the inclusion of the qop parameter in the challenge request. It is recommended to keep this value set to 1. See RFC2617 for a complete description of digest authentication. Please note though that some IP phones do not support qop authentication. You may try to use 0 if you experience Wrong password problems.

15 Since we sent back a 401 error the SIP client on the previous line we no longer need to service this REGISTER message. Therefore we use the break command to return to the main route block.

16 When operating a SIP proxy you must be certain that valid user accounts that have been successfully registered cannot be used by unauthenticated users. SER includes the check_to() function for this very reason.

We call check_to() prior to honouring the REGISTER message. This causes SER to validate the supplied To: header against the previously validated digest credentials. If they do not match then we must reject the REGISTER message and return an error.

17 If check_to() returned FALSE then we send a 401 Unauthorized message back to the SIP client. Then we call the break command to return to the main route block.

18 We do not want to risk sending digest credentials to upstream or downstream servers, so we remove any WWW-Authorize or Proxy-Authorize headers before relaying further messages.

19 If execution of ser.cfg makes it to this line then the SIP user has successfully been validated against the MySQL subscriber table, so we use the save(location) function to add the user's contact record to the MySQL location table. By saving this contact record to MySQL we can safely restart SER without affecting this registered SIP client.

20 Route[3] is introduced here to handle INVITE messages.

21 We use proxy_authorize() to make sure we are not an open relay. Proxy_authorize() will require INVITE messages to have digest credentials included in the request. If they are included the function will try to validate them against the subscriber table to make sure the caller is valid.

Like www_authorize(), proxy_authorize() also takes two arguments. The first is a realm and the second is the MySQL table in which to look for credentials. In our example this is the MySQL subscriber table.

22 If the user did not properly authenticate then SER must reply with a 401 Unauthorized message. The proxy_challenge() function takes two arguments that are the same as the www_challenge() function, namely a realm and a qop specifier.

Now that we've sent a 401 challenge to the caller we execute a break command to return to the main route block.

23 Here we call check_from() to make sure the INVITE message is not using hi-jacked credentials. This function checks the user name against the digest credentials to verify their authenticity.

24 If check_from() returned false then we reply to the client with a 401 unauthorized message and return to the main route block.

25 We do not want to risk sending digest credentials to upstream or downstream servers, so we remove any WWW-Authorize or Proxy-Authorize headers before relaying further messages.

26 Look up any associated aliases with the dialed number. If there is an aliases and it is not a locally served domain then just relay the message.

27 Now that all request URI transformations have been done, we can attempt to look for the correct contact record in the MySQL location table. If an AOR (aka, address of record) cannot be found then we reply with a 404 error.

28 Finally, if execution has made it here, then the caller has been properly authenticated and we can safely relay the INVITE message.

Using the Authenticating ser.cfg Proxy

Before you can use this new SIP route configuration you must configure SIP user accounts by using the serctl shell script. We'll continue with user 1000 and 1001. In the Hello World example we happily registered any user that sent a REGISTER message.

Now we require authentication. So open up a terminal window and execute the following two commands:

1. serctl add 1000 password1 user1@nowhere.com [mailto:user1@nowhere.com]
2. serctl add 1001 password2 user2@nowhere.com [mailto:user2@nowhere.com]

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

You will be prompted for a password. This will be your MySQL SER user's password. Refer to the section on setting up MySQL for further instructions on configuring the MySQL user account.

Once the accounts are created you need to update your SIP phones by changing the passwords to whatever you used in the serctl commands. Then reboot your SIP phones and you should be able to complete telephone calls.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 8. Handling of NAT

There are two implementations for handling of NAT, use of mediaproxy and the use of rtpproxy. The following sections describe each of these methods.

There is considerable debate on the serusers mailing forum about which is better! Both are described in this document, but due to the simple fact that the authors of this document have deployed mediaproxy in their own installations means that the rest of this Getting Started document builds upon the mediaproxy solution. If there are users who would like to contribute material supporting the rtpproxy solution then this will be considered for future issues.

Figure 8.1. Reference Design Plus NAT

SIP Getting Started – Plus NAT

Handling of NAT using Mediaproxy

What this ser.cfg will do:

1. Introduce specialized processing which is necessary when dealing with SIP clients that are behind a NAT device such as a DSL router or corporate firewall

Mediaproxy is one of two NAT traversal solutions for SER, the other being rtpproxy. Mediaproxy is known as a far-end NAT traversal solution which means that it handles all aspects of NAT at the SIP proxy location rather than at the SIP client location.

There are many advantages to handling NAT issues at the SIP proxy server, the primary benefit being that SIP client configuration is much simpler. Some of the important features of mediaproxy are:

1. DNS SRV record capabilities for load distribution
2. Mediaproxy can be installed on remote servers which are not running SER. By doing this your SIP proxy load is lightened

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

3. Web monitoring is included with the mediaproxy distribution

Mediaproxy is a separate software product that is not distributed with SER. The SER distribution only includes the glue which gives SER the ability to communicate with a running instance of mediaproxy. This glue is known as the mediaproxy module. It is really a dispatcher module that can control one or more actual mediaproxy servers.

NOTE: In order for mediaproxy to function properly it must be configured to listen on a public IP address. Also, in most real world configurations, mediaproxy will not be installed on the SER server, but on a remote machine. Refer to the appendix for information on installing mediaproxy. Also, the mediaproxy version used for testing these configuration files can be downloaded from http://ONsip.org/

```
debug=3
fork=yes❶
log_stderror=no❷

listen=192.0.2.13   # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"
fifo_db_url="mysql://ser:heslo@localhost/ser"

loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"❸
loadmodule "/usr/local/lib/ser/modules/domain.so"❹
loadmodule "/usr/local/lib/ser/modules/mediaproxy.so"❺
loadmodule "/usr/local/lib/ser/modules/nathelper.so"❻
loadmodule "/usr/local/lib/ser/modules/textops.so"❼

modparam("auth_db|domain|uri_db|usrloc","db_url","mysql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")

modparam("nathelper", "rtpproxy_disable", 1)❽
modparam("nathelper", "natping_interval", 0)❾

modparam("mediaproxy","natping_interval", 30)❿
modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")⓫
modparam("mediaproxy","sip_asymmetrics","/usr/local/etc/ser/sip-clients")⓬
modparam("mediaproxy","rtp_asymmetrics","/usr/local/etc/ser/rtp-clients")⓭

modparam("usrloc", "db_mode", 2)

modparam("registrar", "nat_flag", 6)⓮

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


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
route {

  # -----------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    break;
  };

  if (msg:len > max_len) {
    sl_send_reply("513", "Message Overflow");
    break;
  };

  # -----------------------------------------------------------------
  # Record Route Section
  # -----------------------------------------------------------------
  if (method=="INVITE" && client_nat_test("3")) { 15
    # INSERT YOUR IP ADDRESS HERE
    record_route_preset("192.0.2.13:5060;nat=yes");16
  } else if (method!="REGISTER") {
    record_route();17
  };

  # -----------------------------------------------------------------
  # Call Tear Down Section
  # -----------------------------------------------------------------
  if (method=="BYE" || method=="CANCEL") { 18
    end_media_session();19
  };

  # -----------------------------------------------------------------
  # Loose Route Section
  # -----------------------------------------------------------------
  if (loose_route()) {

    if ((method=="INVITE" || method=="REFER") && !has_totag()) {20
      sl_send_reply("403", "Forbidden");
      break;
    };

    if (method=="INVITE") {

      21if (!proxy_authorize("","subscriber")) {
        proxy_challenge("","0");
        break;
      } else if (!check_from()) {
        sl_send_reply("403", "Use From=ID");
        break;
      };

      consume_credentials();

      if (client_nat_test("3") || search("^Route:.*;nat=yes")) { 22
        setflag(6);23
        use_media_proxy();24
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
        };
      };

      route(1);
      break;
    };

    # -----------------------------------------------------------------
    # Call Type Processing Section
    # -----------------------------------------------------------------
    if (uri!=myself) {
      route(4);25
      route(1);
      break;
    };

    if (method=="ACK") {
      route(1);
      break;
    26} else if (method=="CANCEL") {
      route(1);
      break;
    } else if (method=="INVITE") {
      route(3);
      break;
    } else  if (method=="REGISTER") {
      route(2);
      break;
    };

    lookup("aliases");
    if (uri!=myself) {
      route(4);27
      route(1);
      break;
    };

    if (!lookup("location")) {
      sl_send_reply("404", "User Not Found");
      break;
    };

    route(1);
  }

  route[1] {

    # -----------------------------------------------------------------
    # Default Message Handler
    # -----------------------------------------------------------------

    t_on_reply("1");28

    if (!t_relay()) {

      29if (method="INVITE" || method=="ACK") {
        end_media_session();
      };
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    sl_reply_error();
  };
}

route[2] {

  # -----------------------------------------------------------------
  # REGISTER Message Handler
  # ----------------------------------------------------------------

  sl_send_reply("100", "Trying");

  if (!search("^Contact:[ ]*\*") && client_nat_test("7")) { 30
    setflag(6);31
    fix_nated_register();32
    force_rport();33
  };

  if (!www_authorize("","subscriber")) {
    www_challenge("","0");
    break;
  };

  if (!check_to()) {
    sl_send_reply("401", "Unauthorized");
    break;
  };

  consume_credentials();

  if (!save("location")) {
    sl_reply_error();
  };
}

route[3] {

  # -----------------------------------------------------------------
  # INVITE Message Handler
  # -----------------------------------------------------------------

  if (client_nat_test("3")) { 34
    setflag(7);35
    force_rport();36
    fix_nated_contact();37
  };

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

  consume_credentials();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  lookup("aliases");
  if (uri!=myself) {
    route(4);38
    route(1);
    break;
  };

  if (!lookup("location")) { 39
    sl_send_reply("404", "User Not Found");
    break;
  };

  route(4);40
  route(1);
}

41route[4] {

  # -----------------------------------------------------------------
  # NAT Traversal Section
  # -----------------------------------------------------------------

  if (isflagset(6) || isflagset(7)) {
    if (!isflagset(8)) {42
      setflag(8);
      use_media_proxy();
    };
  };
}

43onreply_route[1] {

  if ((isflagset(6) || isflagset(7)) && (status=~"(180)|(183)|2[0-9][0-9]")) { 44

    45if (!search("^Content-Length:[ ]*0")) {
      use_media_proxy();
    };
  };

  46if (client_nat_test("1")) {
    fix_nated_contact();
  };
}
```

Mediaproxy Transparent NAT Traversal ser.cfg Analysis

Our NAT traversal implementation is referred to as transparent NAT traversal because there are no differences between SIP clients with public IP addresses and SIP clients sitting behind a NAT device.

This SER configuration handles all NAT related issues invisibly so that configuring SIP phones is a breeze. We do not use STUN because STUN generally adds another layer of complexity that can be avoided.

To handle NATed devices we have introduced the use of mediaproxy. Mediaproxy is an RTP proxy application which is not part of SER. Mediaproxy installation and start script are shown in the appendix.

The issue of NAT is complicated because there are several aspects of NATed SIP clients that all must be addressed properly in order to effectively perform transparent RTP proxying. Please refer to NAT, STUN, and RTP proxy, section 3.5, for an in-depth review of these aspects.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

A brief re-cap of the main NAT issues that must be addressed are:

1. NATed clients must maintain an open SIP port, usually 5060 at all times. If this port closes on the clients NAT device then incoming calls will not be successful because the SIP proxy will be unable to notify the NATed SIP client that an INVITE message is being sent to it. In this case it is possible for the NATed SIP client to still make outbound calls.

2. RTP ports can, and usually do use a random port. This means that users cannot just open a port on their NAT device for RTP.

3. re-INVITE messages must be handled in a special manner in order to prevent RTP media streams from being lost mid-call. If this were to happen you usually end up with one-way audio or no audio after a re-INVITE message is processed. NOTE: The reason for this is that SIP RFC3261 specifies that a SIP UA may change CODECs, RTP ports, etc during mid-call. To compound this issue, it is difficult to determine if either SIP party in a call is NATed when processing re-INVITE messages.

So how do we handle these NAT issues? The solutions to these problems are:

1. The SER configuration presented in this section takes advantage of the fact that we can use the SER proxy to keep NAT bindings open with the use of the natping_interval setting in mediaproxy. By doing so we can be certain that port 5060 for NATed clients will be accessible to our SIP proxy and therefore we will be able to send INVITE messages to those clients at any time.

