iptel.org SIP Express Router v0.11.0 -- Admin's Guide

Jiri Kuthan

Jan Janak

Yacine Rebahi

iptel.org SIP Express Router v0.11.0 -- Admin's Guide

by Jiri Kuthan, Jan Janak, and Yacine Rebahi

The document describes the SIP Express Router and its use in SIP networks. It is intended as an aid to server administrators.

Table of Contents

List of Tables

List of Examples

Chapter 1. General Information

1.1. About SIP Express Router (SER)

SIP Express Router (SER) is an industrial-strength, free VoIP server based on the Session Initiation Protocol (SIP, RFC3261). It is engineered to power IP telephony infrastructures up to large scale. The server keeps track of users, sets up VoIP sessions, relays instant messages and creates space for new plug-in applications. Its proven interoperability guarantees seamless integration with components from other vendors, eliminating the risk of a single-vendor trap. It has successfully participated in various interoperability tests in which it worked with the products of other leading SIP vendors.

The SIP Express Router enables a flexible plug-in model for new applications: Third parties can easily link their plug-ins with the server code and provide thereby advanced and customized services. In this way, plug-ins such as RADIUS accounting, SMS gateway, ENUM queries, or presence agent have already been developed and are provided as advanced features. Other modules are underway: firewall control, postgress and LDAP database drivers and more.

Its performance and robustness allows it to serve millions of users and accommodate needs of very large operators. With a $3000 dual-CPU PC, the SIP Express Router is able to power IP telephony services in an area as large as the Bay Area during peak hours. Even on an IPAQ PDA, the server withstands 150 calls per second (CPS)! The server has been powering our iptel.org free SIP site withstanding heavy daily load that is further increasing with the popularity of Microsoft's Windows Messenger.

The SIP Express Router is extremely configurable to allow the creation of various routing and admission policies as well as setting up new and customized services. Its configurability allows it to serve many roles: network security barrier, application server, or PSTN gateway guard for example.

ser can be also used with contributed applications. Currently, serweb, a ser web interface, SIPSak diagnostic tool and SEMS media server are available. Visit our site, http://www.iptel.org/, for more information on contributed packages.

1.2. About iptel.org

iptel.org is a know-how organization spun off from Germany's national research company FhG Fokus. One of the first SIP implementations ever, low-QoS enhancements, interoperability tests and VoIP-capable firewall control concepts are examples of well-known FhG's work.

iptel.org continues to keep this know-how leadership in SIP. The access rate of the company's site, a well-known source of technological information, is a best proof of interest. Thousands of hits come every day from the whole Internet.

The iptel.org site, powered by SER, offers SIP services on the public Internet. Feel free to apply for a free SIP account at http://www.iptel.org/user/ (http://www.iptel.org/user/)

1.3. Feature List

Based on the latest standards, the SIP Express Router (SER) includes support for registrar, proxy and redirect mode. Further it acts as an application server with support for instant messaging and presence including a 2G/SMS and Jabber gateway, a call control policy language, call number translation, private dial plans and accounting, ENUM, authorization and authentication (AAA) services. SER runs on Sun/Solaris, PC/Linux, PC/BSD, IPAQ/Linux platforms and supports both IPv4 and IPv6. Hosting multiple domains and database redundancy is supported.

ser has been carefully engineered with the following design objectives in mind:

- *Speed* - With ser, thousands of calls per seconds are achievable even on low-cost platforms. This competitive capacity allows setting up networks which are inexpensive and easy to manage due to low number of devices required. The processing capacity makes dealing with many stress factors easier. The stress factors may include but are not limited to broken configurations and implementations, boot avalanches on power-up, high-traffic applications such as presence, redundancy replications and denial-of-service attacks.

  The speed has been achieved by extensive code optimization, use of customized code, ANSI C combined with assembly instructions and leveraging latest SIP improvements. When powered by a dual-CPU Linux PC, ser is able to process thousands of calls per second, capacity needed to serve call signaling demands of Bay Area population.

- *Flexibility* - SER allows its users to define its behavior. Administrators may write textual scripts which determine SIP routing decisions, the main job of a proxy server. They may use the script to configure numerous parameters and introduce additional logic. For example, the scripts can determine for which destinations record routing should be performed, who will be authenticated, which transactions should be processed statefully, which requests will be proxied or redirected, etc.
- *Extensibility* - SER's extensibility allows linking of new C code to ser to redefine or extend its logic. The new code can be developed independently on SER core and linked to it in run-time. The concept is similar to the module concept known for example in Apache Web server. Even such essential parts such as transaction management have been developed as modules to keep the SER core compact and fast.
- *Portability.* ser has been written in ANSI C. It has been extensively tested on PC/Linux and Sun/Solaris. Ports to BSD and IPAQ/Linux exist.
- *Interoperability.* ser is based on the open SIP standard. It has undergone extensive tests with products of other vendors both in iptel.org labs and in the SIP Interoperability Tests (SIPIT). ser powers the public iptel.org site 24 hours a day, 356 days a year serving numerous SIP implementations using this site.
- *Small size.* Footprint of the core is 300k, add-on modules take up to 630k.

1.4. Use Cases

This section illustrates the most frequent uses of SIP. In all these scenarios, the SIP Express Router (SER) can be easily deployed as the glue connecting all SIP components together, be it soft-phones, hard-phones, PSTN gateways or any other SIP-compliant devices.

1.4.1. Added-Value ISP Services

To attract customers, ISPs frequently offer applications bundled with IP access. With SIP, the providers can conveniently offer a variety of services running on top of a single infrastructure. Particularly, deploying VoIP and instant messaging and presence services is as easy as setting up a SIP server and guiding customers to use Windows Messenger. Additionally, the ISPs may offer advanced services such as PSTN termination, user-driven call handling or unified messaging all using the same infrastructure.

SIP Express Router has been engineered to power large scale networks: its capacity can deal with large number of customers under high load caused by modern applications. Premium performance allows deploying a low number of boxes while keeping investments and operational expenses extremely low. ISPs can offer SIP-based instant messaging services and interface them to other instant messaging systems (Jabber, SMS). VoIP can be easily integrated along with added-value services, such as voicemail.

1.4.2. PC2Phone

Internet Telephony Service Providers (ITSPs) offer the service of interconnecting Internet telephony users using PC softphone or appliances to PSTN. Particularly with long-distance and international calls, competitive pricing can be achieved by routing the calls over the Internet.

SIP Express Router can be easily configured to serve pc2phone users, distribute calls to geographically appropriate PSTN gateway, act as a security barrier and keep track of charging.

1.4.3. PBX Replacement

Replacing a traditional PBX in an enterprise can achieve reasonable savings. Enterprises can deploy a single infrastructure for both voice and data and bridge distant locations over the Internet. Additionally, they can benefit of integration of voice and data.

The SIP Express Router scales from SOHOs to large, international enterprises. Even a single installation on a common PC is able to serve VoIP signaling of any world's enterprise. Its policy-based routing language makes implementation of numbering plans of companies spread across the world very easy. ACL features allow for protection of PSTN gateway from unauthorized callers.

SIP Express Router's support for programmable routing and accounting efficiently allows for implementation of such a scenario.

1.5. About SIP Technology

The SIP protocol family is the technology which integrates services. With SIP, Internet users can easily contact each other; figure out willingness to have a conversation and couple different applications such as VoIP, video and instant messaging. Integration with added-value services is seamless and easy. Examples include integration with web (click-to-dial), E-mail (voice2email, UMS), and PSTN-like services (conditional forwarding).

The core piece of the technology is the Session Initiation Protocol (SIP, RFC3261) standardized by IETF. Its main function is to establish communication sessions between users connected to the public Internet and identified by e-mail-like addresses. One of SIP's greatest features is its transparent support for multiple applications: the same infrastructure may be used for voice, video, gaming or instant messaging as well as any other communication application.

There are numerous scenarios in which SIP is already deployed: PBX replacement allows for deployment of single inexpensive infrastructure in enterprises; PC-2-phone long-distance services (e.g., Deltathree) cut callers long-distance expenses; instant messaging offered by public severs (e.g., iptel.org) combines voice and text services with presence information. New deployment scenarios are underway: SIP is a part of UMTS networks and research publications suggest the use of SIP for virtual home environments or distributed network games.

1.6. Known SER Limitations

The following items are not part of current distribution and are planned for next releases:

- Script processing of multiple branches on forking

Warning

ser's request processing language allows to make request decisions based on current URI. When a request if forked to multiple destinations, only the first branch's URI is used as input for script processing. This might lead to unexpected results. Whenever a URI resolves to multiple different next-hop URIs, only the first is processed which may result in handling not appropriate for the other branch. For example, a URI might resolve to an IP phone SIP address and PSTN gateway SIP address. If the IP phone address is the first, then script execution ignores the second branch. If a script includes checking gateway address in request URI, the checks never match. That might result in ignoring of gateway admission control rules or applying them unnecessarily to non-gateway destinations.

List of known problems is publicly available at the ser webpage at http://www.iptel.org/ser/ (http://www.iptel.org/ser/). See the "ISSUES" link.

1.7. Licensing

ser is freely available under terms and conditions of the GNU General Public License.

```
----------------------------------------------------------------------
IMPORTANT NOTES

1) The GPL applies to this copy of SIP Express Router software (ser).
   For a license to use the ser software under conditions
   other than those described here, or to purchase support for this
   software, please contact iptel.org by e-mail at the following addresses:
```

```
    info@iptel.org

   (see http://www.gnu.org/copyleft/gpl-faq.html#TOCHeardOtherLicense
    for an explanation how parallel licenses comply with GPL)

2) ser software allows programmers to plug-in external modules to the
   core part. Note that GPL mandates all plug-ins developed for the
   ser software released under GPL license to be GPL-ed as well.

   (see http://www.gnu.org/copyleft/gpl-faq.html#GPLAndPlugins
    for a detailed explanation)

3) Note that the GPL bellow is copyrighted by the Free Software Foundation,
   but the ser software is copyrighted by FhG

-------------------------------------------------------------------------

GNU Licence FAQ

This FAQ provides answers to most frequently asked questions. To fully
understand implications of the GNU license, read it.

- you can run SER for any purpose
- you can redistribute it as long as you include source code and
  license conditions with the distribution
- you cannot release programs derived from SER without releasing
  their source code


-------------------------------------------------------------------------

    GNU GENERAL PUBLIC LICENSE
       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

    Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
```

```
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
```

```
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
```

```
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.
```

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes

```
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

    NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

     END OF TERMS AND CONDITIONS
```

1.8. Obtaining Technical Assistance

iptel.org offers qualified professional services. We help you to plan your network, configure your server, build applications, integrate SIP components with each other, and set up advanced features such as redundancy, multidomain support, CLID interworking and others not described in this document. Our customer alert services notifies you on all new features and code fixes. We help you to solve operational troubles in short time and keep you updated on latest operational practices. Ask info@iptel.org for information on enrollment in our support program.

Additionaly, help may be obtained from our user forum. The community of SER users is subscribed to the serusers@iptel.org mailing list and discusses issues related to SER operation.

Mailing List Instructions

- Public archives and subscription form: http://mail.iptel.org/mailman/listinfo/serusers (http://mail.iptel.org/mailman/listinfo/serusers)
- To post, send an email to serusers@iptel.org
- If you think you encountered an error, please submit the following information to avoid unnecessary round-trip times:
  - Name and version of your operating system -- you can obtain it by calling **uname -a**
  - ser distribution: release number and package

- ser build -- you can obtain it by calling **ser -V**
- Your ser configuration file
- ser logs -- with default settings few logs are printed to **syslog** facility which typically dumps them to `/var/log/messages`. To enable detailed logs dumped to `stderr`, apply the following configuration options: **debug=8, log_stderror=yes, fork=no**.
- Captured SIP messages -- you can obtain them using tools such as ngrep or ethereal.