2. Mediaproxy gives SER the ability to be oblivious to the specific RTP port that a SIP client uses. The way it does this is that the two SIP clients think they are talking directly to each other, when in reality they are talking to the media proxy, which then forwards RTP data to the other SIP client. This is all handled automatically when the original SIP INVITE message is being processed and the call is set up.NOTE: In this ser.cfg example, we only proxy RTP media when one or more SIP clients are behind a NAT device. In cases where both SIP clients are on the public Internet, then we do not proxy RTP streams since both SIP clients can directly contact each other. This is a key to building a scaleable VoIP platform.

3. We take advantage of the fact that we can embed our own NAT indicator in the Record-Route header of the original INVITE message. By doing so we can look for this special NAT tag when processing re-INVITE messages. If we find this tag then we know we must proxy the RTP media.NOTE: This technique of embedding a NAT flag in the Record-Route header has been suggested by Jan Janak of IPtel. He is one of the projects core developers and is an authoritative person on the subject.NOTE: This technique of embedding a NAT flag in the INVITE message is not required when using rtpproxy because the rtpproxy server has the ability to handle this situation without additional help.

❶ Up until now we have run SER as a foreground process. From this point forward we will run SER as a daemon. The fork directive tells the SER daemon to run in the background. This is a requirement for using the init.d start script shown in the appendix.

❷ Since we are running SER as a background process we must set the log_stderror directive equal to no in order to not keep SER in the foreground.

❸ The uri module is introduced here to access the has_totag() function. This function is necessary for processing re-INVITEs and is describe below.

❹ The domain module is needed because the mediaproxy module has dependencies to it for the is_uri_host_local() and is_from_local() functions.

❺ Here we introduce the use of mediaproxy. This line has confused many SER novices because it implies that mediaproxy is included with SER, however, mediaproxy is an external application. This line only invokes the mediaproxy module, which is the glue between SER and the actual mediaproxy application.

❻ We use some of the nathelper library functions to determine whether or not a SIP UA is NATed. An important item to understand is that just because nathelper is included here does not indicate that we are using rtpproxy (another RTP proxy application comparable to mediaproxy). Also note here that nathelper is generally referred to in the same context as rtpproxy because most people that use rtpproxy also use nathelper.

❼ The textops module provide utility functions for manipulating text, searching for substrings, and checking for the existence of specific header fields.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

❽ Since we are using mediaproxy rather than rtpproxy, we need to tell the nathelper module to not attempt to bind to a running instance of rtpproxy. If we were to omit this line then syslog would contain many errors stating that rtpproxy could not be found.

❾ Our NAT traversal is using mediaproxy, and therefore we have decided to use the mediaproxy keep-alive mechanism, which will ping NATed SIP clients on a regular basis. Because of this we need to disable the rtpproxy ping mechanism since we do not need it.

❿ The mediaproxy natping_interval is a very crucial setting, which controls how often our SER proxy will ping registered SIP clients. Most NAT devices will only hold port bindings open for a minute or two, so we specify 30 seconds here.

This causes SER to send a 4-byte UDP packet to each SIP client every 30 seconds. This is all that is required to keep NATed clients alive.

NOTE: Some NAT devices have been reported to not allow incoming keep-alives. Thus, many user clients have their own implementations of keep-alive. If you experience one-way audio problems after a while, you may have run into this problem. The only solution is to turn on user client keep-alives. Also, keep in mind that it is perfectly acceptable to have the SER proxy send keep-alive traffic as well as the SIP traffic to the client.

11 SER and the mediaproxy dispatcher communicate via a standard Unix socket, which is specified here.

12 Mediaproxy may need to know when a SIP UA is asymmetric with respect to its SIP messaging port. Most SIP UAs are symmetric, meaning that they listen for incoming SIP messages on the same port as they use for sending their own SIP messages. However, if you find asymmetric clients that are not handled correctly, you can specify their User-Agent headers in this file.

A default example of this file can be found in the SER distribution under <ser-sources>/modules/mediaproxy/config/sip-asymmetric-clients. This example file has been copied and renamed to the path specified on this line.

13 Mediaproxy may need to know when a SIP UA is asymmetric with respect to its RTP port. If you find asymmetric clients that not handled correctly you can specify their User-Agent headers in this file.

A default example of this file can be found in the SER distribution under <ser-sources>/modules/mediaproxy/config/rtp-asymmetric-clients. This example file has been copied and renamed to the path specified on this line.

14 When SIP clients attempt to REGISTER with our SIP proxy we need a way to tell the registrar module to store NAT information for any particular UA. We do this by using flag 6, which has been arbitrarily chosen (but defined earlier in the loadmodule parameter section). We could have specified another integer here, but flag 6 seems to be the accepted standard for nat_flag.

If the nat_flag is set before calling the save() function to store contact information, SER will preserve the NAT contact information as well as set the flags column in the MySQL location table. By doing so, we can call lookup(location) when processing messages and flag 6 will be set for NATed clients.

15 If the received message is an INVITE and it is from a SIP client that is behind a NAT we need to embed a special NAT flag, which will be returned to our SIP proxy in the case of a re-INVITE. This embedded flag can then be located in our loose route processing logic to determine if the message originator is NATed or not.

16 If the message originator is NATed we specify the Record-Route header explicitly by using the record_route_preset() function. We pass to this function the IP address of our SIP proxy. If your SIP proxy sits behind a router that has Application Level Gateway (ALG) support, such as a Cisco 3600 series, then you will use the physical RFC1918 address of your SIP proxy here because the ALG-enabled router will rewrite private IP addresses.

NOTE: For those that are familiar with rtpproxy you may be wondering why this step is necessary since rtpproxy doesnt require the embedded ;nat=yes tag. The reason mediaproxy requires this and rtpproxy does not is that the use_media_proxy() function, which is introduced later in this example, will set up a new RTP proxy channel if you call the function and the <Call-ID> header is not found it mediaproxys list of active sessions.

The corresponding function in rtpproxy has the ability to instruct rtpproxy to only take action if a previous <Call-ID> header is found in the list of active rtpproxy sessions.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This difference also affects the loose_route() code block because mediaproxy users must look for the ;nat=yes tag when processing re-INVITE messages and rtpproxy users do not. So from this aspect, rtpproxy is somewhat easier to use.

17 If the message is not an INVITE from a NATed SIP client and it is not a REGISTER then we just call record_route() as normal to ensure that messages return to our SIP proxy from upstream or downstream SIP proxy servers or PSTN gateways.

18 Anytime we receive a BYE or CANCEL message we should assume that it is for a call that has been set up with mediaproxy. So here we just attempt to tear down the RTP proxy session. It is perfectly safe to call end_media_session(), even for calls that were not RTP proxied.

19 Tell mediaproxy to end the session for the current call.

20 Our NAT traversal requirements must handle re-INVITE messages in a special manner to prevent RTP media streams from dropping off during a re-INVITE. So we do special re-INVITE NAT processing here.

In order to ensure that we are dealing only with an actual re-INVITE, we must make sure the has_totag() function returns TRUE and loose_route() is also TRUE. The reason for this is that it is possible for an original INVITE message to include predefined route headers, which would cause loose_route() to return TRUE. Therefore the has_totag() is checked because only connected calls will have a tag= entry in the <To> header (ie, calls where a 200 OK as been generated by the callee).

In other words this new security check is based on the fact that established SIP dialogs will have a "totag" whereas calls in the process of being established will not. To ensure that our loose routing logic is not exposed to malicious users we make sure that INVITE and REFER messages are only accepted for established dialogs.

21 If the message is an INVITE then we need to challenge the message sender for credentials. If the message sender cannot provide valid credentials then SER will reply with a 403 error message.

NOTE: This current example requires that the caller and callee are registered with the SIP router. Future examples will expand on this section to allow "trusted" SIP devices, such as a PSTN gateway.

22 Now we check the NAT status of the re-INVITE to see if the message originator is NATed or not by calling client_nat_test("3"). We also search for a <Route> header that contains the ";nat=yes" embedded tag which would have been included by our record_route_preset() discussed earlier. If the ;nat=yes tag is found, then the caller is NATed.

23 If the message sender is NATed or the re-INVITE contains the ;nat=yes flag, we set flag 6 for later reference. This flag can then be checked in the reply_route.

24 In order to proxy RTP streams we just call use_media_proxy(). This will communicate to the external mediaproxy server causing it to open UDP ports for both clients, or maintain an existing RTP proxy session for an existing call, based on the <Call-ID> header. Calling use_media_proxy() causes the SDP payload to be rewritten with the IP address of the mediaproxy server and the allocated ports.

25 In the event that our message is no longer to be handled by our SIP router, we call our NAT handling route block to enable mediaproxy if needed before sending the message to its destination.

26 We now explicitly handle CANCEL messages. CANCEL messages can be safely processed with a simple call to t_relay() because SER will automatically match the CANCEL message to the original INVITE message. So here we just route the message to the default message handler.

27 Enable mediaproxy if needed before sending the message to its destination.

28 When dealing with NATed clients we must correctly handle response messages that may be heading back to the client. These response messages are accessible in SER by using a reply_route block.

SER allows you to specify multiple reply_route blocks which can perform many tasks. Here we specify that any reply messages must be passed to reply_route[1] which is defined at the end of the ser.cfg file.

In order to invoke a reply_route, you simply need to set the handler prior to calling t_relay(), as we have done here.

29 If the message could not be sent and it is an INVITE or ACK then we should attempt to release any previously established mediaproxy session.

30 If client_nat_test() returns true then we must set flag 6 to inform the registrar module that the client is NATed.

Also note that we must only invoke client_nat_test() if the SIP message being processed contains an actual <Contact:> header. Otherwise an error will be written to syslog.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

NOTE: An interesting side note here is that the regular expression for the <Contact:> header is ^Contact:[ ]** which reads like this;

If the line starts with the text Contact: and is followed by any number of spaces (ie: [ ]*) and then followed by an asterisk (ie: *). The reason for this is that a SIP client can include the header <Contact: *> just the same as <Contact:'space'*> and both are valid. By testing for one or more white space characters we catch all formatting styles.

NOTE: The Contact: header will contain an asterisk "*" character when a SIP client is requesting to be "un-registered" from the SIP proxy. In fact, many SIP clients can and/or will send a REGISTER message with a "Contact: *" header when they are rebooted.

31 If a client is indeed NATed, then we must inform the SER registrar module that it needs to store the actual IP address and port that the SIP message came from. This information is then used by subsequent calls to lookup(location) in order to find the public IP address of a SIP client that is behind a NAT device.

32 Fix_nated_register() is used specifically for processing REGISTER messages from NATed clients. This is very different from fix_nated_contact() because the former will not alter the URI in the <Contact:> header whereas the latter will.

Fix_nated_register() will only append parameters to the <Contact:> header URI, which follows RFC3261 Section 10.3 on handling REGISTER messages. If we had used fix_nated_contact() here then you will likely have compatibility problems where SIP UAs do not honor the 200 OK response that SER replies with upon successful REGISTRATION. This would then cause the SIP client to loose its registration.

33 Force_rport() adds the received IP port to the top most via header in the SIP message. This enables subsequent SIP messages to return to the proper port later on in a SIP transaction.

34 Invite messages have slightly different NAT testing requirements than REGISTER messages. Here we only test to see if the SIP message has an RFC1918 IP address in the <Contact:> header and whether or not the SIP UA contacted SER on a different IP address or port from what is specified in the via header.

35 If it is determined that the SIP client is NATed, then we set flag 7 for later reference.

NOTE: This client_nat_test() only determines if the message sender is behind a NAT device. At this point in the ser.cfg file we have not determined if the message recipient is NATed or not.

36 Add the received port to the VIA header.

37 Now we rewrite the messages <Contact:> header to contain the IP address and port of the public side of the NATed SIP client.

38 Enable mediaproxy if needed before sending the message to its destination.

39 Now we find the contact record for the party being called.

NOTE: A subtle, but very important, detail here is that flag 6 will be set by the call to lookup(location) in route[2] if the party being called is found in the MySQL location table and is NATed. The reason flag 6 will be set is because we specified this as the registrar modules nat_flag parameter.