If you are concerned about your privacy and do not wish your queries to be posted and archived publicly, you may post to serhelp@iptel.org. E-mails to this address are only forwarded to iptel.org's ser development team. However, as the team is quite busy you should not be surprised to get replies with considerable delay.

1.9. More Information

Most up-to-date information including latest and most complete version of this documentation is always available at our website, http://www.iptel.org/ser/. The site includes links to other important information about ser, such as installation guidelines (INSTALL), download links, development pages, programmer's manual, etc.

A SIP tutorial (slide set) is available at http://www.iptel.org/sip/ .

1.10. Release Notes

Release notes for SIP Express Router (ser)
****

$Id: NEWS,v 1.15.2.1 2003/08/27 07:57:08 calrissian Exp $

****
* Changes introduced in 0.8.11
****

```
+-------------------------------------------------------+
| CAUTION: the 0.8.11 release include changes which     |
| are incompatible with scripts and databases used      |
| in previous versions. Care is advised when upgrading  |
| from previous releases to 0.8.11.                     |
+-------------------------------------------------------+
```

New features
=============
- RFC3261 support
- TCP support and cross-transport forwarding [core]
- loose routing support [rr module]
- New modules
- vm -- voicemail interface [vm]
- ENUM support [enum]

- presence agent [pa]
- dynamic domain management -- allows to manipulate
  hosting of multiple domains in run-time [module]
- flat-text-file database support [dbtext]
- rich access control lists [permissions]
- Feature Improvements
- click-to-dial, which is based on improved tm/FIFO
  that better supports external applications [tm module]
- web accounting -- acc module can report to serweb
  on placed calls [acc module]
- improved exec module (header fields passed now
  as environment variables to scripts) [exec module]
- Architectural Improvements
- powerpc fast locking support
- netbsd support
- 64 bits arch. support (e.g. netbsd/sparc64).
- New Experimental Features (not tested at all yet)
- nathelper utility for Cisco/ATA NAT traversal [nathelper]
- another NAT traversal utility [mangler]
- postgress support [postgress]
- pdt module (prefix2domain) [pdt]

Changes to use of ser scripts
=============================

About Multiple Transport Support

SER now suports multiple transport protocols: UDP and TCP. As there may be UAs which support only either protocol and cannot speak to each other directly, we recommend to alway record-route SIP requests, to keep the transport-translating SER in path. Also, if a destination transport is not known, stateful forwarding is recommended -- use of stateless forwarding for TCP2UDP would result in loss of reliability.

core

- reply_route has been renamed to failure_route -- the old name caused
  too much confusion
- forward_tcp and forward_udp can force SER to forward via specific
  transport protocol

acc module:

- radius and sql support integrated in this module; you need to
  recompile to enable it
- acc_flag is now called log_flag to better reflect it relates
  to the syslog mode (as opposed to sql/radius); for the same
  reasons, the accounting action is now called "acc_log_request"
  and the option for missed calls "log_missed_calls"
- log_fmt allows now to specify what will be printed to syslog

auth module:

- auth module has been split in auth, auth_db, auth_radius, group
  group_radius, uri and uri_radius
- all the parameters that were part of former auth module are now
  part of auth_db module
- auth_db module contains all functions needed for database
  authentication
- auth_radius contains functions needed for radius authentication
- group module contains group membership checking functions
- group_radius contains radius group membeship checking functions
- is_in_group has been renamed to is_user_in and places to groups
  module
- check_to and check_from have been moved to the uri module

im module:

- im is no longer used and has been obsoleted by TM

exec module:

- exec_uri and exec_user have been obsoleted by exec_dset;
  exec_dset is identical to exec_uri in capabilities; it
  additionaly passes content of request elements (header
  fields and URI parts) in environment variables; users of
  exec_user can use exec_dset now and use the "URI_USER"
  variable to learn user part of URI
- exec_dset and exec_msg return false, if return value of
  script does not euqal zero
- exec_dset takes an additional parameter, which enables
  validation of SIP URIs returned by external application

jabber module:

- presence support for Jabber users is enabled loading the PA
  module and using handle_subscribe("jabber") for SUBSCRIBE
  requests to jabber user

msilo module:

- m_store has now a parameter to set what should be considered
  for storing as destination uri. This enables support for saving
  the messages on negative replies.

radius_acc module:

- radius_acc module has been removed and radius accounting
  is now part of acc module

registrar/usrloc modules:

- multi domain support, the modules user username@domain as AOR
  if enabled
- descent modification time ordering of contacts
- case sensitive/insensitive comparison of URI can be enabled

rr module:

- addRecordRoute has been replaced with record_route
- rewriteFromRoute has been replaced with loose_route()
- a new option, "enable_full_lr" can be set to make life
  with misimplemented UAs easier and put LR in from "lr=on"
- rr module can insert two Record-Route header fields when
  necesarry (disconnected networks, UDP->TCP and so on)

tm module:

- t_reply_unsafe, used in former versions within reply_routes,
  is deprecated; now t_reply is used from any places in script
- t_on_negative is renamed to t_on_failure -- the old name just
  caused too much confusion
- FIFO t_uac used by some applications (like serweb) has been
  replaced with t_uac_dlg (which allows easier use by dialog-
  oriented applications, like click-to-dial)
- if you wish to do forward to another destination from
  failure_route (reply_route formerly), you need to call t_relay
  or t_relay_to explicitely now
- t_relay_to has been replaced with t_relay_to_udp and t_relay_to_tcp

Chapter 2. Introduction to SER

2.1. Request Routing and SER Scripts

The most important concept of every SIP server is that of request routing. The request routing logic determines the next hop of a request. It can be for example used to implement user location service or enforce static routing to a gateway. Real-world deployments actually ask for quite complex routing logic, which needs to reflect static routes to PSTN gateways, dynamic routes to registered users, authentication policy, capabilities of SIP devices, etc.

SER's answer to this need for routing flexibility is a routing language, which allows administrators to define the SIP request processing logic in a detailed manner. They can for example easily split SIP traffic by method or destination, perform user location, trigger authentication, verify access permissions, and so on.

The primary building block of the routing language are *actions*. There are built-in actions (like **forward** for stateless forwarding or **strip** for stripping URIs) as well as external actions imported from shared library modules. All actions can be combined in compound actions by enclosing them in braces, e.g. **{a1(); a2();}**. Actions are aggregated in one or more *route blocks*. Initially, only the default routing block denoted by **route[0]** is called. Other routing blocks can be called by the action **route(blocknumber)**, recursion is permitted. The language includes *conditional statements*.

The routing script is executed for every received request in sequential order. Actions may return positive/negative/zero value. Positive values are considered success and evaluated as TRUE in conditional expressions. Negative values are considered FALSE. Zero value means error and leaves execution of currently processed route block. The route block is left too, if **break** is explicitly called from it.

The easiest and still very useful way for ser users to affect request routing logic is to determine next hop statically. An example is routing to a PSTN gateway whose static IP address is well known. To configure static routing, simply use the action **forward(IP_address, port_number)**. This action forwards an incoming request "as is" to the destination described in action's parameters.

Example 2-1. Static Forwarding

```
# if requests URI is numerical and starts with
# zero, forward statelessly to a static destination

if (uri=~"^sip:0[0-9]*@iptel.org") {
    forward( 192.168.99.3, 5080 );
}
```

However, static forwarding is not sufficient in many cases. Users desire mobility and change their location frequently. Lowering costs for termination of calls in PSTN requires locating a least-cost gateway. Which next-hop is taken may depend on user's preferences. These and many other scenarios need the routing logic to be more dynamic. We describe in Section 2.2 how to make request processing subject to various conditions and in Section 2.3 how to determine next SIP hop.

2.2. Conditional Statements

A very useful feature is the ability to make routing logic depend on a condition. A script condition may for example distinguish between request processing for served and foreign domains, IP and PSTN routes, it may split traffic by method or username, it may determine whether a request should be authenticated or not, etc. ser allows administrators to form conditions based on properties of processed request, such as method or uri, as well as on virtually any piece of data on the Internet.

Example 2-2. Conditional Statement

This example shows how a conditional statement is used to split incoming requests between a PSTN gateway and a user location server based on request URI.

```
# if request URI is numerical, forward the request to PSTN gateway...
if (uri=~"^sip:[0-9]+@foo.bar") { # match using a regular expression
    forward( gateway.foo.bar, 5060 );
} else { # ... forward the request to user location server otherwise
    forward( userloc.foo.bar, 5060 );
};
```

Conditional statements in ser scripts may depend on a variety of expressions. The simplest expressions are action calls. They return true if they completed successfully or false otherwise. An example of an action frequently used in conditional statements is **search** imported from textops module. **search** action leverages textual nature of SIP and compares SIP requests against a regular expression. The action returns true if the expression matched, false otherwise.

Example 2-3. Use of search Action in Conditional Expression

```
# prevent strangers from claiming to belong to our domain;
# if sender claims to be in our domain in From header field,
# better authenticate him
if (search("(f|From): .*@mydomain.com)) {
    if (!(proxy_authorize("mydomain.com" /* realm */,"subscriber" /* table name */ ))) {
           proxy_challenge("mydomain.com /* ream */, "1" /* use qop */ );
           break;
    }
}
```

As modules may be created, which export new functions, there is virtually no limitation on what functionality ser conditions are based on. Implementers may introduce new actions whose return status depends on request content or any external data as well. Such actions can query SQL, web, local file systems or any other place which can provide information wanted for request processing.

Furthermore, many request properties may be examined using existing built-in operands and operators. Available left-hand-side operands and legal combination with operators and right-hand-side operands are described in Table 2-1. Expressions may be grouped together using logical operators: negation (**!**), AND (**&&**), OR (**||** and precedence parentheses (()).

2.2.1. Operators and Operands

There is a set of predefined operators and operands in ser, which in addition to actions may be evaluated in conditional expressions.

Left hand-side operands, which ser understands are the following:

- *method*, which refers to request method such as REGISTER or INVITE
- *uri*, which refers to current request URI, such as "sip:john.doe@foo.bar"

  Note: Note that "uri" always refers to current value of URI, which is subject to change be uri-rewriting actions.

- *src_ip*, which refers to IP address from which a request came.

  Warning

  Note that comparison of src_ip to an IP address may cause DNS lookups and delay request processing. To avoid DNS lookups, don't enclose IP addresses in quotes. Otherwise, reverse DNS lookup can be performed to compare to host aliases.