40 Enable mediaproxy if needed before sending the message to its destination.

Now that we have taken care of all the NAT related items, we can safely send the INVITE message to its destination.

41 Route[4] is a convenience route block which enabled mediaproxy if either the message sender (flag 7) or the message recipient (flag 6) are NATed.

42 Flag 8 is used to make sure the mediaproxy doesn't get invoked more than once for our call. If mediaproxy were to erroneously be called more than once then the SIP message would end up with a corrupted SDP payload because the call to use_media_proxy() would alter the message incorrectly.

If flag 8 is not set then set it to prevent calling route[4] again.

43 Here we introduce a reply_route handler. A reply_route is defined just like any other block in SER. The only difference is that it is called onreply_route.

Any message that is passed to this block will be returning to the original sender. You can think of these messages as the response to the original request that the caller made. The types of messages that will appear here will have an integer response code, much like HTTP response codes. Examples here would be 200, 401, 403, and 404.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

44 In this ser.cfg we are only interested in response codes of 180, 183, and all 2xx messages for NATed clients. We can check the status as shown with a regular expression. If any of these response codes are found then this statement will be TRUE.

An important thing to note is that we can check flags set in the other route blocks because their scope is still valid. So our caller and callee NAT flags are still accessible. If flag 6 is set then the caller is NATed, and if flag 7 is set then the callee is NATed.

45 We can only call use_media_proxy() for SIP messages that have a valid <Contact> parameter in the SDP payload. So here we test to make sure the c= parameter is valid by simply checking the SDP payload length. We assume that if we have an SDP payload then we will have a c= parameter and can call use_media_proxy().

If we were to simply call use_media_proxy() then we would likely see errors in syslog.

46 Finally we rewrite the messages <Contact:> header to contain the IP address and port of the public side of the NATed SIP client.

Using the Mediaproxy Transparent NAT Traversal ser.cfg

We have now made SER 100% NAT aware while keeping all of our SIP clients unaware of NAT. To test this new NAT functionality make sure you have an instance of mediaproxy running on the same physical server as SER.

Once mediaproxy is running you can start SER and register your SIP clients as normal. It should not matter whether the SIP client is behind a NAT device or not.

To see if a calls RTP streams have been routed to mediaproxy you can execute the Python script located at /usr/local/mediaproxy/sessions.py (assuming you installed mediaproxy at that location.)

You can also install the web monitoring tool located at /usr/local/mediaproxy/web/ if you have an Apache web server running on your SIP proxy. A screen shot of this web monitoring tool is available at http://www.ag-projects.com/docs/MediaSessions.pdf

Handling of NAT using RTPproxy

What this ser.cfg will do:

1. Introduce specialized processing which is necessary when dealing with SIP clients that are behind a NAT device such as a DSL router or corporate firewall

RTPproxy is one of two NAT traversal solutions for SER, the other is mediaproxy just descussed in the previous section. Both RTPproxy and mediaproxy are known as far-end NAT traversal solutions which means that they handle all aspects of NAT at the SIP proxy location rather than at the SIP client location.

There are many advantages to handling NAT issues at the SIP proxy server, the primary benefit being that SIP client configuration is much simpler. Some of the important features of RTPproxy are:

1. Is called by nathelper, the most configurable NAT helper modules (mediaproxy module is the other one)
2. Can handle almost twice as many calls as mediaproxy on the same hardware
3. Developed in C and is thus easily extendable for most programmers
4. RTPproxy can be installed on a remote server which is not running SER

RTPproxy is a standalone software and is thus not a part of the sip_router CVS and distribution. However, it can be found in the same CVS repository as SER (on the same level as sip_router). Just replace the sip_router directory name with rtpproxy (or use the ONsip.org Getting Started source package). The SER distribution only includes the glue which gives SER the ability to communicate with a running instance of RTPproxy. This glue is known as the nathelper module.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

NOTE: In order for RTPproxy to function properly it must be configured to listen on a public IP address. Also, in most real world configurations, RTPproxy will not be installed on the SER server, but on a remote machine. Refer to the appendix for information on installing rtpproxy.

```
debug=3
fork=yes❶
log_stderror=no❷

listen=192.0.2.13     # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"
fifo_db_url="mysql://ser:heslo@localhost/ser"

loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"❸
loadmodule "/usr/local/lib/ser/modules/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/nathelper.so"❹
loadmodule "/usr/local/lib/ser/modules/textops.so"❺

modparam("auth_db|uri_db|usrloc", "db_url", "mysql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")

modparam("nathelper", "natping_interval", 30)❻
modparam("nathelper", "ping_nated_only", 1)❼
modparam("nathelper", "rtpproxy_sock", "unix:/var/run/rtpproxy.sock")❽

modparam("usrloc", "db_mode", 2)

modparam("registrar", "nat_flag", 6)❾

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

route {

  # -----------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    break;
  };

  if (msg:len > max_len) {
    sl_send_reply("513", "Message Overflow");
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  break;
};

# -----------------------------------------------------------------
# Record Route Section
# -----------------------------------------------------------------
if (method!="REGISTER") {
  record_route();
};

if (method=="BYE" || method=="CANCEL") { ❿
  unforce_rtp_proxy();
}

# -----------------------------------------------------------------
# Loose Route Section
# -----------------------------------------------------------------
if (loose_route()) { 11

  if ((method=="INVITE" || method=="REFER") && !has_totag()) {
    sl_send_reply("403", "Forbidden");
    break;
  };

  if (method=="INVITE") {

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

    consume_credentials();

    if (nat_uac_test("19")) {
      setflag(6);
      force_rport();
      fix_nated_contact();
    };
    force_rtp_proxy("l");
  };

  route(1);
  break;
};

# -----------------------------------------------------------------
# Call Type Processing Section
# -----------------------------------------------------------------
if (uri!=myself) {
  route(4);12
  route(1);
  break;
};

if (method=="ACK") {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      route(1);
      break;
    } if (method=="CANCEL") { 13
      route(1);
      break;
    } else if (method=="INVITE") {
      route(3);
      break;
    } else  if (method=="REGISTER") {
      route(2);
      break;
    };

    lookup("aliases");
    if (uri!=myself) {
      route(4); 14
      route(1);
      break;
    };

    if (!lookup("location")) {
      sl_send_reply("404", "User Not Found");
      break;
    };

    route(1);
  }

  route[1] {

    # -----------------------------------------------------------------
    # Default Message Handler
    # -----------------------------------------------------------------

    t_on_reply("1"); 15

    if (!t_relay()) { 16
      if (method=="INVITE" && isflagset(6)) {
        unforce_rtp_proxy();
      };
      sl_reply_error();
    };
  }

  route[2] {

    # -----------------------------------------------------------------
    # REGISTER Message Handler
    # ----------------------------------------------------------------

    17if (!search("^Contact:[ ]*\*") && nat_uac_test("19")) {
      setflag(6);
      fix_nated_register();
      force_rport();
    };

    sl_send_reply("100", "Trying");
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  if (!www_authorize("","subscriber")) {
    www_challenge("","0");
    break;
  };

  if (!check_to()) {
    sl_send_reply("401", "Unauthorized");
    break;
  };

     consume_credentials();

  if (!save("location")) {
    sl_reply_error();
  };
}

route[3] {

  # -----------------------------------------------------------------
  # INVITE Message Handler
  # -----------------------------------------------------------------

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

     consume_credentials();

  if (nat_uac_test("19")) { 18
    setflag(6);
     }

  lookup("aliases");
  if (uri!=myself) {
    route(4); 19
    route(1);
    break;
  };

  if (!lookup("location")) {
    sl_send_reply("404", "User Not Found");
    break;
  };

  route(4); 20
  route(1); 21
}

22route[4] {

  # -----------------------------------------------------------------
  # NAT Traversal Section
  # -----------------------------------------------------------------
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    if (isflagset(6)) {
      force_rport();
      fix_nated_contact();
      force_rtp_proxy();
    }
  }

23onreply_route[1] {

    if (isflagset(6) && status=~"(180)|(183)|2[0-9][0-9]") { 24
        if (!search("^Content-Length:[ ]*0")) { 25
        force_rtp_proxy();
      };
    };

  26if (nat_uac_test("1")) {
      fix_nated_contact();
    };
  }
```

RTPproxy Transparent NAT Traversal ser.cfg Analysis

❶ Up until now we have run SER as a foreground process. From this point forward we will run SER as a daemon. The fork directive tells the SER daemon to run in the background. This is a requirement for using the init.d start script shown in the appendix.

❷ Since we are running SER as a background process we must set the log_stderror directive equal to no in order to not keep SER in the foreground.

❸ The uri module is introduced here to access the has_totag() function. This function is necessary for processing re-INVITEs and is described below.

❹ Here we load the nathelper module. Nathelper has functionality for rewriting SIP messages. It also communicates with rtpproxy, which is a standalone program. Make sure that you have rtpproxy started before you start SER. It can be found in the same cvs repository as SER or you can use the SER package provided on http://www.onsip.org/ under Downloads. Start rtpproxy without parameters and make sure it is running using the 'ps -ax | grep rtpproxy' command. If SER cannot communicate with rtpproxy, you will get errors in /var/log/messages when staring SER."

❺ The textops module provide utility functions for manipulating text, searching for substrings, and checking for the existence of specific header fields.

❻ Unlike mediaproxy, rtpproxy does not have a built-in ping keep-alive capability (i.e. sending packets regularly to the clients to make sure the NAT binding is kept in place and thus incoming SIP messages or RTP streams can come in). However, this functionality is instead built into nathelper.

The natping_interval is a very crucial setting which controls how often our ser proxy will ping registered SIP UAs. Most NAT devices will only hold port bindings open for a minute or two, so we specify 30 seconds here.

This causes ser to send a 4-byte UDP packet to each SIP UA every 30 seconds. This is all that is required to keep NATed clients alive.

NOTE: Some NAT devices have been reported to not allow incoming keep-alives. Thus, many UAs have their own implementations of keep-alive. If you experience one-way audio problems after a while, you may have run into this problem. The only solution is to turn on user client keep-alives.

❼ Nathelper can ping all UAs or only the ones that have been flagged with a special nat flag (see further below). We only need to ping NATed clients as we assume all other clients have public addresses and keep-alive is not needed.

❽ Ser and rtpproxy communicate via a standard Unix socket, with the default socket path specified here.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

NOTE: If you change the socket path here, you must be sure to start rtpproxy with the parameter: s unix:socketpath

You can also start rtpproxy with f to let it run without forking. It will then show you what is happening.

❾ When SIP clients attempt to REGISTER with our SIP proxy we need a way to tell the registrar module to store NAT information for the UA. We do this by using flag 6, which has been arbitrarily chosen (but defined earlier in the moduel parameter section). We could have specified another integer here, but 6 has become the accepted default for nat_flag.

When nat_flag is set before calling the save() function to store contact information, then ser will also preserve the NAT contact information as well as set the flags column in the MySQL location table. By doing so, we can call lookup(location) when processing messages and flag 6 will be set for NATed clients.

❿ When we decide that a particular call must be proxied through our proxy server, we must also ensure that the call is torn down when a call is hung up (BYE) or cancelled (CANCEL).

11 Our NAT traversal requirements must handle re-INVITE messages in a special manner to prevent RTP media streams from dropping off during a re-INVITE. So we do special re-INVITE NAT processing here. The has_totag() function will return true if the SIP message has a To header field. If it has, the message be will be in-dialog.

force_rtp_proxy() has an l (lookup) parameter. When this parameter is specified, rtpproxy will only proxy the call if a session already exists. The reason for doing this is that rtpproxy cannot really know that this is a re-INVITE and we want to avoid unnecessary proxying calls that do not need it. All re-INVITEs will call force_rtp_proxy(l), but only existing proxied sessions will be re-proxied. We cannot put the force_rtp_proxy() call inside the nat_uac_test check, as this will prevent proxying to happen if the re-INVITE is sent from a non-NATed UA to a NATed UA (because nat_uac_test() will be false).

The other things we do here are: if NAT is detected for the UA sending the INVITE, set the NAT flag, so we know this is a NATed call (setflag(6)), add the received port to top-most via header (force_rport), as well as rewrite the Contact header with the public IP address of the user client (fix_nated_contact).

12 In the event that our message is no longer to be handled by our SIP router, we call our NAT handling route block to enable rtpproxy if needed before sending the message to its destination.