- *dst_ip* refers to server's IP address at which a request was received
- *src_port* port number from which a SIP request came

ser understands the following operators:

- == stands for equity
- =~ stands for regular expression matching
- logical operators: and, or, negation, parentheses (C-notation for the operators may be used too)

Table 2-1. Valid Combinations of Operands and Operators in Expressions

left-hand-side operand	valid operators	valid right-hand side operators	examples/comments
method	== (exact match), =~ (regular expression matching)	string	method=="INVITE" || method=="ACK" || method=="CANCEL"
uri	== (exact match), =~ (regular expression matching)	string	uri=="sip:foo@bar.com" matches only if exactly this uri is in request URI

left-hand-side operand	valid operators	valid right-hand side operators	examples/comments
	== (exact match)	myself	the expression uri==myself is true if the host part in request URI equals a server name or a server alias (set using the alias option in configuration file)
src_ip	== (match)	IP, IP/mask_length, IP/mask, hostname, myself	src_ip==192.168.0.0/16 matches requests coming from a private network
dst_ip	== (match)	IP, IP/mask_length, IP/mask, hostname, myself	dst_ip==127.0.0.1 matches if a request was received via loopback interface
src_port	== (match)	port number	port number from which a request was sent, e.g. src_port==5060

Example 2-4. More examples of use of ser operators and operands in conditional statements

```
# using an action as condition input; in this
# case, an actions 'search' looks for Contacts
# with private IP address in requests; the condition
# is processed if such a contact header field is
# found

if (search("^(Contact|m): .*@(192\.168\.|10\.|172\.16)")) {
# ....

# this condition is true if request URI matches
# the regular expression "@bat\.iptel\.org"
    if (uri=~"@bat\.iptel\.org") {
# ...

# and this condition is true if a request came
# from an IP address (useful for example for
# authentication by IP address if digest is not
# supported) AND the request method is INVITE

# if ( (src_ip==192.68.77.110 and method=="INVITE")
# ...
```

2.2.2. URI Matching

URI matching expressions have a broad use in a SIP server and deserve more explanation. Typical uses of URI matching include implementation of numbering plans, domain matching, binding external applications to specific URIs, etc. This section shows examples of typical applications of URI-matching.

2.2.2.1. Domain Matching

One of most important uses of URI matching is deciding whether a request is targeted to a served or outside domain. Typically, different request processing applies. Requests for outside domains are simply forwarded to them, whereas more complex logic applies to requests for a served domain. The logic may include saving user's contacts when REGISTER requests are received, forwarding requests to current user's location or a PSTN gateways, interaction with external applications, etc.

The easiest way to decide whether a request belongs a served domain is using the **myself** operand. The expression "uri==myself" returns true if domain name in request URI matches name of the host at which ser is running. This may be insufficient in cases when server name is not equal to domain name for which the server is responsible. For example, the "uri==myself" condition does not match if a server "sipserver.foo.bar" receives a request for "sip:john.doe@foo.bar". To match other names in URI than server's own, set up the `alias` configuration option. The option may be used multiple times, each its use adds a new item to a list of aliases. The myself condition returns then true also for any hostname on the list of aliases.

Example 2-5. Use of uri==myself Expression

```
# ser powers a domain "foo.bar" and runs at host sipserver.foo.bar;
# Names of served domains need to be stated in the aliases
# option; myself would not match them otherwise and would only
# match requests with "sipserver.foo.bar" in request-URI
alias="foo.bar"
alias="sales.foo.bar"
route[0] {
        if (uri==myself) {
            # the request either has server name or some of the
            # aliases in its URI
            log(1,"request for served domain")
            # some domain-specific logic follows here ....
        } else {
            # aha -- the server is not responsible for this
            # requests; that happens for example with the following URIs
            #  - sip:a@marketing.foo.bar
            #  - sip:a@otherdomain.bar
            log(1,"request for outbound domain");
            # outbound forwarding
            t_relay();
        };
}
```

It is possible to recognize whether a request belongs to a domain using regular expressions too. Care needs to be paid to construction of regular expressions. URI syntax is rich and an incorrect expression would result in incorrect call processing. The following example shows how an expression for domain matching can be formed.

Example 2-6. Domain Matching Using Regular Expressions

In this example, server named "sip.foo.bar" with IP address 192.168.0.10 is responsible for the "foo.bar" domain. That means, requests with the following hostnames in URI should be matched:

- foo.bar, which is the name of server domain
- sip.foo.bar, since it is server's name and some devices put server's name in request URI
- 192.168.0.10, since it is server's IP address and some devices put server's IP address in request URI

Note how this regular expression is constructed. In particular:

- User name is optional (it is for example never included in REGISTER requests) and there are no restrictions on what characters it contains. That is what *(.+@)?* mandates.
- Hostname must be followed by port number, parameters or headers -- that is what the delimiters *[:;\?]* are good for. If none it these follows, the URI must be ended (*$*). Otherwise, longer hostnames such as 192.168.0.101 or foo.bar.otherdomain.com would mistakenly match.
- Matches are case-insensitive. All hostnames "foo.bar", "FOO.BAR" and "FoO.bAr" match.

```
if (uri=~"^sip:(.+@)?(192\.168\.0\.10|(sip\.)?foo\.bar)([:;\?].*)?$")
      log(1, "yes, it is a request for our domain");
      break;
 };
```

2.2.2.2. Numbering Plans

Other use of URI matching is implementation of dialing plans. A typical task when designing a dialing plan for SIP networks is to distinguish between "pure-IP" and PSTN destinations. IP users typically have either alphanumerical or numerical usernames. The numerical usernames are convenient for PSTN callers who can only use numeric keypads. Next-hop destination of IP users is looked up dynamically using user location database. On the other hand, PSTN destinations are always indicated by nummerical usernames. Requests to PSTN are statically forwarded to well-known PSTN gateways.

Example 2-7. A simple Numbering Plan

This example shows a simple dialing plan which reserves dialing prefix "8" for IP users, other numbers are used for PSTN destinations and all other non-nummerical usernames are used for IP users.

```
# is it a PSTN destination? (is username nummerical and does not begin with 8?)
if (uri=~"^sip:[0-79][0-9]*@") { # ... forward to gateways then;
      # check first to which PSTN destination the requests goes;
      # if it is US (prefix "1"), use the gateway 192.168.0.1...
      if (uri=~"^sip:1") {
           # strip the leading "1"
           strip(1);
           forward(192.168.0.1, 5060);
      } else {
           # ... use the gateway 10.0.0.1 for all other destinations
```

```
            forward(10.0.0.1, 5060);
        }
        break;
} else {
        # it is an IP destination -- try to lookup it up in user location DB
        if (!lookup("location")) {
            # bad luck ... user off-line
            sl_send_reply("404", "Not Found");
            break;
        }
        # user on-line...forward to his current destination
        forward(uri:host,uri:port);
}
```

2.3. Request URI Rewriting

The ability to give users and services a unique name using URI is a powerful tool. It allows users to advertise how to reach them, to state to whom they wish to communicate and what services they wish to use. Thus, the ability to change URIs is very important and is used for implementation of many services. "Unconditional forwarding" from user "boss" to user "secretary" is a typical example of application relying on change of URI address.

ser has the ability to change request URI in many ways. A script can use any of the following built-in actions to change request URI or a part of it: **rewriteuri**, **rewritehost**, **rewritehostport**, **rewriteuser**, **rewriteuserpass** and **rewriteport**. When later in the script a forwarding action is encountered, the action forwards the request to address in the rewritten URI.

Example 2-8. Rewriting URIs

```
if (uri=~"dan@foo.bar") {
    rewriteuri("sip:bla@somewherelse.com")
    # forward statelessly to the destination in current URI, i.e.,
    # to sip:bla@somewherelese.com:5060
    forward( uri:host, uri:port);
}
```

Two more built-in URI-rewriting commands are of special importance for implementation of dialing plans and manipulation of dialing prefixes. **prefix(s)** , inserts a string "s" in front of SIP address and **strip(n)** takes away the first "n" characters of a SIP address. See Table 2-2 for examples of use of built-in URI-rewriting actions.

Commands exported by external modules can change URI too and many do so. The most important application is changing URI using the user location database. The command **lookup(table)** looks up current user's location and rewrites user's address with it. If there is no registered contact, the command returns a negative value.

Example 2-9. Rewriting URIs Using User Location Database

```
# store user location if a REGISTER appears
if (method=="REGISTER") {
   save("mydomain1");
} else {
# try to use the previously registered contacts to
# determine next hop
   if(lookup("mydomain1")) {
     # if found, forward there...
     t_relay();
   } else {
     # ... if no contact on-line, tell it upstream
     sl_send_reply("404", "Not Found" );
   };
};
```

External applications can be used to rewrite URI too. The "exec" module provides script actions, which start external programs and read new URI value from their output. **exec_dset** both calls an external program, passes SIP request elements to it, waits until it completes, and eventually rewrites current destination set with its output.

It is important to realize that ser operates over *current URI* all the time. If an original URI is rewritten by a new one, the original will will be forgotten and the new one will be used in any further processing. In particular, the uri matching operand and the user location action **lookup** always take current URI as input, regardless what the original URI was.

Table 2-2 shows how URI-rewriting actions affect an example URI, sip:12345@foo.bar:6060.

Table 2-2. URI-rewriting Using Built-In Actions

Example Action	Resulting URI
rewritehost("192.168.0.10") rewrites the hostname in URI, other parts (including port number) remain unaffected.	sip:12345@192.168.10:6060
rewriteuri("sip:alice@foo.bar"); rewrites the whole URI completely.	sip:alice@foo.bar
rewritehostport("192.168.0.10:3040")rewrites both hostname and port number in URI.	sip:12345@192.168.0.10:3040
rewriteuser("alice") rewrites user part of URI.	sip:alice@foo.bar:6060
rewriteuserpass("alice:pw") replaces the pair user:password in URI with a new value. Rewriting password in URI is of historical meaning though, since basic password has been replaced with digest authentication.	sip:alice:pw@foo.bar:6060
rewriteport("1234") replaces port number in URI	sip:12345@foo.bar:1234
prefix("9") inserts a string ahead of user part of URI	sip:912345@foo.bar:6060

Example Action	Resulting URI
strip(2) removes leading characters from user part of URI	sip:345@foo.bar:6060

You can verify whether you understood URI processing by looking at the following example. It rewrites URI several times. The question is what is the final URI to which the script fill forward any incoming request.

Example 2-10. URI-rewriting Exercise

```
exec_dset("echo sip:2234@foo.bar; echo > /dev/null");
strip(2);
if (uri=~"^sip:2") {
    prefix("0");
} else {
    prefix("1");
};
forward(uri:host, uri:port);
```

The correct answer is the resulting URI will be "sip:134@foo.bar". **exec_dset** rewrites original URI to "sip:2234@foo.bar", **strip(2)** takes two leading characters from username away resulting in "34@iptel.org", the condition does not match because URI does not begin with "2" any more, so the prefix "1" is inserted.

2.4. Destination Set

Whereas needs of many scenarios can by accommodated by maintaining a single request URI, some scenarios are better served by multiple URIs. Consider for example a user with address john.doe@iptel.org. The user wishes to be reachable at his home phone, office phone, cell phone, softphone, etc. However, he still wishes to maintain a single public address on his business card.

To enable such scenarios, ser allows translation of a single request URI into multiple outgoing URIs. The ability to forward a request to multiple destinations is known as *forking* in SIP language. All outogoing URIs (in trivial case one of them) are called *destination set*. The destination set always includes one default URI, to which additional URIs can be appended. Maximum size of a destination set is limited by a compile-time constant, MAX_BRANCHES, in `config.h`.

Some actions are designed for use with a single URI whereas other actions work with the whole destination set.

Actions which are currently available for creating the destination set are **lookup** from usrloc module and **exec_dset** from exec module. **lookup** fills in the destination set with user contact's registered previously with REGISTER requests. The **exec** actions fill in the destination set with output of an external program. In both cases, current destination set is completely rewritten. New URIs can be appended to destination set by a call to the built-in action **append_branch(uri)**.

Currently supported features which utilize destination sets are *forking* and *redirection*. Action **t_relay** (TM module) for stateful forwarding supports forking. If called with a non-trivial destination set, **t_relay** forks incoming request to all URIs in current destination set. See Example 2-9. If a user previously registered from three locations, the destination set is filled with all of them by **lookup** and the **t_relay** command forwards the incoming request to all these destinations. Eventually, all user's phone will be ringing in parallel.

SIP redirection is another feature which leverages destination sets. It is a very light-weighted method to establish communication between two parties with minimum burden put on the server. In ser, the action **sl_send_reply** (SL module) is used for this purpose. This action allows to generate replies to SIP requests without keeping any state. If the status code passed to the action is 3xx, the current destination set is printed in reply's Contact header fields. Such a reply instructs the originating client to retry at these addresses. (See Example 2-19).

Most other ser actions ignore destination sets: they either do not relate to URI processing (**log**, for example) or they work only with the default URI. All URI-rewriting functions such as **rewriteuri** belong in this category. URI-comparison operands only refer to the first URI (see Section 2.2.1). Also, the built-in action for stateless forwarding, **forward** works only with the default URI and ignores rest of the destination set. The reason is a proxy server willing to fork must guarantee that the burden of processing multiple replies is not put unexpectedly on upstream client. This is only achievable with stateful processing. Forking cannot be used along with stateless **forward**, which thus only processes one URI out of the whole destination set.

2.5. User Location

Mobility is a key feature of SIP. Users are able to use one one or more SIP devices and be reachable at them. Incoming requests for users are forwarded to all user's devices in use. The key concept is that of soft-state registration. Users can -- if in possession of valid credentials -- link SIP devices to their e-mail like address of record. Their SIP devices do so using a REGISTER request, as in Example 2-11. The request creates a binding between the public address of record (To header field) and SIP device's current address (Contact header field).

Example 2-11. REGISTER Request

```
REGISTER sip:192.168.2.16 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.16;branch=z9hG4bKd5e5.5a9947e4.0
Via: SIP/2.0/UDP 192.168.2.33:5060
From: sip:123312@192.168.2.16
To: sip:123312@192.168.2.16
Call-ID: 00036bb9-0fd30217-491b6aa6-0a7092e9@192.168.2.33
Date: Wed, 29 Jan 2003 18:13:15 GMT
CSeq: 101 REGISTER
User-Agent: CSCO/4
Contact: sip:123312@192.168.2.33:5060
Content-Length: 0
Expires: 600
```

Similar requests can be used to query all user's current contacts or to delete them. All Contacts have certain time to live, when the time expires, contact is removed and no longer used for processing of incoming requests.

ser is built to do both: update user location database from received REGISTER requests and look-up these contacts when inbound requests for a user arrive. To achieve high performance, the user location table is stored in memory. In regular intervals (usrloc module's parameter `timer_interval` determines their length), all changes to the in-memory table are backed up in mysql database to achieve peristence accross server reboots. Administrators or application writers can lookup list of current user's contacts stored in memory using the serctl tool (see Section 5.1).

Example 2-12. Use of serctl Tool to Query User Location

```
[jiri@fox jiri]$ sc ul show jiri
<sip:jiri@212.202.172.134>;q=0.00;expires=456
<sip:7271@gateway.foo.bar>;q=0.00;expires=36000
```

Building user location in ser scripts is quite easy. One first needs to determine whether a request is for served domain, as described in Section 2.2.2.1. If that is the case, the script needs to distinguish between REGISTER requests, that update user location table, and all other requests for which next hop is determined from the table. The **save** action is used to update user location (i.e., it writes to it). The **lookup** actions reads from the user location table and fills in destination set with current user's contacts.

Example 2-13. Use of User Location Actions

```
# is the request for my domain ?
if (uri==myself) {
    if (method=="REGISTER") { # REGISTERs are used to update
         save("location");
         break; # that's it, we saved the contacts, exit now
    } else {
         if (!lookup("location") { # no registered contact
            sl_send_reply("404", "Not Found");
            break;
         }
         # ok -- there are some contacts for the user; forward
         # the incoming request to all of them
         t_relay();
    };
};
```

Note that we used the action for stateful forwarding, **t_relay**. That's is because stateful forwarding allows to fork an incoming request to multiple destinations. If we used stateful forwarding, the request would be forwarded only to one uri out of all user's contacts.

2.6. External Modules

ser provides the ability to link the server with external third-party shared libraries. Lot of functionality which is included in the ser distribution is actually located in modules to keep the server "core" compact and clean. Among others, there are modules for checking max_forwards value in SIP requests (maxfwd), transactional processing (tm), record routing (rr), accounting (acc), authentication (auth), SMS gateway (sms), replying requests (sl), user location (usrloc, registrar) and more.

In order to utilize new actions exported by a module, ser must first load it. To load a module, the directive **loadmodule "filename"** must be included in beginning of a ser script file.

Example 2-14. Using Modules

This example shows how a script instructs ser to load a module and use actions exported by it. Particularly, the sl module exports an action **sl_send_reply** which makes ser act as a stateless user agent and reply all incoming requests with 404.

```
# first of all, load the module!
loadmodule "/usr/lib/ser/modules/sl.so
route{
    # reply all requests with 404
    sl_send_reply("404", "I am so sorry -- user not found");
}
```

> **Note:** Note that unlike with core commands, all actions exported by modules must have parameters enclosed in quotation marks in current version of ser. In the following example, the built-in action **forward** for stateless forwarding takes IP address and port numbers as parameters without quotation marks whereas a module action **t_relay** for stateful forwarding takes parameters enclosed in quotation marks.
>
> **Example 2-15. Parameters in built-in and exported actions**
>
> ```
> # built-in action doesn't enclose IP addresses and port numbers
> # in quotation marks
> forward(192.168.99.100, 5060);
> # module-exported functions enclose all parameters in quotation
> # marks
> t_relay_to_udp("192.168.99.100", "5060");
> ```

Many modules also allow users to change the way how they work using predefined parameters. For example, the authentication module needs to know location of MySQL database which contains users' security credentials. How module parameters are set using the **modparam** directive is shown in Example 2-16. **modparam** always contains identification of module, parameter name and parameter value. Description of parameters available in modules is available in module documentation.

Yet another thing to notice in this example is module dependency. Modules may depend on each other. For example, the authentication modules leverages the mysql module for accessing mysql databases and sl module for generating authentication challenges. We recommend that modules are loaded in dependency order to avoid ambiguous server behaviour.

Example 2-16. Module Parameters

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

# load first modules on which 'auth' module depends;
# sl is used for sending challenges, mysql for storage
# of user credentials
```

```
loadmodule "modules/sl/sl.so"
loadmodule "modules/mysql/mysql.so"
loadmodule "modules/auth/auth.so"

# ------------------ module parameters --------------------------------
# tell the auth module the access data for SQL database:
# username, password, hostname and database name
modparam("auth", "db_url","sql://ser:secret@dbhost/ser")


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

# authenticate all requests prior to forwarding them

route{

        if (!proxy_authorize("foo.bar" /* realm */,
                        "subscriber" /* table name */ )) {
                proxy_challenge("foo.bar", "0");
                break;
        };
        forward(192.168.0.10,5060);
}
```

2.7. Writing Scripts

This section demonstrates simple examples how to configure server's behaviour using the ser request routing language. All configuration scripts follow the ser language syntax, which dictates the following section ordering:

- *global configuration parameters* -- these value affect behaviour of the server such as port number at which it listens, number of spawned children processes, and log-level. See Section 6.1 for a list of available options.
- *module loading* -- these statements link external modules, such as transaction management (tm) or stateless UA server (sl) dynamically. See Section 6.4 for a list of modules included in ser distribution.

  Note: If modules depend on each other, than the depending modules must be loaded after modules on which they depend. We recommend to load first modules **tm** and **sl** because many other modules (authentication, user location, accounting, etc.) depend on these.

- *module-specific parameters* -- determine how modules behave; for example, it is possible to configure database to be used by authentication module.
- one or more *route blocks* containing the request processing logic, which includes built-in actions as well as actions exported by modules. See Section 6.2 for a list of built-in actions.

- optionally, if modules supporting reply processing (currently only TM) are loaded, one or more *failure_route blocks* containing logic triggered by received replies. Restrictions on use of actions within **failure_route** blocks apply -- see Section 6.2 for more information.

2.7.1. Default Configuration Script

The configuration script, `ser.cfg`, is a part of every ser distribution and defines default behaviour. It allows users to register with the server and have requests proxied to each other.

After performing routine checks, the script looks whether incoming request is for served domain. If so and the request is "REGISTER", ser acts as SIP registrar and updates database of user's contacts. Optionally, it verifies user's identity first to avoid unauthorized contact manipulation.

Non-REGISTER requests for served domains are then processed using user location database. If a contact is found for requested URI, script execution proceeds to stateful forwarding, a negative 404 reply is generated otherwise. Requests outside served domain are always statefully forwarded.

Note that this simple script features several limitations:

- By default, authentication is turned off to avoid dependency on mysql. Unless it it turned on, anyone can register using any name and "steal" someone else's calls.
- Even it authentication is turned on, there is no relationship between authentication username and address of record. That means that for example a user authenticating himself correctly with "john.doe" id may register contacts for "gw.bush". Site policy may wish to mandate authentication id to be equal to username claimed in To header field. check_to action from auth module can be used to enforce such a policy.
- There is no dialing plan implemented. All users are supposed to be reachable via user location database. See Section 2.2.2.2 for more information.
- The script assumes users will be using server's name as a part of their address of record. If users wish to use another name (domain name for example), this must be set using the `alias` options. See Section 2.2.2.1 for more information.
- If authentication is turned on by uncommenting related configuration options, clear-text user passwords will by assumed in back-end database.

Example 2-17. Default Configuration Script

```
#
# $Id: ser.cfg,v 1.21.2.1 2003/07/30 16:46:18 andrei Exp $
#
# simple quick-start config script
#

# ----------- global configuration parameters ------------------------

#debug=3          # debug level (cmd line: -dddddddddd)
#fork=yes
#log_stderror=no # (cmd line: -E)
```

```
/* Uncomment these lines to enter debugging mode
debug=7
fork=no
log_stderror=yes
*/

check_via=no # (cmd. line: -v)
dns=no           # (cmd. line: -r)
rev_dns=no      # (cmd. line: -R)
#port=5060
#children=4
fifo="/tmp/ser_fifo"

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

# Uncomment this if you want to use SQL database
#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"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
#loadmodule "/usr/local/lib/ser/modules/auth.so"
#loadmodule "/usr/local/lib/ser/modules/auth_db.so"

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

# -- usrloc params --

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

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

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

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

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

```
# main routing logic

route{

# initial sanity checks -- messages with
# max_forwards==0, or excessively long requests
if (!mf_process_maxfwd_header("10")) {
sl_send_reply("483","Too Many Hops");
break;
};
if (len_gt( max_len )) {
sl_send_reply("513", "Message too big");
break;
};

# we record-route all messages -- to make sure that
# subsequent messages will go through our proxy; that’s
# particularly good if upstream and downstream entities
# use different transport protocol
record_route();
# loose-route processing
if (loose_route()) {
t_relay();
break;
};

# if the request is for other domain use UsrLoc
# (in case, it does not work, use the following command
# with proper names and addresses in it)
if (uri==myself) {

if (method=="REGISTER") {

# Uncomment this if you want to use digest authentication
# if (!www_authorize("iptel.org", "subscriber")) {
# www_challenge("iptel.org", "0");
# break;
# };

save("location");
break;
};

# native SIP destinations are handled using our USRLOC DB
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
break;
};
};
# forward to current uri now; use stateful forwarding; that
# works reliably even if we forward from TCP to UDP
if (!t_relay()) {
sl_reply_error();
};

}
```

2.7.2. Stateful User Agent Server

This examples shows how to make ser act as a stateful user agent (UA). Ability to act as as a stateful UA is essential to many applications which terminate a SIP path. These applications wish to focus on their added value. They do not wish to be involved in all SIP gory details, such as request and reply retransmission, reply formatting, etc. For example, we use the UA functionality to shield SMS gateway and instant message store from SIP transactional processing. The simple example bellow issues a log report on receipt of a new transaction. If we did not use a stateful UA, every single request retransmission would cause the application to be re-executed which would result in duplicated SMS messages, instant message in message store or log reports.