13 We now explicitly handle CANCEL messages. CANCEL messages can be safely processed with a simple call to t_relay() because SER will automatically match the CANCEL message to the original INVITE message. So here we just route the message to the default message handler.

14 Enable rtpproxy if needed before sending the message to its destination.

15 When dealing with NATed clients, we must correctly handle response messages that are destined back to the client. These response messages are accessible in ser by using a reply_route block.

Ser allows you to specify multiple reply_route blocks, which can perform many tasks. Here we specify that any reply messages must be passed to reply_route["1"], which is defined at the end of the ser.cfg file.

In order to invoke a reply_route, you simply need to set the handler prior to calling t_relay(), as we have done here.

16 If something goes wrong and we have called force_rtp_proxy (so that rtpproxy set up a new proxy call), we need to tell rtpproxy to tear it down again using unforce_rtp_proxy().

17 When SIP clients attempt to REGISTER with our SIP proxy we need a way to tell the registrar module to store NAT information this particular UA. We do this by using flag 6, which has been arbitrarily chosen (but defined earlier in the module parameter section). We could have specified another integer here, but 6 seems to be the accepted standard for nat_flag.

When nat_flag is set before calling the save() function to store contact information, then ser will also preserve the NAT contact information as well as set the flags column in the MySQL location table. By doing so, we can later call lookup(location) when processing messages and flag 6 will be set for NATed clients.

In order to determine whether a SIP client is NATed we use the nathelper function nat_uac_test(), which accepts an integer as a parameter. This parameter specifies which part of the SIP message to examine for determining the NAT status of the client. 19 indicates a common subset of tests, and are the recommended tests.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

NOTE: These are the tests you can run using nat_uac_test (in the order they are employed):

1. (16) Is received port (source port of the packet) different from the port specified in the top-most Via header? (this can be used to detect some broken STUN implementations)
2. (2) Is the received IP address (source IP of the packet) different from the IP address in the top-most Via header?
3. (1) Does the Contact header contain an RFC1918 address? RFC1918 addresses are 10.0.0.0/24, 172.16.0.0.0/20, and 192.168.0.0/16
4. (8) Does the SDP payload of an INVITE or OK message contain an RFC1918 address?
5. (4) Does the top-most Via header contain an RFC1918 address?

The numbers in paranthesis indicate the number to be used to invoke the test in nat_uac_test(). To invoke more than one test, add up the numbers of the tests you want to run. So, as we use nat_uac_test(19) in our script, it means that test #1 (16) + #2 (2) + #3 (1) = 19, are run.

WARNING: Before changing the tests, you should know what you are doing. You should scrutinize the SIP message coming from your UAs to determine which tests will be true.

You will notice that we first call the search() function to find out if the <Contact:> header contains an asterisk or not. We only test for NAT if the <Contact:> header is not an asterisk.

NOTE: If the <Contact:> header is an asterisk rather than a SIP URI, then this indicates that the SIP client is attempting to unregister with the SIP proxy. The ser save() function has special provisions to remove all binds for the AOR in this case. It is important to understand that we cannot accurately determine the NAT status of a client when the contact information is missing.

18 We test to see if the user client is behind NAT, if so, we set the NAT flag (6). This flag is used further below.

19 Enable rtpproxy if needed before sending the message to its destination.

20 Enable rtpproxy if needed before sending the message to its destination.

21 Now that we have taken care of all the NAT related items, we can safely send the INVITE message to its destination.

22 Route[4] is a convenience route to enable rtpproxy.

If a client is NATed, we must do the following things: Add the received (public) port to the top-most Via header (force_rport), rewrite the Contact header with the public IP address and port of the NAT in front of the user client (fix_nated_contact), and set up the proxying using force_rtp_proxy(). Nathelper will then communicate to rtpproxy, which will allocate RTP (UDP) ports and the SDP payload of the INVITE will be rewritting (see the introductory section on NATing for details).

23 Here we introduce a reply_route. A reply_route is defined just like any other block in ser. The only difference is that it is called onreply_route.

Any message that is passed to this block will be returning to the original sender. You can think of these messages as the response to the original request that the caller made. The types of messages that will appear here will have an integer response code, much like HTTP response codes. Examples here would be 200, 401, 403, and 404.

24 In this ser.cfg we are only interested in response codes of 180, 183, and all 2xx messages for NATed clients. We can check the status as shown with a regular expression. If any of these response codes are found then this statement will be TRUE.

An important thing to note is that we can check flags set in the other route blocks because their scope is still valid. So our caller and callee NAT flags are still accessible.

25 We can only call force_rtp_proxy() for SIP messages that have a valid <contact> parameter in the SDP payload. So here we test to make sure the c= parameter is valid by simply checking the SDP payload length. We assume that if we have an SDP payload then we will have a c= parameter and can call force_rtp_proxy ().

26 Before our reply_route processing is complete we have one more NAT test to try. Here we make sure the <Contact:> header does not contain an RFC1918 IP address. If it does, then we replace it with the public IP address and port of the NAT in front of the user client.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 9. PSTN Gateway Connectivity

What this ser.cfg does:

1. Adds the ability to connect to the PSTN for inbound and outbound calling

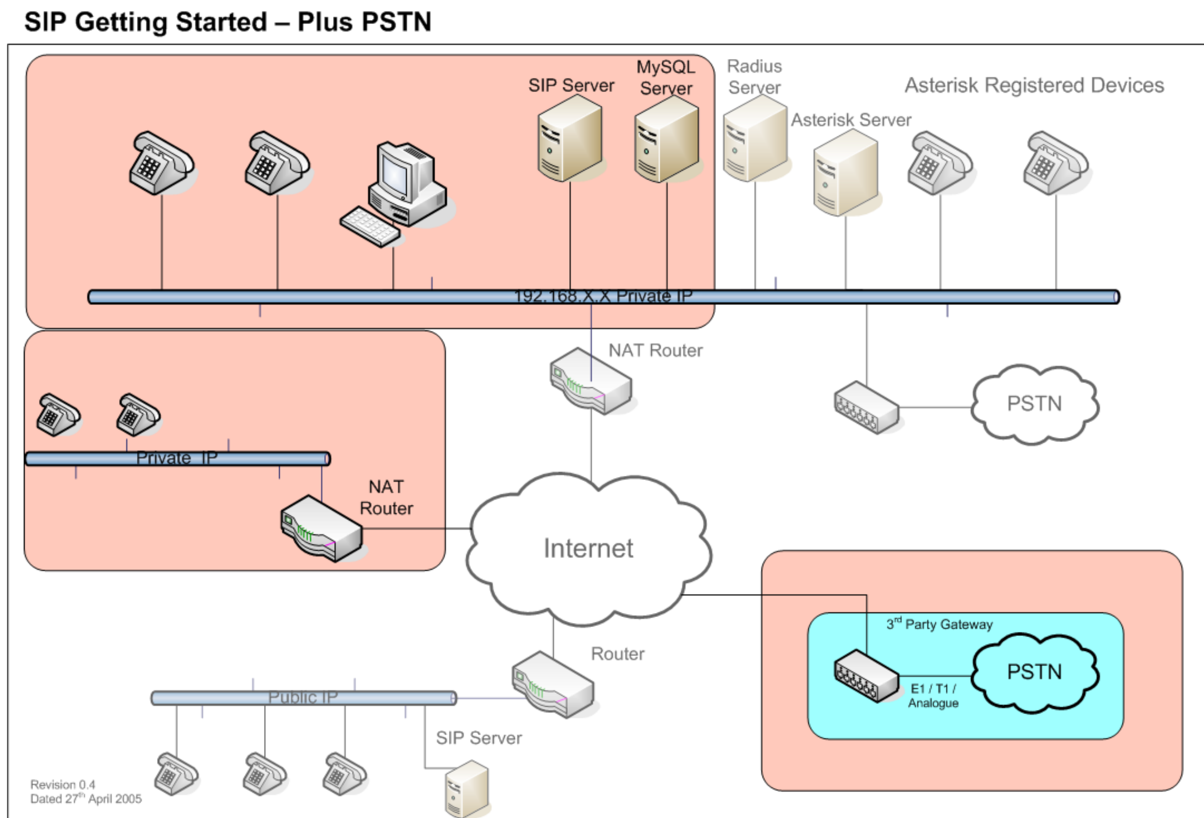
Up until now we have focused on SIP only calls, which means you could only send and receive calls between IP phones on your SIP proxy. But most people will need to call regular land lines as well as receive calls from them.

This is accomplished by means of a PSTN Gateway. SER itself does not provide PSTN access. You must install external equipment for this. A very popular and cost effective solution is a Cisco AS5300. The Cisco AS5300 is a multi-function chassis which allows you to plug in many different feature cards. In order for the Cisco AS5300 to operate as a PSTN gateway you must have a T1/E1 interface board and a Voice Card. The T1/E1 card is used to connect the AS5300 to the actual PSTN by means of a PRI from your local telephone company. The voice card contains DSP (digital signal processing) chips that convert audio streams between the SIP world and the PSTN world.

NOTE: Some users have asked how do I get a real telephone number for my SIP phone and the answer to that question is to get a PSTN gateway. When your local telephone company installs your PRI they will be able to allocate DIDs (direct inward dial, or telephone numbers) to you. You can then assign the DIDs as usernames in the MySQL subscriber table and when the DID is called from the PSTN it will cause your PSTN gateway to generate an INVITE message and send it to SER which will ring the SIP phone just like a normal SIP-to-SIP call.

When dealing with PSTN calls, you can actually think of the SIP client as one end of the call and the PSTN gateway as the other end. Many people get confused on this minor detail because they fail to understand that the actual PSTN gateway is indeed a SIP client itself.

Figure 9.1. Reference Design - PSTN



XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

In order to call a PSTN phone from your SIP phone SER receives the INVITE message as usual. If the R-URI (request URI) is determined to be a PSTN number then SER changes the host IP of the R-URI to be that of the PSTN gateway and then it relays the message to the PSTN gateway. After this, the call behaves just like a normal SIP-to-SIP call.

When receiving a call from the PSTN, the PSTN gateway will send a new INVITE message to SER and it will be processed just like any normal SIP-to-SIP call.

So now that you have a brief understanding of how SIP-to-PSTN and PSTN-to-SIP calling works, lets take a look at the PSTN-enabled ser.cfg configuration file.

NOTE: This ser.cfg file does not take in to consideration user ACLs. This means that all registered user accounts will have PSTN access. We will introduce ACL control in a future example.

```
debug=3
fork=yes
log_stderror=no

listen=192.0.2.13     # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"
fifo_db_url="mysql://ser:heslo@localhost/ser"

loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/domain.so"
loadmodule "/usr/local/lib/ser/modules/mediaproxy.so"
loadmodule "/usr/local/lib/ser/modules/nathelper.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"
loadmodule "/usr/local/lib/ser/modules/avpops.so"❶
loadmodule "/usr/local/lib/ser/modules/permissions.so"❷

modparam("auth_db|permissions|uri_db|usrloc",
   "db_url",  "mysql://ser:heslo@localhost/ser")❸
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")

modparam("nathelper", "rtpproxy_disable", 1)
modparam("nathelper", "natping_interval", 0)

modparam("mediaproxy","natping_interval", 30)
modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")
modparam("mediaproxy","sip_asymmetrics","/usr/local/etc/ser/sip-clients")
modparam("mediaproxy","rtp_asymmetrics","/usr/local/etc/ser/rtp-clients")
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
modparam("usrloc", "db_mode", 2)

modparam("registrar", "nat_flag", 6)

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

modparam("tm", "fr_inv_timer", 27)❹
modparam("tm", "fr_inv_timer_avp", "inv_timeout")

modparam("permissions", "db_mode", 1)❺
modparam("permissions", "trusted_table", "trusted")❻

route {

  # -----------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    break;
  };

  if (msg:len > max_len) {
    sl_send_reply("513", "Message Overflow");
    break;
  };

  # -----------------------------------------------------------------
  # Record Route Section
  # -----------------------------------------------------------------
  if (method=="INVITE" && client_nat_test("3")) {
    # INSERT YOUR IP ADDRESS HERE
    record_route_preset("192.0.2.13:5060;nat=yes");
  } else if (method!="REGISTER") {
    record_route();
  };

  # -----------------------------------------------------------------
  # Call Tear Down Section
  # -----------------------------------------------------------------
  if (method=="BYE" || method=="CANCEL") {
    end_media_session();
  };

  # -----------------------------------------------------------------
  # Loose Route Section
  # -----------------------------------------------------------------
  if (loose_route()) {

    if ((method=="INVITE" || method=="REFER") && !has_totag()) {
      sl_send_reply("403", "Forbidden");
      break;
    };

    if (method=="INVITE") {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      if (!allow_trusted()) {❼

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

        consume_credentials();
      };

      if (client_nat_test("3")||search("^Route:.*;nat=yes")){
        setflag(6);
        use_media_proxy();
      };
    };

    route(1);
    break;
};

# -----------------------------------------------------------------
# Call Type Processing Section
# -----------------------------------------------------------------
❽if (!is_uri_host_local()) { ❾
  if (is_from_local() || allow_trusted()) { ❿
    route(4);⓫
    route(1);
  } else {
    sl_send_reply("403", "Forbidden");⓬
  };
  break;⓭
};

if (method=="ACK") {
  route(1);
  break;
} if (method=="CANCEL") {
  route(1);
  break;
} else if (method=="INVITE") {
  route(3);
  break;
} else  if (method=="REGISTER") {
  route(2);
  break;
};

lookup("aliases");
if (!is_uri_host_local()) { ⓮
  route(4);
  route(1);
  break;
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    if (!lookup("location")) {
      sl_send_reply("404", "User Not Found");
      break;
    };

    route(1);
  }

  route[1] {

    # -----------------------------------------------------------------
    # Default Message Handler
    # -----------------------------------------------------------------

    t_on_reply("1");

    if (!t_relay()) {

      if (method=="INVITE" || method=="ACK") {
        end_media_session();
      };

      sl_reply_error();
    };
  }

  route[2] {

    # -----------------------------------------------------------------
    # REGISTER Message Handler
    # -----------------------------------------------------------------

    sl_send_reply("100", "Trying");

    if (!search("^Contact:[ ]*\*") && client_nat_test("7")) {
      setflag(6);
      fix_nated_register();
      force_rport();
    };

    if (!www_authorize("","subscriber")) {
      www_challenge("","0");
      break;
    };

    if (!check_to()) {
      sl_send_reply("401", "Unauthorized");
      break;
    };

    consume_credentials();

    if (!save("location")) {
      sl_reply_error();
    };
  }

  route[3] {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  # -----------------------------------------------------------------
  # INVITE Message Handler
  # -----------------------------------------------------------------

  if (client_nat_test("3")) {
    setflag(7);
    force_rport();
    fix_nated_contact();
  };

  if (!allow_trusted()) { 15

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

    consume_credentials();
  };

  if (uri=~"^sip:1[0-9]{10}@") { 16
    strip(1);
  };

  lookup("aliases");
  if (!is_uri_host_local()) { 17
    route(4);
    route(1);
    break;
  };

  if (uri=~"^sip:011[0-9]*@") {   # International PSTN 18
    route(4);
    route(5);
    break;
  };

  if (!lookup("location")) {
    if (uri=~"^sip:[0-9]{10}@") { # Domestic PSTN 19
      route(4);
      route(5);
      break;
    };

    sl_send_reply("404", "User Not Found");
    break;
  };

  route(4);
  route(1);
}

route[4] {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    # -----------------------------------------------------------------
    # NAT Traversal Section
    # -----------------------------------------------------------------

    if (isflagset(6) || isflagset(7)) {
      if (!isflagset(8)) {
        setflag(8);
        use_media_proxy();
      };
    };
  }

20route[5] {

    # -----------------------------------------------------------------
    # PSTN Handler
    # -----------------------------------------------------------------

    rewritehost("192.0.2.245");21 # INSERT YOUR PSTN GATEWAY IP ADDRESS

    avp_write("i:45", "inv_timeout");22

    route(1);
  }

  onreply_route[1] {

    if ((isflagset(6) || isflagset(7)) &&
        (status=~"(180)|(183)|2[0-9][0-9]")) {

      if (!search("^Content-Length:[ ]*0")) {
        use_media_proxy();
      };
    };

    if (client_nat_test("1")) {
      fix_nated_contact();
    };
  }
```

PSTN Gateway Connectivity ser.cfg Analysis

❶ The avpops module is a very power library for manipulating SIP message headers, storing temporary or persistent message data, and comparing data items. Avpops is also used to dynamically alter certain parts of SER itself. We use avpops in this example to dynamically change the amount of time SER will wait for an INVITE message to be forwarded to its destination.

The reason we need to do this is that a SIP-to-SIP call generally connects much faster than a SIP-to-PSTN call. Therefore we will allow SER to wait a while longer when calling a PSTN telephone.

❷ The permissions module gives SER access to the trusted MySQL table. The trusted table is acts as a repository of privileged devices. We must add our PSTN gateway device to the trusted table to prevent SER from challenging the PSTN gateway for authentication credentials.

❸ The permissions module needs to access MySQL so we add it to the db_url parameter list.

❹ The default INVITE timer will be allowed 27 seconds to connect. If we are going to call a PSTN destination then we will set the inv_timeout AVP to a higher value in order to allow additional time for the call to connect. This is show in the PSTN route handler.

❺ The permissions module needs to know that it should connect to the MySQL database to find its data.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

❻ The trusted_table parameter informs the permissions module as to which MySQL table it will read from. In this case the table called trusted will be used.

❼ Now that we can receive INVITE messages from registered SIP clients as well as our PSTN gateway, we need to challenge INVITE messages in a more specialized fashion. The allow_trusted() function is provided by the permissions module. It will access the MySQL table called "trusted" to get a list of privileged IP addresses. When the PSTN gateway sends an INVITE to SER, the allow_trusted() should return TRUE to prevent SER from replying with a 401 Unauthorized message.

As a general rule, the PSTN gateway should always be allowed to send messages to SER without being challenged for credentials, however, all INVITE messages being sent to the PSTN gateway should be challenged to prevent toll fraud for which the PSTN gateway owner is responsible for.

NOTE: It is very important to understand the significance of using proxy_authorize() and allow_trusted() in the loose_route() code block. A re-INVITE should only be processed by SER if it originated from a SIP client that can be **authenticated or trusted** by SER. Authentication is verified with the call to proxy_authorize(), which will check the MySQL subscriber table whereas trust relationships are tested with the call to allow_trusted(), which will check the MySQL trusted table. A PSTN gateway which cannot authenticate against SER must be added to the MySQL trusted table, otherwise re-INVITE and REFER messages from the PSTN gateway will not be routed by SER.

❽ Our previous configuration files had a potential open relay problem where by SIP messages targeted at a domain that is not locally served could be relayed without being challenged. This was exposed in the following code

if (uri!=myself) {

route(4);

route(1);

break;

};

Now that our SIP proxy has PSTN access we need to close this potential open relay to prevent toll fraud.

❾ In previous ser.cfg examples we used the test condition (uri != myself) to determine if the SIP message was directed at our SIP router or not. From this point forward will will replace this test condition with a call to is_uri_host_local(). which will determine if the domain portion of the Request URI is a domain that our SIP proxy is responsible for. It does so by querying the MySQL table called domain. In order for this function to return TRUE for messages that our SIP proxy is responsible for, we must add our domain to the MySQL domain table. If your SIP proxy handles multiple domains, then you must add all domains to this table.

NOTE: The is_uri_host_local() function is exposed in the domain.so module.

❿ If the SIP message is not targeted at our domain, we must now check to see if the message sender belongs to our domain. We should only forward messages to external domains if the message sender belongs to our domain.

The is_from_local() function will query the MySQL domain table for the host name that appears in the <From:> header of the SIP message.

NOTE: The is_from_local() function is exposed in the domain.so module.

Another thing that we must allow is messages from our PSTN gateway. Since our PSTN gateway is a trusted device, it must be allowed to forward messages to external domains. This is accomplished with the call to allow_trusted().

11 If the message is permitted to leave our SIP proxy then we must take care of NAT traversal prior to relaying the message.

12 If the message is not permitted to leave our SIP proxy then we reply with a 403 message to inform the message sender that we are not an open relay.

13 Now that we have handled the message we simply stop processing.

14 Determine if the domain portion of the Request URI is a domain that our SIP proxy is responsible for.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

15 INVITE message must be challenged for credentials here just as they were in the loose_route() section of the main route code block.

NOTE: It is important to understand that INVITE messages can potentially pass through the SIP router in either route[3] or in the loose_route() section of the main route. Generally, original INVITE messages will be passed to route[3], whereas re-INVITE messages will get picked up by the loose_route() function. Therefore it is very important to secure both sections with the appropriate calls as demonstrated here.

NOTE: Failure to properly secure your SIP route could result in toll charges that you are responsible for, assuming your SIP router has access to the PSTN.

16 Many PSTN gateways do need or permit long distance prefixes when dialing. In North America we dial 1 + a 10-digit number to place a call outside of our local calling area. So we remove the leading 1 from the R-URI before sending the message to the PSTN gateway.

17 Determine if the domain portion of the Request URI is a domain that our SIP proxy is responsible for.

18 This ser.cfg assumes that international numbers are prefixed with 011 and that domestic numbers are 10 numeric digits.

Here we use regular expression matching to check the R-URI. If it starts with 011 and is followed by numeric digits only, then we have a match for an international PSTN destination. If a match is found then we route the message to the PSTN handler, route[4].

19 If the R-URI match 10 numeric digits then we know we have a domestic number and we route the INVITE message to the PSTN hander, route[4].

NOTE: We know we have a PSTN number at this point because the call to lookup(location) on the previous line returned FALSE.

20 Here we introduce route block 5 for handling PSTN destinations.

21 Rewritehost() is used to alter the R-URI so that the message will be relayed to our PSTN gateway. You must pass the IP address of your PSTN gateway to this function. SER will then send the INVITE message to your PSTN gateway properly when you call t_relay().

22 Avp_write() alters the amount of time allowed for a PSTN call to connect. If you recall we specified inv_timeout as the loadparam value for fr_inv_timeout.

This line specifies that we want to allow 45 seconds before timing out.

Using the PSTN Gateway Connectivity ser.cfg Example

Before you can use the PSTN enabled ser.cfg you we must take care of a few remaining items. The first thing to do is copy the following files which the permissions module will look for upon server startup.

cp [ser-source]/modules/permissions/config/permissions.allow /usr/local/etc/ser

cp [ser-source]/modules/permissions/config/permissions.deny /usr/local/etc/ser

The last thing to take care of before you can use the PSTN gateway example, namely add a record to the MySQL trusted table. To do so you can issue the following SQL command by opening up your MySQL terminal.

[#] mysql u ser p

-- enter your mysql password --

mysql> use ser;

mysql> insert into trusted values (192.0.2.245, any, ^sip:.*$);

NOTE: The above INSERT SQL statement tells SER to allow any protocol (udp or tcp) from IP 192.0.2.245 (the IP of the PSTN gateway) to send any message to SER without being challenged for credentials.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

NOTE: You must restart SER when altering the trusted table because the entries in this table are read only during server start up. Alternatively, you can use the FIFO command: serctl fifo trusted_reload

Now that your SIP router is PSTN Enabled you can start dialing domestic and international destinations. A word of caution is that many PSTN gateways perform their own accounting. You must be certain that SER is properly record-routing all messages to guarantee that BYE messages are sent to the PSTN gateway.

If a BYE message is not delivered to a PSTN gateway then you run the risk of additional toll charges because the PSTN circuit could be left open.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 10. Call Forwarding ser.cfg

What this ser.cfg will do:

1. Introduce forwarding concepts known as serial forking and redirection
2. Implement blind call forwarding
3. Implement forward on busy
4. Implement forward on no answer

Call forwarding can be achieved using one of two methods, namely serial forking and redirection. Both methods have advantages. Serial forking, as the name implies creates a new leg of the call (after the first fails) and sends a new INVITE message to the forwarded destination. On the other hand, redirection sends a reply message back to the caller and gives them the forwarded location. The caller then creates a brand new INVITE message with the forwarded destination in the R-URI and places the call.