The most important actions are **t_newtran** and **t_reply**. **t_newtran** shields subsequent code from retransmissions. It returns success and continues when a new request arrived. It exits current route block immediately on receipt of a retransmission. It only returns a negative value when a serious error, such as lack of memory, occurs.

t_reply generates a reply for a request. It generates the reply statefully, i.e., it is kept for future retransmissions in memory.

> **Note:** Applications that do not need stateful processing may act as stateless UA Server too. They just use the **sl_send_reply** action to send replies to requests without keeping any state. The benefit is memory cannot run out, the drawback is that each retransmission needs to be processed as a new request. An example of use of a stateless server is shown in Section 2.7.3 and Section 2.7.4.

Example 2-18. Stateful UA Server

```
#
# $Id: uas.cfg,v 1.7 2003/06/03 03:18:12 jiri Exp $
#
# this example shows usage of ser as user agent
# server which does some functionality (in this
# example, 'log' is used to print a notification
# on a new transaction) and behaves statefuly
# (e.g., it retransmits replies on request
# retransmissions)

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

loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"


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

# main routing logic

route{
```

```
# for testing purposes, simply okay all REGISTERs
if (method=="REGISTER") {
log("REGISTER");
sl_send_reply("200", "ok");
break;
};

# create transaction state; abort if error occured
if ( !t_newtran()) {
sl_reply_error();
break;
};

# the following log will be only printed on receipt of
# a new message; retranmissions are absorbed by t_newtran
log(1, "New Transaction Arrived\n");
        # do what you want to do as a sever...
if (uri=~"a@") {
if (!t_reply("409", "Bizzar Error")) {
sl_reply_error();
};
} else {
if (!t_reply("699", "I don't want to chat with you")) {
sl_reply_error();
};
    };
}
```

2.7.3. Redirect Server

The redirect example shows how to redirect a request to multiple destination using 3xx reply. Redirecting requests as opposed to proxying them is essential to various scalability scenarios. Once a message is redirected, ser discards all related state and is no more involved in subsequent SIP transactions (unless the redirection addresses point to the same server again).

The key ser actions in this example are **append_branch** and **sl_send_reply** (sl module).

append_branch adds a new item to the destination set. The destinations set always includes the current URI and may be enhanced up to `MAX_BRANCHES` items. **sl_send_reply** command, if passed SIP reply code 3xx, takes all values in current destination set and adds them to Contact header field in the reply being sent.

Example 2-19. Redirect Server

```
#
# $Id: redirect.cfg,v 1.5 2002/12/09 02:32:57 jiri Exp $
#
# this example shows use of ser as stateless redirect server
#
```

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

loadmodule "modules/sl/sl.so"


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

# main routing logic

route{
# for testing purposes, simply okay all REGISTERs
if (method=="REGISTER") {
log("REGISTER");
sl_send_reply("200", "ok");
break;
};
# rewrite current URI, which is always part of destination ser
rewriteuri("sip:parallel@iptel.org:9");
# append one more URI to the destination ser
append_branch("sip:redirect@iptel.org:9");
# redirect now
sl_send_reply("300", "Redirect");
}
```

2.7.4. Executing External Script

Like in the previous example, we show how to make ser act as a redirect server. The difference is that we do not use redirection addresses hardwired in ser script but get them from external shell commands. We also use ser's ability to execute shell commands to log source IP address of incoming SIP requests.

The new commands introduced in this example are **exec_msg** and **exec_dset**. **exec_msg** takes current requests, starts an external command, and passes the requests to the command's standard input. It also passes request's source IP address in environment variable named `SRCIP`.

exec_dset serves for URI rewriting by external applications. The **exec_dset** action passes current URI to the called external program, and rewrites current destination set with the program's output. An example use would be an implementation of a Least-Cost-Router, software which returns URI of the cheapest PSTN provider for a given destination based on some pricing tables. Example 2-20 is much easier: it prints fixed URIs on its output using shell script **echo** command.

> **Note:** This script works statelessly -- it uses this action for stateless replying, **sl_send_reply**. No transaction is kept in memory and each request retransmission is processed as a brand-new request. That may be a particular concern if the server logic (**exec** actions in this example) is too expensive. See Section 2.7.2 for instructions on how to make server logic stateful, so that retransmissions are absorbed and do not cause re-execution of the logic.

Example 2-20. Executing External Script

```
#
# $Id: exec.cfg,v 1.7 2003/06/03 03:18:12 jiri Exp $
#
# this example shows use of ser as stateless redirect server
# which rewrites URIs using an exernal utility
#

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

loadmodule "modules/exec/exec.so"
loadmodule "modules/sl/sl.so"

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

# main routing logic

route{
# for testing purposes, simply okay all REGISTERs
if (method=="REGISTER") {
log("REGISTER");
sl_send_reply("200", "ok");
break;
};

# first dump the message to a file using cat command
exec_msg("printenv SRCIP > /tmp/exectest.txt; cat >> /tmp/exectest.txt");
# and then rewrite URI using external utility
# note that the last echo command trashes input parameter
if (exec_dset("echo sip:mra@iptel.org;echo sip:mrb@iptel.org;echo>/dev/null")) {
sl_send_reply("300", "Redirect");
} else {
sl_reply_error();
log(1, "alas, rewriting failed\n");
};
}
```

2.7.5. On-Reply Processing (Forward on Unavailable)

Many services depend on status of messages relayed downstream: *forward on busy* and *forward on no reply* to name the most well-known ones. To support implementation of such services, ser allows to return to request processing when request forwarding failed. When a request is reprocessed, new request branches may be initiated or the transaction can be completed at discretion of script writer.

The primitives used are **t_on_failure(r)** and **failure_route[r]{}.** If **t_on_failure** is called before a request is statefuly forwarded and a forwarding failure occurs, ser will return to request processing in a **failure_route** block. Failures include receipt of a SIP error (status code >= 300) from downstream or not receiving any final reply within final response period.

The length of the timer is governed by parameters of the tm module. `fr_timer` is the length of timer set for non-INVITE transactions and INVITE transactions for which no provisional response is received. If a timer hits, it indicates that a downstream server is unresponsive. `fr_inv_timer` governs time to wait for a final reply for an INVITE. It is typically longer than `fr_timer` because final reply may take long time until callee (finds a mobile phone in a pocket and) answers the call.

In Example 2-21, **failure_route[1]** is set to be entered on error using the **t_on_failure(1)** action. Within this reply block, ser is instructed to initiate a new branch and try to reach called party at another destination (sip:nonsense@iptel.org). To deal with the case when neither the alternate destination succeeds, t_on_failure is set again. If the case really occurs, **failure_route[2]** is entered and a last resort destination (sip:foo@iptel.org) is tried.

Example 2-21. On-Reply Processing

```
#
# $Id: onr.cfg,v 1.8 2003/06/03 03:18:12 jiri Exp $
#
# example script showing both types of forking;
# incoming message is forked in parallel to
# 'nobody' and 'parallel', if no positive reply
# appears with final_response timer, nonsense
# is retried (serial forking); than, destination
# 'foo' is given last chance

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

loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"

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

# -- tm params --
# set time for which ser will be waiting for a final response;
# fr_inv_timer sets value for INVITE transactions, fr_timer
# for all others
modparam("tm", "fr_inv_timer", 15 )
modparam("tm", "fr_timer", 10 )

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

# main routing logic

route{
# for testing purposes, simply okay all REGISTERs
if (method=="REGISTER") {
log("REGISTER");
sl_send_reply("200", "ok");
break;
};
# try these two destinations first in parallel; the second
# destination is targeted to sink port -- that will make ser
# wait until timer hits
seturi("sip:nobody@iptel.org");
append_branch("sip:parallel@iptel.org:9");
# if we do not get a positive reply, continue at reply_route[1]
t_on_failure("1");
```

```
# forward the request to all destinations in destination set now
t_relay();
}

failure_route[1] {
# forwarding failed -- try again at another destination
append_branch("sip:nonsense@iptel.org");
log(1,"first redirection\n");
# if this alternative destination fails too, proceed to reply_route[2]
t_on_failure("2");
t_relay();
}

failure_route[2] {
# try out the last resort destination
append_branch("sip:foo@iptel.org");
log(1, "second redirection\n");
# we no more call t_on_negative here; if this destination
# fails too, transaction will complete
t_relay();
}
```

Chapter 3. Server Operation

3.1. Recommended Operational Practices

Operation of a SIP server is not always easy task. Server administrators are challenged by broken or misconfigured user agents, network and host failures, hostile attacks and other stress-makers. All such situations may lead to an operational failure. It is sometimes very difficult to figure out the root reason of a failure, particularly in a distributed environment with many SIP components involved. In this section, we share some of our practices and refer to tools which have proven to make life of administrators easier

1. Keeping track of messages is good

Frequently, operational errors are discovered or reported with a delay. Users frustrated by an error frequently approach administrators and scream "even though my SIP requests were absolutely ok yesterday, they were mistakenly denied by your server". If administrators do not record all SIP traffic at their site, they will be no more able to identify the problem reason. We thus recommend that site operators record all messages passing their site and keep them stored for some period of time. They may use utilities such as ngrep or tcpdump . There is also a utility scripts/harv_ser.sh in ser distribution for post-processing of captured messages. It summarizes messages captured by reply status and user-agent header field.

2. Real-time Traffic Watching

Looking at SIP messages in real-time may help to gain understanding of problems. Though there are commercial tools available, using a simple, text-oriented tool such as ngrep makes the job very well thanks to SIP's textual nature.

Example 3-1. Using ngrep

In this example, all messages at port 5060 which include the string "bkraegelin" are captured and displayed

```
[jiri@fox s]$ ngrep bkraegelin@ port 5060
interface: eth0 (195.37.77.96/255.255.255.240)
filter: ip and ( port 5060 )
match: bkraegelin@
#
U +0.000000 153.96.14.162:50240 -> 195.37.77.101:5060
REGISTER sip:iptel.org SIP/2.0.
Via: SIP/2.0/UDP 153.96.14.162:5060.
From: sip:bkraegelin@iptel.org.
To: sip:bkraegelin@iptel.org.
Call-ID: 0009b7aa-1249b554-6407d246-72d2450a@153.96.14.162.
Date: Thu, 26 Sep 2002 22:03:55 GMT.
CSeq: 101 REGISTER.
Expires: 10.
Content-Length: 0.
.