Redirection must be used with caution if a PSTN gateway is accessible. The reason is that it is conceivable that SER could send a redirection response back to a PSTN gateway and the PSTN gateway could then send a new INVITE message to the forwarded destination and bypass SER, which could be an international call for which SER is unaware. This would result in toll charges that are not billable. The reason that SER could be unaware of the new call is that redirection is accomplished by replying with a 302 Temporarily Moved message. In the IP world, the intention is that the IP phone shall decide what to do, possibly even ask the user should I go ahead and make the call to the new destination? This way the user will acknowledge potential charges. However, there is no standard way 302 messages are handling in the interface between PSTN and IP and non-billable calls is a possibility.

Serial forking does not have this possible security exposure because SER will record route all messages, which means that even if the forwarded destination is a PSTN phone, SER will be able to account for the call and the toll charges, if any, would be billable to the subscriber.

This section implements three types of call forwarding:

1. Blind Call Forwarding All INVITE messages sent to the phone will be intercepted at the SIP router. The SIP router will create a new call leg and send the INVITE message to the forwarded destination. This means that the phone with blind call forwarding set will not even ring. This also means that it doesnt matter if the phone is registered with SER.
2. Forward On Busy If a phone replies to an INVITE message with a 486 Busy, SER will intercept the 486 response and create a new call leg, which will send an INVITE message to the forwarded destination.
3. Forward No Answer If a phone replies to an INVITE message with a 408 Request Timeout, SER will intercept the 408 message and create a new call leg, which will send an INVITE message to the forwarded destination.

Blind call forwarding is handled somewhat differently than forward no answer and forward on busy because blind call forwarding is altering an INVITE message for which the destination set has yet to be processed. This means that we can simply change the R-URI and relay the message.

The other two types of call forwarding require much more effort to implement because we must catch the error replies, alter the R-URI, and then append the forwarded contact location to the destination set and finally relay the message.

All call forwarding preferences are stored in the MySQL usr_preferences table. We will use the AVPOPS module to read a subscribers call forwarding settings. AVPOPS will also be used to alter the call sequence and perform serial forking that we require.

PLEASE NOTE! This configuration cannot recursive forwarding, i.e. forwarding to somebody who has forwarded. Adding support for this is complex, but is possible and is left as an exercise for the reader ;-)

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

In addition we have included an appendix describing the Call processing Language that allows more comprehensive control of a call.

```
debug=3
fork=yes
log_stderror=no

listen=192.0.2.13       # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"
fifo_db_url="mysql://ser:heslo@localhost/ser"

loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/domain.so"
loadmodule "/usr/local/lib/ser/modules/mediaproxy.so"
loadmodule "/usr/local/lib/ser/modules/nathelper.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"
loadmodule "/usr/local/lib/ser/modules/avpops.so"
loadmodule "/usr/local/lib/ser/modules/permissions.so"

modparam("auth_db|permissions|uri_db|usrloc",
  "db_url", "mysql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")

modparam("nathelper", "rtpproxy_disable", 1)
modparam("nathelper", "natping_interval", 0)

modparam("mediaproxy","natping_interval", 30)
modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")
modparam("mediaproxy","sip_asymmetrics","/usr/local/etc/ser/sip-clients")
modparam("mediaproxy","rtp_asymmetrics","/usr/local/etc/ser/rtp-clients")

modparam("usrloc", "db_mode", 2)

modparam("registrar", "nat_flag", 6)

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

modparam("tm", "fr_inv_timer", 27)
modparam("tm", "fr_inv_timer_avp", "inv_timeout")

modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
❶modparam("avpops", "avp_url", "mysql://ser:heslo@localhost/ser")
modparam("avpops", "avp_table", "usr_preferences")

route {

  # -----------------------------------------------------------------
  # Sanity Check Section
  # -----------------------------------------------------------------
  if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    break;
  };

  if (msg:len > max_len) {
    sl_send_reply("513", "Message Overflow");
    break;
  };

  # -----------------------------------------------------------------
  # Record Route Section
  # -----------------------------------------------------------------
  if (method=="INVITE" && client_nat_test("3")) {
    # INSERT YOUR IP ADDRESS HERE
    record_route_preset("192.0.2.13:5060;nat=yes");
  } else if (method!="REGISTER") {
    record_route();
  };

  # -----------------------------------------------------------------
  # Call Tear Down Section
  # -----------------------------------------------------------------
  if (method=="BYE" || method=="CANCEL") {
    end_media_session();
  };

  # -----------------------------------------------------------------
  # Loose Route Section
  # -----------------------------------------------------------------
  if (loose_route()) {

    if (!has_totag()) {
      sl_send_reply("403", "Forbidden");
      break;
    };

    if (method=="INVITE") {

      if ((method=="INVITE" || method=="REFER") && !has_totag()) {
        if (!proxy_authorize("","subscriber")) {
          proxy_challenge("","0");
          break;
        } else if (!check_from()) {
          sl_send_reply("403", "Use From=ID");
          break;
        };
        consume_credentials();
      };
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
        if (client_nat_test("3")||search("^Route:.*;nat=yes")){
          setflag(6);
          use_media_proxy();
        };
      };

      route(1);
      break;
    };

    # -----------------------------------------------------------
    # Call Type Processing Section
    # -----------------------------------------------------------
    if (!is_uri_host_local()) {
      if (is_from_local() || allow_trusted()) {
        route(4);
        route(1);
      } else {
        sl_send_reply("403", "Forbidden");
      };
      break;
    };

    if (method=="ACK") {
      route(1);
      break;
    } if (method=="CANCEL") {
      route(1);
      break;
    } else if (method=="INVITE") {
      route(3);
      break;
    } else  if (method=="REGISTER") {
      route(2);
      break;
    };

    lookup("aliases");
    if (!is_uri_host_local()) {
      route(4);
      route(1);
      break;
    };

    if (!lookup("location")) {
      sl_send_reply("404", "User Not Found");
      break;
    };

    route(1);
  }

  route[1] {

    # -----------------------------------------------------------
    # Default Message Handler
    # -----------------------------------------------------------
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  t_on_reply("1");

  if (!t_relay()) {

    if (method=="INVITE" || method=="ACK") {
      end_media_session();
    };

    sl_reply_error();
  };
}

route[2] {

  # -----------------------------------------------------------
  # REGISTER Message Handler
  # -----------------------------------------------------------

  sl_send_reply("100", "Trying");

  if (!search("^Contact:[ ]*\*") && client_nat_test("7")) {
    setflag(6);
    fix_nated_register();
    force_rport();
  };

  if (!www_authorize("","subscriber")) {
    www_challenge("","0");
    break;
  };

  if (!check_to()) {
    sl_send_reply("401", "Unauthorized");
    break;
  };

  consume_credentials();

  if (!save("location")) {
    sl_reply_error();
  };
}

route[3] {

  # -----------------------------------------------------------
  # INVITE Message Handler
  # -----------------------------------------------------------

  if (client_nat_test("3")) {
    setflag(7);
    force_rport();
    fix_nated_contact();
  };

  if (!allow_trusted()) {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

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

  consume_credentials();
};

if (uri=~"^sip:1[0-9]{10}@") {
  strip(1);
};

lookup("aliases");
if (!is_uri_host_local()) {
  route(4);
  route(1);
  break;
};

if (uri=~"^sip:011[0-9]*@") {
  route(4);
  route(5);
  break;
};

if (avp_db_load("$ruri/username", "s:callfwd")) { ❷
  setflag(22);❸
  avp_pushto("$ruri", "s:callfwd");❹
  route(6);❺
  break;
};

if (!lookup("location")) {
  if (uri=~"^sip:[0-9]{10}@") {
    route(4);
    route(5);
    break;
  };

  sl_send_reply("404", "User Not Found");
  break;
};

if (avp_db_load("$ruri/username", "s:fwdbusy")) { ❻
  if (!avp_check("s:fwdbusy", "eq/$ruri/i")) { ❼
    setflag(26);
  };
};

if (avp_db_load("$ruri/username", "s:fwdnoanswer")) { ❽
  if (!avp_check("s:fwdnoanswer", "eq/$ruri/i")) { ❾
    setflag(27);
  };
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  t_on_failure("1");10

  route(4);
  route(1);
}

route[4] {

  # -----------------------------------------------------------
  # NAT Traversal Section
  # -----------------------------------------------------------

  if (isflagset(6) || isflagset(7)) {
    if (!isflagset(8)) { 11
      setflag(8);
      use_media_proxy();
    };
  };
}

route[5] {

  # -----------------------------------------------------------
  # PSTN Handler
  # -----------------------------------------------------------

  rewritehost("192.0.2.245"); # INSERT YOUR PSTN GATEWAY IP ADDRESS

  avp_write("i:45", "inv_timeout");

  t_on_failure("1");12

  route(1);
}

13route[6] {

  # -----------------------------------------------------------
  # Call Forwarding Handler
  #
  # This must be done as a route block because sl_send_reply() cannot be
  # called from the failure_route block
  # -----------------------------------------------------------

  if (uri=~"^sip:1[0-9]{10}@") { 14
    strip(1);
  };

  lookup("aliases");15
  if (!is_uri_host_local()) { 16

    if (!isflagset(22)) {
      append_branch();
    };

    route(4);
    route(1);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    break;
  };

  if (uri=~"^sip:011[0-9]*@") {
    route(4); 17
    route(5);
    break;
  };

  if (!lookup("location")) { 18

    if (uri=~"^sip:[0-9]{10}@") {
      route(4);
      route(1);
      break;
    };

    sl_send_reply("404", "User Not Found");
  };

  route(4);
  route(1);
}

onreply_route[1] {

  if ((isflagset(6) || isflagset(7)) &&
      (status=~"(180)|(183)|2[0-9][0-9]")) {

    if (!search("^Content-Length:[ ]*0")) {
      use_media_proxy();
    };
  };

  if (client_nat_test("1")) {
    fix_nated_contact();
  };
}

19failure_route[1] {

  if (t_check_status("487")) { 20
    break;
  };

  if (isflagset(26) && t_check_status("486")) { 21
    if (avp_pushto("$ruri", "s:fwdbusy")) {22
      avp_delete("s:fwdbusy");23
      resetflag(26);
      route(6);24
      break;
    };
  };

  if (isflagset(27) && t_check_status("408")) { 25
    if (avp_pushto("$ruri", "s:fwdnoanswer")) { 26
      avp_delete("s:fwdnoanswer");27
      resetflag(27);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
        route(6); 28
        break;
      };
    };

    end_media_session();29
}
```

Call Forwarding ser.cfg Analysis

1. Call forwarding is dependent on the avpops module. This module needs to access the MySQL database in order to read a subscribers call forwarding preferences from the usr_preferences table. So, here we specify the MySQL database and table for call forwarding preferences.
2. Here is where we implement the blind call forwarding functionality. When processing an INVITE message we need to lookup any row in the MySQL usr_preferences table to see if we can find a record that has the name portion of the R-URI and an attribute of callfwd. If the avp_db_load() function returns TRUE then the AVP named s:callfwd will have the blind call forwarding destination set.
3. If we are about to serial fork the call because of blind call forwarding, then we set flag 22 for future reference in route[6]. This is important because this flag determines whether or not we need to call the append_branch() function which actually forks the call. A subtle item to understand here is that blind call forwarding can only happen when processing the original INVITE. This means that the destination set of the call has not been processed yet, so we can safely alter the R-URI and the call to append_branch() should not be done.

   Forward on busy and forward no answer however, only occur after the original INVITE has failed, which means the destination set has been processed. In order to fork the call at this point the append_branch() function must be called.Therefore, flag 22 will be set only for blind call forwarding.
4. Since avp_db_load() found a blind call forwarding record in MySQL we need to write that new destination to the R-URI in order to send the INVITE message to the correct destination. Avp_pushto() does just this.

   This statement copies the value in AVP s:callfwd to the R-URI of the actual message.
5. Send the processing to route(6) which handles all call forwarding.
6. This section loads the subscribers preferences for forward on busy and forward no answer. Load the forward on busy setting from the MySQL usr_preferences table. Here we look for records in the database table where the attribute column value is fwdbusy and the username part (before @) of the R-URI can be found in the tables username column. avp_db_load() returns true if one or more records were found.
7. We do not allow users to forward to their now SIP phone, which would cause looping problems for SER. So we use avp_check() to make sure the forward on busy destination is not the same as the R-URI destination.

   If forward on busy is set then we need to remember this for future processing in the failure_route of this configuration script. If the subscriber in the R-URI has forward on busy enabled, then flag 26 will be set, and we will be able to detect this in the failure_route.
8. Load the forward no answer setting from the MySQL usr_preferences table. Here we look for records where the username port of the R-URI has an attribute column value of fwdnoanswer.
9. We do not allow users to forward to their own SIP phone, which would cause looping problems for SER. So we use avp_check() to make sure the forward no answer destination is not the same as the R-URI destination.

   If forward on busy is set then we need to remember this for future processing in the failure_route of this configuration script. If the subscriber in the R-URI has forward no answer enable then flag 27 will be set, and we will be able to detect this in the failure_route.
10. t_on_failure() informs SER that we want to perform special handling when a failure condition occurs. Failure conditions in this context refer to 4xx and 5xx response codes. For example, 486 is the response code for a busy answer.

    By setting t_on_failure(1) before calling t_relay(), SER will pass control to the code block defined as failure_route[1] which appears at the end of the configuration script.
11. We must take special precautions not to call use_media_proxy() more than once because by doing so the c= field in the SDP will get corrupted. This can happen if the original INVITE was NATed in route(4), get called during the call set up, and a 486 Busy is received. This will trigger the forward on busy functionality. Once

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

the call is forked, use_media_proxy() must not be called because it was already called during the original INVITE processing logic.

12 Enable the failure route for all PSTN calls.

13 Here we introduce the call forwarding handler. This handler will create the new call leg for all three types of call forwarding.

14 In North America we dial 1+ ten digits to call a long distance number. If the R-URI contains this prefix we strip it off before processing.

15 Find any aliases. The call forwarding number in the MySQL usr_preferences table could very well be an alias in our database.

16 If the R-URI is not served by our domain then call append_branch() for forward on busy and forward no answer before enabling the NAT processing code in route(4) and relaying the cal from route(1).

17 If the R-URI is an international PSTN destination then call the NAT processing code in route(4) and relay to the PSTN via route(5).

18 Find the R-URI in the user location cache. If a user is not found, then determine if it is a domestic PSTN destination. If so, enable the NAT processing logic in route(4) and relay the call to the PSTN via route(5).

Enable NAT traversal code and relay the message. Messages hitting this line are INVITE messages being forwarded to another SIP phone that is registered on this SIP router.

19 Here we introduce the failure route. If we call t_on_failure() before t_relay(), then SER will execute this code block when a 4xx or 5xx reply message is received.

20 If we entered the failure route because the caller cancelled the call (ie, response code 487), then we simply stop processing.

21 If forward on busy (flag 26) is set and the reply message is a 486, then enter the forward on busy code block.

22 Here we retrieve the previously loaded forward on busy destination URI. The AVP named s:fwdbusy is copied to the R-URI.

23 Since we've now copied the forward on busy URI to the R-URI we can safely delete the AVP. We could omit this line altogether and SER would free the AVP when the transaction is freed.

We now clear the forward on busy flag to prevent accidentally entering this code block on future 486 messages.

24 Here we pass control to the call forwarding route block and exit.

25 If forward no answer (flag 27) is set and the reply message is a 408 then enter the forward no answer code block.

26 Here we retrieve the previously loaded forward no answer destination URI. The AVP named s:fwdnoanswer is copied to the R-URI.

27 Since we've now copied the forward no answer URI to the R-URI we can safely delete the AVP. We could omit this line altogether and SER would free the AVP when the transaction is freed.

We now clear the forward no answer flag to prevent accidentally entering this code block on future 408 messages.

28 Here we pass control to the call forwarding route block and exit.

29 Disable mediaproxy if it was enabled during the call set up.

Using the Call Forwarding ser.cfg Example

In order to use the new call forwarding SER configuration you need to populate the usr_preferences table in MySQL. Below are a few example rows which show each of the call forwarding types.

```
Example usr_preferences Call Forwarding Records
+------------+-------------+--------------------------------+
| username   | attribute   | value                          |
+------------+-------------+--------------------------------+
| 5025552001 | callfwd     | sip:5025554706@sip.example.com |
| 9145551451 | fwdbusy     | sip:9145550257@sip.example.com |
| 5615553320 | fwdnoanswer | sip:9545553040@sip.example.com |
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 11. Appendix - How to download and configure the latest version of SER

This appendix will show you how to download all the relevant source code, compile and install all the binaries to make a working system. You will also be shown how to modify the source to support MySQL.

Downloading the Latest SER Source Code

As an alternative, you can get the source from the CVS. The source code to SER is kept under CVS control on the website cvs.berlios.de.

To load the source you will either need the CVS package or you will have to browse the ftp.berlios.de [ftp://ftp.berlios.de/] website to find the source code.

Most linux distributions have cvs loaded, in which case you can login to your usual account and enter the following command :-

cd src

export CVSROOT=:pserver:anonymous@cvs.berlios.de:/cvsroot/ser

cvs login

[you will be prompted for a password, please just enter RETURN]

cvs co -r rel_0_9_0 sip_router

This will create a directory sip_router in the current directory with all the source code, but not RTPproxy. To get RPTproxy, you run the command cvs co -r rel_0_9_0 rtpproxy

Making the SER Binaries and installing

Once the source code has been loaded (either via cvs or by extraction from a zip file) we are now ready to compile and install the binaries. The default makefiles does not compile a number of module (including the MySQL module) but for the first hello world scenario we dont need it.

To make the code enter the following commands :-

cd sip_router

make all

All the modules will be compiled and ready for installation. To perform this task you need root privileges, so enter the following command :-

su

[Enter your root password]

make install

This will install the binaries by default in the directory /usr/local/etc/ser

Your installation is now ready to be executed. However the next sub-section describes how to make your sip server start whenever your machine is rebooted.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

To check that everything is OK, enter the ser -V and you should get the following output :-

```
version: ser 0.9.3 (i386/linux)
flags: STATS: Off, USE_IPV6, USE_TCP, DISABLE_NAGLE, USE_MCAST, DNS_IP_HACK, SHM_MEM, S
ADAPTIVE_WAIT_LOOPS=1024, MAX_RECV_BUFFER_SIZE 262144, MAX_LISTEN 16, MAX_URI_SIZE 1024
@(#) $Id: main.c,v 1.197 2004/12/03 19:09:31 andrei Exp $
main.c compiled on 18:17:55 Mar 6 2005 with gcc 3.3
```

You will now have a working system. The binaries are loaded into the directory /usr/local/sbin, and the configuration files (including ser.cfg) loaded into the directory /usr/local/etc/ser.

If you take the hello world ser.cfg file and replace the one in the configuration directory you can start the sip server by having root privileges and entering the command :-

ser

To check everything is OK, run the ps command to see if there are a number of ser processes running.

You now have a working system.

Installing MediaProxy

You can find the mediaproxy source package used for these configuration files at http:/ONsip.org/. Look for the MediaProxy Getting Started package.

You can get the latest version from http://mediaproxy.ag-projects.com/.

The package has thorough INSTALL and README files. PLEASE NOTE that the configuration files in this document does not use proxydispatcher.py. That is, the mediaproxy SER module calls the mediaproxy.py proxy daemon directly. If you want to install mediaproxy on another server, you will have to start proxydispatcher.py on the server running SER and mediaproxy.py on the dedicated RTP proxy server. Change the following line in ser.cfg:

modparam("mediaproxy","mediaproxy_socket", "/var/run/mediaproxy.sock")

to

modparam("mediaproxy","mediaproxy_socket", "/var/run/proxydispatcher.sock")

Please refer to the INSTALL and README files if you want to run multiple mediaproxy RTP proxy servers.

Configuring the system

Generally, the system has to be configured so that the installed programmes are executed at start-up. Normally, this is completed by providing files in /etc/init.d. We leave it to the user to configure the services of their own system; however we have provided example scripts to start the SER and mediaproxy processes. Please note, if you are not supporting RTP proxying, then you do not need to run the mediaproxy process.

To install the scripts, please create the file(s) in the /etc/init.d directory and then run the commands :-

chkconfig -add ser

chkconfig -add mediaproxy

Init.d/ser

```
#!/bin/bash
#
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
# Startup script for SER
#
# chkconfig: 345 91 15
# description: Ser is a fast SIP Proxy.
#
# processname: ser
# pidfile: /var/run/ser.pid
# config: /etc/ser/ser.cfg

# Source function library.
. /etc/rc.d/init.d/functions

ser=/usr/local/sbin/ser
prog=ser
OPTIONS="-d -d -d -d -d -d -d -d -d"
RETVAL=0

start() {
        echo -n $"Starting $prog: "
        daemon $ser $OPTIONS
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && touch /var/lock/subsys/ser
        return $RETVAL
}

stop() {
  echo -n $"Stopping $prog: "
  killproc $ser
  RETVAL=$?
  echo
  [ $RETVAL = 0 ] && rm -f /var/lock/subsys/ser /var/run/ser.pid
}

reload() {
  echo -n $"Reloading $prog: "
  killproc $ser -HUP
  RETVAL=$?
  echo
}

# See how we were called.
case "$1" in
  start)
  start
  ;;
  stop)
  stop
  ;;
  status)
        status $ser
  RETVAL=$?
  ;;
  restart)
  stop
  sleep 3
  start
  ;;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  condrestart)
  if [ -f /var/run/ser.pid ] ; then
    stop
    start
  fi
  ;;
  *)
  echo $"Usage: $prog {start|stop|restart|condrestart|status|help}"
  exit 1
esac

exit $RETVAL
```

Init.d/mediaproxy

The mediaproxy programme is used to support NAT configurations.

```
#!/bin/sh
#
# chkconfig: 2345 90 20
# description: VoIP RTP Proxy Server
#
# processname: mediaproxy
# pidfile: /var/run/mediaproxy.pid

# source function library
. /etc/rc.d/init.d/functions


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

INSTALL_DIR="/usr/local"
RUNTIME_DIR="/var/run"

PROXY="$INSTALL_DIR/mediaproxy/mediaproxy.py"
DISPATCHER="$INSTALL_DIR/mediaproxy/proxydispatcher.py"
PROXY_PID="$RUNTIME_DIR/mediaproxy.pid"
DISPATCHER_PID="$RUNTIME_DIR/proxydispatcher.pid"

# Options for mediaproxy and dispatcher. Do not include --pid <pidfile>
# --pid <pidfile> will be added automatically if needed.
PROXY_OPTIONS="--ip=10.3.0.221 --listen=127.0.0.1"
DISPATCHER_OPTIONS="domain://sip.dialez.com"