#
U +0.000406 195.37.77.101:5060 -> 153.96.14.162:5060
SIP/2.0 401 Unauthorized.
Via: SIP/2.0/UDP 153.96.14.162:5060.
From: sip:bkraegelin@iptel.org.
To: sip:bkraegelin@iptel.org.
Call-ID: 0009b7aa-1249b554-6407d246-72d2450a@153.96.14.162.
```

```
CSeq: 101 REGISTER.
WWW-Authenticate: Digest realm="iptel.org", nonce="3d9385170000000043acbf6ba9c9741790e0c57adee73812",
Server: Sip EXpress router(0.8.8 (i386/linux)).
Content-Length: 0.
Warning: 392 127.0.0.1:5060 "Noisy feedback tells: pid=31604 req_src_ip=153.96.14.162 in_uri=sip:ipte
```

3. Tracing Errors in Server Chains

A request may pass any number of proxy servers on its path to its destination. If an error occurs in the chain, it is difficult for upstream troubleshooters and/or users complaining to administrators to learn more about error circumstances. ser does its best and displays extensive diagnostics information in SIP replies. It allows troubleshooters and/or users who report to troubleshooters to gain additional knowledge about request processing status. This extended debugging information is part of the warning header field. See Example 3-1 for an illustration of a reply that includes such a warning header field. The header field contains the following pieces of information:

- Server's IP Address -- good to identify from which server in a chain the reply came.
- Incoming and outgoing URIs -- good to learn for which URI the reply was generated, as it may be rewritten many times in the path. Particularly useful for debugging of numbering plans.
- Number of Via header fields in replied request -- that helps in assessment of request path length. Upstream clients would not know otherwise, how far away in terms of SIP hops their requests were replied.
- Server's process id. That is useful for debugging to discover situations when mutliple servers listen at the same address.
- IP address of previous SIP hop as seen by the SIP server.

If server administrator is not comfortable with disclosing all this information, he can turn them off using the `sip_warning` configuration option.

A nice utility for debugging server chains is sipsak, SIP Swiss Army Knife, traceroute-like tool for SIP developed at iptel.org. It allows you to send OPTIONS request with low, increasing Max-Forwards header-fields and follow how it propagates in SIP network. See its webpage at http://sipsak.berlios.de/ (http://sipsak.berlios.de/).

Example 3-2. Use of SIPSak for Learning SIP Path

```
[jiri@bat sipsak]$ ./sipsak -T -s sip:7271@iptel.org
warning: IP extract from warning activated to be more informational
0: 127.0.0.1 (0.456 ms) SIP/2.0 483 Too Many Hops
1: ?? (31.657 ms) SIP/2.0 200 OK
without Contact header
```

Note that in this example, the second hop server does not issue any warning header fields in replies and it is thus impossible to display its IP address in SIPsak's output.

4. Watching Server Health

Watching Server's operation status in real-time may also be a great aid for trouble-shooting. ser has an excellent facility, a FIFO server, which allows UNIX tools to access server's internals. (It is similar to how Linux tool access Linux kernel via the proc file system.) The FIFO server accepts commands via a FIFO (named pipe) and returns data asked for. Administrators do not need to learn details of the FIFO communication and can serve themselves using a front-end utility serctl. Of particular interest for monitoring server's operation are serctl commands **ps** and **moni**. The former displays running ser processes, whereas the latter shows statistics.

Example 3-3. serctl ps command

This example shows 10 processes running at a host. The process 0, "attendant" watches child processes and terminates all of them if a failure occurs in any of them. Processes 1-4 listen at local interface and processes 5-8 listen at Ethernet interface at port number 5060. Process number 9 runs FIFO server, and process number 10 processes all server timeouts.

```
[jiri@fox jiri]$ serctl ps
0 31590 attendant
1 31592 receiver child=0 sock=0 @ 127.0.0.1::5060
2 31595 receiver child=1 sock=0 @ 127.0.0.1::5060
3 31596 receiver child=2 sock=0 @ 127.0.0.1::5060
4 31597 receiver child=3 sock=0 @ 127.0.0.1::5060
5 31604 receiver child=0 sock=1 @ 195.37.77.101::5060
6 31605 receiver child=1 sock=1 @ 195.37.77.101::5060
7 31606 receiver child=2 sock=1 @ 195.37.77.101::5060
8 31610 receiver child=3 sock=1 @ 195.37.77.101::5060
9 31611 fifo server
10 31627 timer
```

5. Is Server Alive

It is essential for solid operation to know continuously that server is alive. We've been using two tools for this purpose. sipsak does a great job of "pinging" a server, which may be used for alerting on unresponsive servers.

monit is a server watching utility which alerts when a server dies.

6. Dealing with DNS

SIP standard leverages DNS. Administrators of ser should be aware of impact of DNS on server's operation. Server's attempt to resolve an unresolvable address may block a server process in terms of seconds. To be safer that the server doesn't stop responding due to being blocked by DNS resolving, we recommend the following practices:

- Start a sufficient number of children processes. If one is blocked, the other children will keep serving.
- Use DNS caching. For example, in Linux, there is an nscd daemon available for this purpose.
- Process transactions statefully if memory allows. That helps to absorb retransmissions without having to resolve DNS for each of them.
- In your script expressions compare IP addresses without enclosing them in quotes. Enclosing IP addresses in quotes may cause additional reverse DNS lookups.

Example 3-4. IP Address Comparison

```
# this expression takes no DNS lookup
if (src_ip==192.168.2.15) {
....
# whereas this does
if (src_ip=="192.168.2.15") {
...
```

7. Logging

ser by default logs to syslog facility. It is very useful to watch log messages for abnormal behaviour. Log messages, subject to syslog configuration may be stored at different files, or even at remote systems. A typical location of the log file is `/var/log/messages`.

> **Note:** One can also use other syslogd implementation. metalog (http://metalog.sourceforge.net/ (http://http://metalog.sourceforge.net//)) features regular expression matching that enables to filter and group log messages.

For the purpose of debugging configuration scripts, one may want to redirect log messages to console not to pollute syslog files. To do so configure ser in the following way:

- Attach ser to console by setting `fork=no`.
- Set explicitely at which address ser should be listening, e.g., `listen=192.168.2.16`.
- Redirect log messages to standard error by setting `log_stderror=yes`
- Set appropriately high log level. (Be sure that you redirected logging to standard output. Flooding system logs with many detailed messages would make the logs difficult to read and use.) You can set the global logging threshold value with the option `debug=nr`, where the higher `nr` the more detailed output. If you wish to set log level only for some script events, include the desired log level as the first parameter of the **log** action in your script. The messages will be then printed if **log**'s level is lower than the global threshold, i.e., the lower the more noisy output you get.

Example 3-5. Logging Script

```
#
# $Id: logging.cfg,v 1.1 2003/02/27 20:29:25 jiri Exp $
#
# logging example
#

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

fork=no
listen=192.168.2.16
log_stderror=yes
debug=3
```

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

# main routing logic

route{
# for testing purposes, simply okay all REGISTERs
if (method=="REGISTER") {
log(1, "REGISTER received\n");
} else {
log(1, "non-REGISTER received\n");
};
if (uri=~"sip:.*[@:]iptel.org") {
log(1, "request for iptel.org received\n");
} else {
log(1, "request for other domain received\n");
};
}
```

The following SIP message causes then logging output as shown bellow.

```
REGISTER sip:192.168.2.16 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.33:5060
From: sip:113311@192.168.2.16
To: sip:113311@192.168.2.16
Call-ID: 00036bb9-0fd305e2-7daec266-212e5ec9@192.168.2.33
Date: Thu, 27 Feb 2003 15:10:52 GMT
CSeq: 101 REGISTER
User-Agent: CSCO/4
Contact: sip:113311@192.168.2.33:5060
Content-Length: 0
Expires: 600

[jiri@cat sip_router]$ ./ser -f examples/logging.cfg
Listening on
              192.168.2.16 [192.168.2.16]::5060
Aliases: cat.iptel.org:5060 cat:5060
WARNING: no fork mode
 0(0) INFO: udp_init: SO_RCVBUF is initially 65535
 0(0) INFO: udp_init: SO_RCVBUF is finally 131070
 0(17379) REGISTER received
 0(17379) request for other domain received
```

8. Labeling Outbound Requests

Without knowing, which pieces of script code a relayed request visited, trouble-shooting would be difficult. Scripts typically apply different processing to different routes such as to IP phones and PSTN gateways. We thus recommend to label outgoing requests with a label describing the type of processing applied to the request.

Attaching "routing-history" hints to relayed requests is as easy as using the **append_hf** action exported by textops module. The following example shows how different labels are attached to requests to which different routing logic was applied.

Example 3-6. "Routing-history" labels

```
# is the request for our domain?
# if so, process it using UsrLoc and label it so.
if (uri=~[@:\.]domain.foo") {
   if (!lookup("location")) {
    sl_send_reply("404", "Not Found");
    break;
   };
   # user found -- forward to him and label the request
   append_hf("P-hint: USRLOC\r\n");
} else {
# it is an outbound request to some other domain --
# indicate it in the routing-history label
   append_hf("P-hint: OUTBOUND\r\n");
};
t_relay();
```

This is how such a labeled requests looks like. The last header field includes a label indicating the script processed the request as outbound.

```
#
U 2002/09/26 02:03:09.807288 195.37.77.101:5060 -> 203.122.14.122:5060
SUBSCRIBE sip:rajesh@203.122.14.122 SIP/2.0.
Max-Forwards: 10.
Via: SIP/2.0/UDP 195.37.77.101;branch=53.b44e9693.0.
Via: SIP/2.0/UDP 203.122.14.115:16819.
From: sip:rajeshacl@iptel.org;tag=5c7cecb3-cfa2-491d-a0eb-72195d4054c4.
To: sip:rajesh@203.122.14.122.
Call-ID: bd6c45b7-2777-4e7a-b1ae-11c9ac2c6a58@203.122.14.115.
CSeq: 2 SUBSCRIBE.
Contact: sip:203.122.14.115:16819.
User-Agent: Windows RTC/1.0.
Proxy-Authorization: Digest username="rajeshacl", realm="iptel.org", algorithm="MD5", uri="sip:rajesh
Expires: 1800.
Content-Length: 0.
P-hint: OUTBOUND.
```

3.2. HOWTOs

This section is a "cookbook" for dealing with common tasks, such as user management or controlling access to PSTN gateways.

3.2.1. User Management

There are two tasks related to management of SIP users: maintaining user accounts and maintaining user contacts. Both these jobs can be done using the serctl command-line tool. Also, the complimentary web interface, serweb, can be used for this purpose as well.

If user authentication is turned on, which is a highly advisable practice, user account must be created before a user can log in. To create a new user account, call the **serctl add** utility with username, password and email as parameters. It is important that the environment `SIP_DOMAIN` is set to your realm and matches realm values used in your script. The realm value is used for calculation of credentials stored in subscriber database, which are bound permanently to this value.

```
[jiri@cat gen_ha1]$ export SIP_DOMAIN=foo.bar
[jiri@cat gen_ha1]$ serctl add newuser secret newuser@foo.bar
MySql Password:
new user added
```

serctl can also change user’s password or remove existing accounts from system permanently.

```
[jiri@cat gen_ha1]$ serctl passwd newuser newpassword
MySql Password:
password change succeeded
[jiri@cat gen_ha1]$ serctl rm newuser
MySql Password:
user removed
```

User contacts are typically automatically uploaded by SIP phones to server during registration process and administrators do not need to worry about them. However, users may wish to append permanent contacts to PSTN gateways or to locations in other administrative domains. To manipulate the contacts in such cases, use serctl ul tool. Note that this is the only correct way to update contacts -- direct changes to back-end MySql database do not affect server’s memory. Also note, that if persistence is turned off (usrloc "db_mode" parameter set to "0"), all contacts are gone on server reboot. Make sure that persistence is enabled if you add permanent contacts.

To add a new permanent contact for a user, call serctl ul add <username> <contact>. To delete all user’s contacts, call serctl ul rm <username>. serctl ul show <username> prints all current user’s contacts.

```
[jiri@cat gen_ha1]$ serctl ul add newuser sip:666@gateway.foo.bar
sip:666@gateway.foo.bar
200 Added to table
('newuser','sip:666@gateway.foo.bar') to 'location'
[jiri@cat gen_ha1]$ serctl ul show newuser
<sip:666@gateway.foo.bar>;q=1.00;expires=1073741812
[jiri@cat gen_ha1]$ serctl ul rm newuser
200 user (location, newuser) deleted
[jiri@cat gen_ha1]$ serctl ul show newuser
```

```
404 Username newuser in table location not found
```

3.2.2. User Aliases

Frequently, it is desirable for a user to have multiple addresses in a domain. For example, a user with username "john.doe" wants to be reachable at a shorter address "john" or at a nummerical address "12335", so that PSTN callers with digits-only key-pad can reach him too.