NAME="mediaproxy"
DESC="SER MediaProxy server"

test -f $PROXY      || exit 0
test -f $DISPATCHER || exit 0

if [ "$PROXY_PID" != "/var/run/mediaproxy.pid" ]; then
    PROXY_OPTIONS="--pid $PROXY_PID $PROXY_OPTIONS"
fi
if [ "$DISPATCHER_PID" != "/var/run/proxydispatcher.pid" ]; then
    DISPATCHER_OPTIONS="--pid $DISPATCHER_PID $DISPATCHER_OPTIONS"
fi
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
start() {
    echo -n "Starting $DESC: $NAME"
    $PROXY $PROXY_OPTIONS
    $DISPATCHER $DISPATCHER_OPTIONS
    echo "."
}

stop () {
    echo -n "Stopping $DESC: $NAME"
    kill `cat $PROXY_PID`
    kill `cat $DISPATCHER_PID`
    echo "."
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart|force-reload)
        stop
        #sleep 1
        start
        ;;
    *)
        echo "Usage: /etc/init.d/$NAME {start|stop|restart|force-reload}" >&2
        exit 1
        ;;
esac

exit 0
```

Supporting MySQL

A database is needed to support a number of functions within ser. Namely authentication and accounting. In our configuration we have used MySQL, other databases can be supported but will require a different setup.

We are assuming that you have a working MySQL database with a root password. Of course, you will also need the development MySQL package loaded as it contains the header files for the source code.

To support MySQL, three steps are needed :-

1. Modify the source, rebuild and reinstall the binaries
2. Modify MySQL to support the database and tables needed
3. Modify the ser.cfg to use the database

All the source has already been loaded, what we need to change is the Makefile so that we now include the MySQL module. Edit the file src/sip_router/Makefile and at the line starting exclude_modules= remove the reference to mysql. Save the file and then recompile the code with the command make all. Reinstall the binaries by taking root privileges and executing make install.

MySQL needs to be modified to support the SER database and tables. Again take on root privileges and execute make dbinstall. You will be asked for the root MySQL password and also the realm that you want. This is configurable, but normally you use the IP address of the sip server.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Now replace the ser.cfg file with the one that supports MySQL and restart the sip server.

Debugging Tips

There a number of techniques available to debug SER, the main types fall into two categories :-

1. Capture the SIP messages
2. Generate debug information

These types will now be described.

Capture of SIP Messages

A very useful technique is to look at the actual SIP messages that are produced by SER as seen by other clients on the network. Two tools are commonly used, ethereal (or the text based equivalent tethereal) and ngrep. Assuming that you have root access to the sip server, then the syntax for running ethereal is :-

ethereal port 5060

This will display all packets being sent and received from port 5060 which is the common port used by sip. Added -w file.cap will store the data for future viewing.

Generate Debug Information

SER has the ability to generate a vast amount of information that can be used for debugging. To capture this data, either the stderr file can be redirected, or the program run in foreground. This later technique is achieved by modifying the line fork=no in ser.cfg.

Once the data is being displayed /captured, then the level of detail can be varied by modifying the line debug=4 in ser.cfg higher the number more detail is generated.

Finally the user can add more information by inserting into ser.cfg the command xlog at appropriate points then debug information is required. Please refer to the xlog module README for details.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Chapter 12. Appendix The Call Processing Language (CPL)

The Call Processing Language (CPL) is a language designed to describe and control Internet telephony services and provide call processing functionality. It is designed for end user service creation and is purposely limited its capabilities. It works on top of SIP or H323 and is a safe language for non-experienced users as it can only access limited resources, cannot call external programs and does not provide loops or recursion. To highlight the applicability of this language, suggestions are given below regarding possible uses of CPL scripts, including call routing, screening and logging services:

1. Call Forward on Busy/No Answer
2. Call Screening and Rule Based Call Processing
3. Intelligent User Location
4. Administrator Service Definition

[Enables administrators to devise policies for site wide telephony use]

1. Web Middleware
2. Sequential Call Forking (Note the SER LCR and AVPOPS module can also be utilised)

Ser.cfg

Before CPL scripts can become operational on the SER server, the ser.cfg must be modified to interpret the scripts correctly.

The cpl-c module must first be loaded by ser:

loadmodule "/opt/ser/lib/ser/modules/cpl-c.so"

Next the modules must be configured:

modparam("cpl-c", "cpl_db", "mysql://root:password@localhost/ser")

In this case "username" and "password" represent the username and password for the mysql database named ser. The entry at "localhost" should match with the server name on which the mysql database is running. ser and heslo are the default username and password.

modparam("cpl-c", "cpl_table", "cpl")

This refers to the "cpl" table which is the default table for the cpl-scripts in the database.

modparam("cpl-c", "cpl_dtd_file", "/tmp/ser-0.9.0/modules/cpl-c/cpl-06.dtd")

Pointers to the location of the CPL XML DTD file must be given (the XML DTD is necessary for parsing CPL scripts and is described in further detail later in this section).

modparam("cpl-c", "lookup_domain", "location")

This parameter should be set to "location" to let the lookup-node work correctly.

All the above parameters are mandatory. Two more parameters exist which are optional: A debugging parameter pointing to the existence of a log file and a parameter that specifies the maximum of recursive executions in CPL.

If an invite is incoming and a cpl script exists for the recipient, the default call processing is forfeited in favour of the CPL script logic.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
#--------------------Call Type Processing Section---------------------#
  # if the request is for other domain use UsrLoc
  # (in case, it does not work, use the following command
  # with proper names and addresses in it)
  if (uri==myself) {

    if (method == "INVITE"){
      if(!cpl_run_script("incoming", "is_stateless"))
      {
        # script execution failed
        t_reply("500", "CPL script execution failed");
      };
      route(3);
      break;
    } else if (method == "REGISTER"){
      #handle REGISTER messages with CPL script
      cpl_process_register();
      route(2);
      break;
    };
```

Once the ser.cfg has been modified and SER has been restarted, issue the pstree command and two child processes should be observed:

e.g

|---rtpproxy

|---ser-+-25&[ser]

---2*[ser---ser]

Uploading a CPL Script

A CPL script can be uploaded onto SER using either the SIP REGISTER message or manually via SERs FIFO facility.

SIP REGISTER Message

A SIP REGISTER message (the message used by SIP to register a user on the network) contains the CPL script in the body/payload. When the REGISTER message reaches SER, the CPL script is retrieved from the payload and stored in the cpl table of the ser database. Thus, the CPL script resides in the SER database and will be executed when a call arrives addressed to that user. Then the script will drive the action to be taken with that call according to the requirements specified by the user in the script

CPLED, a free graphical tool, can be used for the above purpose. It includes a script transport feature whereby scripts can be downloaded, uploaded or removed via http or the SIP REGISTER method (authentication supported). For more information refer to http://www.iptel.org/products/cpled/.

Serctl FIFO Interface

To use the FIFO interface, a command similar to the following can be used:

serctl fifo LOAD_CPL user@domain /path/to/cpl/script

e.g.

```
serctl fifo LOAD_CPL 2000@server /opt/ser/etc/ser/cplscript.cpl.xml
```

Removing a CPL Script

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

If a user wishes to remove a particular CPL script, this can also be done using SERs FIFO facility.

```
serctl fifo REMOVE_CPL user@domain
```

What, Where, Which & How?

What does a CPL script do? - A CPL script runs in a signaling server (not protocol specific, can be SIP or H323) and controls that systems proxy, redirect or rejection actions for the set-up of a particular call. More specifically it replaces the user location functionality of a signaling server. It takes the registration information, the specifics of the call request and other external information and chooses the signaling actions to perform.

Where are CPL scripts located? - Users can have CPL scripts on any network server which their call establishment requests pass through and with which they have a trust relationship. CPL scripts can reside on SIP/H323 servers, application servers or intelligent agents.

Which party generates CPL scripts? - CPL scripts are extremely generic in their ability to be adapted to the requirements of all parties involved in a call transaction. In the most direct approach, end users can utilise CPL for describing their services. Third parties can also utilise CPL to create and/or customize services for clients, running on either servers owned by the user or the users service provider. Service administrators can also use CPL to define server service policies and it can also act as a back end for a web interface whereby service creation and customization is provided transparently to the user.

CPL scripts are usually associated with a particular Internet telephony address. Each SER user can only have one CPL script. If a new script is loaded, the previous one is overwritten. If different services are required by the same user e.g. time based routing, call screening, forward on busy etc, then these must be mixed in the same script during provisioning.

How? - Methods for CPL Script Generation

Manually

CPL scripts can be easily created by hand due to its uncomplicated syntax and similarity with HTML. Examples of such scripts are included at the end of this document.

Automated

As previously described, web middleware could be utilised to transparently provision the CPL syntax.

Graphical Tools

Graphical User Interface (GUI) tools are also available for provisioning CPL scripts. These provide inexperienced users with powerful access to CPLs functionality through a simple interface.

An example of this is CPLED, a free graphical CPL editor which was mentioned earlier in the section. It is written in JAVA and can be used as a standalone application or JAVA applet.

The XML DTD for CPL

Syntactically, CPL scripts are represented by XML documents. This is advantageous in that it makes parsing easy and parsing tools are publicly available. It is simple to understand and like HTML consists of a hierarchical structure of tags. A complete Document Type Declaration (DTD) describing the XML syntax of the CPL can be found in the /ser-0.9.0/modules/cpl-c directory and also should be called from the ser.cfg as shown earlier. It should be consulted if syntax errors are experienced uploading CPL scripts and all CPL scripts should comply with this document. The XML DTD is also provided in Appendix X.

CPL Script Examples

Call Screening a Particular Contact

This script demonstrates CPLs call screening abilities and rejects all calls received from extension 2000.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
<cpl>
  <incoming>
    <address-switch field="origin" subfield="user>
     <address is=2000>
       <reject status=reject reason=I don't take calls from extension 2000 />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Time-Based Switch

This script illustrates a time-based CPL script. Incoming calls received between Monday Friday and 9 a.m to 5 p.m. are proxied as normal. However if a call presents at the server outside those hours, it is directed to the users voicemail.

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
<cpl>
  <time-switch>
    <time dtstart=9 dtend=5 wkst=MO|TU|WE|TH|FR>
      <lookup source=registration>
        <success>
          <proxy />
        </success>
      </lookup>
    </time>
    <otherwise>
      <location url=sip:2000@voicemail.server.com>
        <proxy />
      </location>
    </otherwise>
  </time-switch>
</cpl>
```

Forward on No Answer or Busy

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
<cpl>
  <subaction id=voicemail>
  <location url=sip:2000@voicemail.server.com>
    <proxy />
  </location>
  </subaction>

  <incoming>
    <location url=sip:2000@pc.server.com>
      <proxy timeout=8>
        <busy>
          <sub ref=voicemail/>
        </busy>
        <noanswer>
          <sub ref=voicemail/>
        </noanswer>
      </proxy>
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    </location>
  </incoming>
</cpl>
```

The above script could perhaps be modified for serial forking along the lines of the following:

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
<cpl>
  <subaction id=pda>
    <location url=sip:2000@pda.server.com>
      <proxy />
    </location>
  </subaction>

  <incoming>
    <location url=sip:2000@pc.server.com>
      <proxy timeout=8>
        <noanswer>
          <sub ref=pda/>
        </noanswer>
      </proxy>
    </location>
  </incoming>
</cpl>
```

Outgoing Call Screening [*]

This script illustrates how to filter outgoing calls. This script blocks an outgoing calls to premium rate numbers.

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
<cpl>
  <subaction id=pda>
    <location url=sip:2000@pda.server.com>
      <proxy />
    </location>
  </subaction>

  <incoming>
    <location url=sip:2000@pc.server.com>
      <proxy timeout=8>
        <noanswer>
          <sub ref=pda/>
        </noanswer>
      </proxy>
    </location>
  </incoming>
</cpl>
```

Priority and Language Routing [*]

The following example illustrates a service based a calls priority value and language settings. If the call request has a priority of urgent or higher, the default script behavior is performed. Otherwise the language field is checked for the language (Spanish) and if it is present the call is proxied to a Spanish-speaking operator, otherwise to an English-speaking operator.

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
<cpl>
  <incoming>
    <priority-switch>
      <priority greater=urgent />
      <otherwise>
        <language-switch>
          <language matches=es>
            <location url=sip:Spanish@operator.server.com>
              < proxy />
            </location>
          </language>
          <location url=sip:English@operator.server.com>
            < proxy />
          </location>
        </language-switch>
      </otherwise>
    </priority-switch>
  </incoming>
</cpl>
```

A More Complex Example [not verified]

In this scenario a user wants all calls directed to the terminal at his desk. If he does not answer after a certain period of time, calls from this boss are forwarded to his pda and all others are directed to his voicemail.

```
<?xml version=1.0 ?>
<!DOCTYPE cpl SYSTEM cpl.dtd>
<cpl>
  <location url=sip:2000@deskphone.server.com>
    <proxy timeout=8s>
      <busy>
        <location url=sip:2000@voicemail.server.com merge=clear>
          <redirect />
        </location>
      </busy>
      <noanswer>
        <string-switch field=from>
          <string matches=boss@*server.com>
            <location url=sip:2000@pda.server.com merge=clear>
              <proxy />
            </location>
          </string>
          <otherwise>
            <location url=sip:2000@voicemail.server.com merge=clear>
              <redirect />
          </otherwise>
        </string-switch>
      </noanswer>
    </proxy>
  </location>
</cpl>
```

References

The following documents have been referenced throughout this document and can be used as a source of further information regarding CPL.

http://www.faqs.org/rfcs/rfc2824.html

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

http://www1.cs.columbia.edu/~lennox/draft-ietf-iptel-cpl-00.pdf

http://www1.cs.columbia.edu/~lennox/thesis.pdf

http://lotos.site.uottawa.ca/ftp/pub/Lotos/Theses/dj_msc.pdf

http://www.denic.de/media/pdf/enum/veranstaltungen/Goertz.pdf

http://mia.ece.uic.edu/~papers/WWW/MultimediaStandards/cpl_xml.pdf

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/