With ser, you can maintain a special user-location table and translate existing aliases to canonical usernames using the **lookup** action from usrloc module. The following script fragment demonstrates use of **lookup** for this purpose.

Example 3-7. Configuration of Use of Aliases

```
if (!uri==myself) { # request not for our domain...
  route(1); # go somewhere else, where outbound requests are processed
  break;
};
# the request is for our domain -- process registrations first
if (method=="REGISTER") { route(3); break; };

# look now, if there is an alias in the "aliases" table; don't care
# about return value: whether there is some or not, move ahead then
lookup("aliases");

# there may be aliases which translate to other domain and for which
# local processing is not appropriate; check again, if after the
# alias translation, the request is still for us
if (!uri==myself) { route(1); break; };

# continue with processing for our domain...
...
```

The table with aliases is updated using the serctl tool. serctl alias add <alias> <uri> adds a new alias, serctl alias show <user> prints an existing alias, and serctl alias rm <user> removes it.

```
[jiri@cat sip_router]$ serctl alias add 1234 sip:john.doe@foo.bar
sip:john.doe@foo.bar
200 Added to table
('1234','sip:john.doe@foo.bar') to 'aliases'
[jiri@cat sip_router]$ serctl alias add john sip:john.doe@foo.bar
sip:john.doe@foo.bar
200 Added to table
('john','sip:john.doe@foo.bar') to 'aliases'
[jiri@cat sip_router]$ serctl alias show john
<sip:john.doe@foo.bar>;q=1.00;expires=1073741811
[jiri@cat sip_router]$ serctl alias rm john
```

```
200 user (aliases, john) deleted
```

Note that persitence needs to be turned on in usrloc module. All changes to aliases will be otherwise lost on server reboot. To enable persistence, set the db_mode usrloc parameter to a non-zero value.

```
# ....load module ...
loadmodule "modules/usrloc/usrloc.so"
# ... turn on persistence -- all changes to user tables are immediately
# flushed to mysql
modparam("usrloc", "db_mode",   1)
# the SQL address:
modparam("usrloc", "db_url","sql://ser:secret@dbhost/ser")
```

3.2.3. Access Control (PSTN Gateway)

It is sometimes important to exercise some sort of access control. A typical use case is when ser is used to guard a PSTN gateway. If a gateway was not well guarded, unauthorized users would be able to use it to terminate calls in PSTN, and cause high charges to its operator.

There are few issues you need to understand when configuring ser for this purpose. First, if a gateway is built or configured to accept calls from anywhere, callers may easily bypass your access control server and communicate with the gateway directly. You then need to enforce at transport layer that signaling is only accepted if coming via ser and deny SIP packets coming from other hosts and port numbers. Your network must be configured not to allow forged IP addresses. Also, you need to turn on record-routing to assure that all session requests will travel via ser. Otherwise, caller's devices would send subsequent SIP requests directly to your gateway, which would fail because of transport filtering.

Authorization (i.e., the process of determining who may call where) is facilitated in ser using *group membership* concept. Scripts make decisions on whether a caller is authorized to make a call to a specific destination based on user's membership in a group. For example a policy may be set up to allow calls to international destinations only to users, who are members of an "int" group. Before user's group membership is checked, his identity must be verified first. Without cryptographic verification of user's identity, it would be impossible to assert that a caller really is who he claims to be.

The following script demonstrates, how to configure ser as an access control server for a PSTN gateway. The script verifies user identity using digest authentication, checks user's privileges, and forces all requests to visit the server.

Example 3-8. Script for Gateway Access Control

```
#
# $Id: pstn.cfg,v 1.2 2003/06/03 03:18:12 jiri Exp $
#
# example: ser configured as PSTN gateway guard; PSTN gateway is located
```

```
# at 192.168.0.10
#

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

loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"
loadmodule "modules/acc/acc.so"
loadmodule "modules/rr/rr.so"
loadmodule "modules/maxfwd/maxfwd.so"
loadmodule "modules/mysql/mysql.so"
loadmodule "modules/auth/auth.so"
loadmodule "modules/auth_db/auth_db.so"
loadmodule "modules/group/group.so"
loadmodule "modules/uri/uri.so"

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

modparam("auth_db", "db_url","sql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")

# -- acc params --
modparam("acc", "log_level", 1)
# that is the flag for which we will account -- don't forget to
# set the same one :-)
modparam("acc", "log_flag", 1 )

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

# main routing logic

route{

/* ********* ROUTINE CHECKS  ********************************** */

# filter too old messages
if (!mf_process_maxfwd_header("10")) {
log("LOG: Too many hops\n");
sl_send_reply("483","Too Many Hops");
break;
};
if (len_gt( max_len )) {
sl_send_reply("513", "Wow -- Message too large");
break;
};

/* ********* RR ********************************** */

/* grant Route routing if route headers present */
if (loose_route()) { t_relay(); break; };

/* record-route INVITEs -- all subsequent requests must visit us */
if (method=="INVITE") {
record_route();
};
```

```
# now check if it really is a PSTN destination which should be handled
# by our gateway; if not, and the request is an invitation, drop it --
# we cannot terminate it in PSTN; relay non-INVITE requests -- it may
# be for example BYEs sent by gateway to call originator
if (!uri=~"sip:\+?[0-9]+@.*") {
if (method=="INVITE") {
sl_send_reply("403", "Call cannot be served here");
} else {
forward(uri:host, uri:port);
};
break;
};

# account completed transactions via syslog
setflag(1);

# free call destinations ... no authentication needed
if ( is_user_in("Request-URI", "free-pstn")  /* free destinations */
|  uri=~"sip:[79][0-9][0-9][0-9]@.*"  /* local PBX */
| uri=~"sip:98[0-9][0-9][0-9][0-9]") {
log("free call");
} else if (src_ip==192.168.0.10) {
# our gateway doesn't support digest authentication;
# verify that a request is coming from it by source
# address
log("gateway-originated request");
} else {
# in all other cases, we need to check the request against
# access control lists; first of all, verify request
# originator's identity

if (!proxy_authorize( "gateway" /* realm */,
"subscriber" /* table name */))  {
proxy_challenge( "gateway" /* realm */, "0" /* no qop */ );
break;
};

# authorize only for INVITEs -- RR/Contact may result in weird
# things showing up in d-uri that would break our logic; our
# major concern is INVITE which causes PSTN costs

if (method=="INVITE") {

# does the authenticated user have a permission for local
# calls (destinations beginning with a single zero)?
# (i.e., is he in the "local" group?)
if (uri=~"sip:0[1-9][0-9]+@.*") {
if (!is_user_in("credentials", "local")) {
sl_send_reply("403", "No permission for local calls");
break;
};
# the same for long-distance (destinations begin with two zeros")
} else if (uri=~"sip:00[1-9][0-9]+@.*") {
if (!is_user_in("credentials", "ld")) {
sl_send_reply("403", " no permission for LD ");
break;
};
```

```
# the same for international calls (three zeros)
} else if (uri=~"sip:000[1-9][0-9]+@.*") {
if (!is_user_in("credentials", "int")) {
sl_send_reply("403", "International permissions needed");
break;
};
# everything else (e.g., interplanetary calls) is denied
} else {
sl_send_reply("403", "Forbidden");
break;
};

}; # INVITE to authorized PSTN

}; # authorized PSTN

# if you have passed through all the checks, let your call go to GW!

rewritehostport("192.168.0.10:5060");

# forward the request now
if (!t_relay()) {
sl_reply_error();
break;
};

}
```

Use the serctl tool to maintain group membership. serctl acl grant <username> <group> makes a user member of a group, serctl acl show <username> shows groups of which a user is member, and serctl acl revoke <username> [<group>] revokes user's membership in one or all groups.

```
[jiri@cat sip_router]$ serctl acl grant john int
MySql Password:
+------+-----+---------------------+
| user | grp | last_modified       |
+------+-----+---------------------+
| john | int | 2002-12-08 02:09:20 |
+------+-----+---------------------+
```

3.2.4. Accounting

In some scenarios, like termination of calls in PSTN, SIP administrators may wish to keep track of placed calls. ser can be configured to report on completed transactions. Reports are sent by default to syslog facility. Support for RADIUS and mysql accounting exists as well.

Note that ser is no way call-stateful. It reports on completed transactions, i.e., after a successful call set up is reported, it drops any call-related state. When a call is terminated, transactional state for BYE request is created and forgotten again after the transaction completes. This is a feature and not a bug -- keeping only transactional state allows for significantly higher scalability. It is then up to the accounting application to correlate call initiation and termination events.

To enable call accounting, tm and acc modules need to be loaded, requests need to be processed statefuly and labeled for accounting. That means, if you want a transaction to be reported, the initial request must have taken the path "**setflag(X)**, **t_relay**" in ser script. X must have the value configured in `acc_flag` configuration option.

Also note, that by default only transactions that initiate a SIP dialog (typically INVITE) visit a proxy server. Subsequent transactions are exhanged directly between end-devices, do not visit proxy server and cannot be reported. To be able to report on subsequent transactions, you need to force them visit proxy server by turning record-routing on.

Example 3-9. Configuration with Enabled Accounting

```
#
# $Id: acc.cfg,v 1.3 2003/06/03 03:18:12 jiri Exp $
#
# example: accounting calls to nummerical destinations
#

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

loadmodule "modules/tm/tm.so"
loadmodule "modules/acc/acc.so"
loadmodule "modules/sl/sl.so"
loadmodule "modules/maxfwd/maxfwd.so"
loadmodule "modules/rr/rr.so"

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

# -- acc params --
# set the reporting log level
modparam("acc", "log_level", 1)
# number of flag, which will be used for accounting; if a message is
# labeled with this flag, its completion status will be reported
modparam("acc", "log_flag", 1 )

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

# main routing logic

route{

/* ********* ROUTINE CHECKS  ********************************** */

# filter too old messages
if (!mf_process_maxfwd_header("10")) {
log("LOG: Too many hops\n");
sl_send_reply("483","Too Many Hops");
break;
```

```
};
if (len_gt( max_len )) {
sl_send_reply("513", "Wow -- Message too large");
break;
};

    #  Process record-routing
    if (loose_route()) { t_relay(); break; };


# labeled all transaction for accounting
setflag(1);

# record-route INVITES to make sure BYEs will visit our server too
if (method=="INVITE") record_route();

# forward the request statefuly now; (we need *stateful* forwarding,
# because the stateful mode correlates requests with replies and
# drops retranmissions; otherwise, we would have to report on
# every single message received)
if (!t_relay()) {
sl_reply_error();
break;
};

}
```

3.2.5. Reliability

It is essential to guarantee continuous service operation even under erroneous conditions, such as host or network failure. The major issue in such situations is transfer of operation to a backup infrastructure and making clients use it.

The SIP standard's use of DNS SRV records has been explicitly constructed to handle with server failures. There may be multiple servers responsible for a domain and referred to by DNS. If it is impossible to communicate with a primary server, a client can proceed to another one. Backup servers may be located in a different geographic area to minimize risk caused by areal operational disasters: lack of power, flooding, earthquake, etc.

> Unless there are redundant DNS servers, fail-over capability cannot be guaranteed.

Unfortunately, at the moment of writing this documentation (end of December 2002) only very few SIP products actually implement the DNS fail-over mechanism. Unless networks with SIP devices supporting this mechanism are built, alternative mechanisms must be used to force clients to use backup servers. Such a mechanism is disconnecting primary server and replacing it with a backup server locally. It unfortunately precludes geographic dispersion and requires network multihoming to avoid dependency on single IP access. Another method is to update DNS when failure of the primary server is detected. The primary drawback of this method is its latency: it may take long time until all clients learn to use the new server.

The easier part of the redundancy story is replication of ser data. ser relies on replication capabilities of its back-end database. This works with one exception: user location database. User location database is a frequently accessed table, which is thus cached in server's memory to improve performance. Back-end replication does not affect in-memory tables, unless server reboots. To facilitate replication of user location database, server's SIP replication feature must be enabled in parallel with back-end replication.

The design idea of replication of user location database is easy: Replicate any successful REGISTER requests to a peer server. To assure that digest credentials can be properly verified, both servers need to use the same digest generation secret and maintain synchronized time. A known limitation of this method is it does not replicate user contacts entered in another way, for example using web interface through FIFO server. The following script example shows configuration of a server that replicates all REGISTERs.

Example 3-10. Script for Replication of User Contacts

```
#
# $Id: replicate.cfg,v 1.2 2003/06/03 03:18:12 jiri Exp $
#
# demo script showing how to set-up usrloc replication
#

# ----------- global configuration parameters ------------------------

debug=3          # debug level (cmd line: -dddddddddd)
fork=no
log_stderror=yes # (cmd line: -E)

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

loadmodule "modules/mysql/mysql.so"
loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"
loadmodule "modules/maxfwd/maxfwd.so"
loadmodule "modules/usrloc/usrloc.so"
loadmodule "modules/registrar/registrar.so"
loadmodule "modules/auth/auth.so"
loadmodule "modules/auth_db/auth_db.so"

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

# digest generation secret; use the same in backup server;
# also, make sure that the backup server has sync'ed time
modparam("auth", "secret", "alsdkhglaksdhfkloiwr")

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

# main routing logic

route{

# initial sanity checks -- messages with
# max_forwars==0, or excessively long requests
if (!mf_process_maxfwd_header("10")) {
sl_send_reply("483","Too Many Hops");
break;
};
if (len_gt( max_len )) {
```

```
sl_send_reply("513", "Message too big");
break;
};

# if the request is for other domain use UsrLoc
# (in case, it does not work, use the following command
# with proper names and addresses in it)
if (uri==myself) {

if (method=="REGISTER") {

# verify credentials
if (!www_authorize("foo.bar", "subscriber")) {
www_challenge("foo.bar", "0");
break;
};

# if ok, update contacts and ...
save("location");
# ... if this REGISTER is not a replica from our
# peer server, replicate to the peer server
if (!src_ip==backup.foo.bar) {
t_replicate("backup.foo.bar", "5060");
};
break;
};
# do whatever else appropriate for your domain
log("non-REGISTER\n");
};
}
```

3.2.6. Stateful versus Stateless Forwarding

ser allows both stateless and stateful request processing. This memo explains what are pros and cons of using each method. The rule of thumb is "stateless for scalability, stateful for services". If you are unsure which you need, stateful is a safer choice which supports more usage scenarios.

Stateless forwarding with the **forward(uri:host, uri:port)** action guarantees high scalability. It withstands high load and does not run out of memory. A perfect use of stateless forwarding is load distribution.

Stateful forwarding using the **t_relay()** action is known to scale worse. It can quickly run out of memory and consumes more CPU time. Nevertheless, there are scenarios which are not implementable without stateful processing. In particular:

- *Accounting* requires stateful processing to be able to collect transaction status and issue a single report when a transaction completes.
- *Forking* only works with stateful forwarding. Stateless forwarding only forwards to the default URI out of the whole destination set.

- *DNS resolution*. DNS resolution may be better served with stateful processing. If a request is forwarded to a destination whose address takes long time to resolve, a server process is blocked and unresponsive. Subsequent request retransmissions from client will cause other processes to block too if requests are processed statelessly. As a result, ser will quickly run out of available processes. With stateful forwarding, retransmissions are absorbed and do not cause blocking of another process.
- *Forwarding Services*. All sort of services with the "forward_on_event" logic, which rely on **t_on_failure** tm action must be processed statefuly.
- *Fail-over*. If you wish to try out another destination, after a primary destination failed you need to use stateful processing. With stateless processing you never know with what status a forwarded request completed downstream because you immediately release all processing information after the request is sent out.

> **Note:** Positive return value of stateless **forward** action only indicates that a request was successfuly sent out, and does not gain any knowledge about whether it was successfuly received or replied. Neither does the return value of the stateful **t_relay** action family gain you this knowledge. However, these actions store transactional context with which includes original request and allows you to take an action when a negative reply comes back or a timer strikes. See Section 2.7.5 for an example script which launches another branch if the first try fails.

3.2.7. Serving Multiple Domains

ser can be configured to serve multiple domains. To do so, you need to take the following steps:

1. Create separate subscriber and location database table for each domain served and name them uniquely.
2. Configure your script to distinguish between multiple served domains. Use regular expressions for domain matching as described in Example 2-6.
3. Update table names in usrloc and auth actions to reflect names you created in 1.

The latest SER release includes automated multidomain management which greatly automates maintenance of multiple domains. Ask our technical support for more help.

3.2.8. Reporting Missed Calls

ser can report missed calls via syslog facility or to mysql. Mysql reporting can be utilized by ser's complementary web-interface, serweb. (See more in Section 5.2).

Reporting on missed calls is enabled by acc module. There are two cases, on which you want to report. The first case is when a callee is off-line. The other case is when a user is on-line, but call establishment fails. There may be many failure reasons (call cancellation, inactive phone, busy phone, server timer, etc.), all of them leading to a negative (>=300) reply sent to caller. The acc module can be configured to issue a missed-call report whenever a transaction completes with a negative status. Two following script fragment deals with both cases.

First, it reports on calls missed due to off-line callee status using the **acc_request** action. The action is wrapped in transactional processing (**t_newtran**) to guarantee that reports are not duplicated on receipt of retransmissions.

Secondly, transactions to on-line users are marked to be reported on failure. That is what the **setflag(3)** action is responsible for, along with the configuration option "log_missed_flag". This option configures ser to report on all transactions, which were marked with flag 3.

```
loadmodule("modules/tm/tm.so");
loadmodule("modules/acc/acc.so");
....
# if a call is labeled using setflag(3) and is missed, it will
# be reported
...
modparam("acc", "log_missed_flag", 3 );
if (!lookup("location")) {
     # call invitations to off-line users are reported using the
     # acc_request action; to avoid duplicate reports on request
     # retransmissions, request is processed statefuly (t_newtran,
     # t_reply)
     if ((method=="INVITE" || method=="ACK") && t_newtran() ) {
          t_reply("404", "Not Found");
  acc_request("404 Not Found");
          break;
     };
     # all other requests to off-line users are simply replied
     # statelessly and no reports are issued
    sl_send_reply("404", "Not Found");
    break;
} else {
     # user on-line; report on failed transactions; mark the
     # transaction for reporting using the same number as
     # configured above; if the call is really missed, a report
     # will be issued
     setflag(3);
     # forward to user's current destination
     t_relay();
     break;
};
```

3.2.9. NAT Traversal

NATs are worst things that ever happened to SIP. These devices are very popular because they help to conserve IP address space and save money charged for IP addresses. Unfortunately, they translate addresses in a way which is not compatible with SIP. SIP advertises receiver addresses in its payload. The advertised addresses are invalid out of NATted networks. As a result, SIP communication does not work accross NATs without extra effort.

There are few methods that may be deployed to traverse NATs. How proper their use is depends on the deployment scenario. Unfortunatelly, all the methods have some limitations and there is no straight-forward solution addressing all scenarios. Note that none of these methods takes explicit support in ser.

The first issue is whether SIP users are in control of their NATs. If not (NATs are either operated by ISP or they are sealed to prevent users setting them up), the only method is use of a STUN-enabled phone. STUN is a very simple protocol used to fool NAT in such a way, they permit SIP sessions. Currently, we are aware of one softphone (kphone) and one hardphone (snom) with STUN support, other vendors are working on STUN support too. Unfortunately, STUN gives no NAT traversal guarantee -- there are types of NATs, so called symmetric NATs, over which STUN fails to work.

> **Note:** There is actually yet another method to address SIP-unaware, user-uncontrolled NATs. It is based on a proxy server, which relays all signaling and media and mangles packets to make them more NAT-friendly. The very serious problem with this method is it does not scale.

If users are in control of their own NAT, as typically residential users are, they can still use STUN. However, they may use other alternatives too. One of them is to replace their NAT with a SIP-aware NAT. Such NATs have built-in SIP awareness, that patches problems caused by address translations. Prices of such devices are getting low and there are available implementations (Intertex, Cisco/PIX). No special support in phones is needed.

Other emerging option is UPnP. UPnP is a protocol that allows phones to negotiate with NAT boxes. You need UPnP support in both, NAT and phones. As UPnP NATs are quite affordable, costs are not an obstacle. Currently, we are aware of one SIP phone (SNOM) with UPnP support.

Geeks not wishing to upgrade their firewall to a SIP-aware or UPnP-enabled one may try to configure static address translation. That takes phones with configuration ability to use fixed port numbers and advertise outside address in signaling. Cisco phones have this capability, for example. The NAT devices need to be configured to translate outside port ranges to the ranges configured in phones.

3.2.10. Using Only Latest User's Contact for Forwarding

In some scenarios, it may be beneficial only to use only one registered contact per user. If that is the case, setting registar module's parameter `append_branches` to 1 will eliminate forking and forward all requests only to a single contact. If there are multiple contacts, a contact with highest priority is chosen. This can be changed to the "freshest" contact by setting module parameter's `desc_time_order` to 1.

3.2.11. Authentication Policy: Prevention of Unauthorized Domain Name Use in From and More

Malicous users can claim a name of domain, to which they do not administratively belong, in From header field. This behaviour cannot be generally prevented. The reason is that requests with such a faked header field do not need to visit servers of the domain in question. However, if they do so, it is desirable to assure that users claiming membership in a domain are actually associated with it. Otherwise the faked requests would be relayed and appear as coming from the domain, which would increase credibility of the faked address and decrease credibility of the proxy server.

Preventing unathorized domain name use in relayed requests is not difficult. One needs to authenticate each request with name of the served domain in From header field. To do so, one can search for such a header field using **search** action (textops module) and force authentication if the search succeeds.

> **Note:** A straight-forward solution might be to authenticate ALL requests. However, that only works in closed networks in which all users have an account in the server domain. In open networks, it is desirable to permit incoming calls from callers from other domains without any authentication. For example, a company may wish to accept calls from unknown callers who are new prospective customers.

```
# does the user claim our domain "foo.bar" in From?
if (search("^(f|From):.*foo.bar")) {
        # if so, verify credential
if (!proxy_authorize("foo.bar", "subscriber")) {
              # don't proceed if credentials broken; challenge
      proxy_challenge("foo.bar", "0");
      break;
        };
};
```

In general, the authentication policy may be very rich. You may not forget each request deserves its own security and you need to decide whether it shall be authenticated or not. As mentioned above, in closed networks, you may want to authenticate absolutely every request. That however prohibits traffic from users from other domains. A pseudo-example of a reasonable policy is attached: it looks whether a request is registration, it claims to originate from our domain in From header field, or is a local request to another domain.

```
# (example provided by Michael Graff on [serusers] mailing list
if (to me):
    if register
          www_authorize or fail if not a valid register
          done
    if claiming to be "From" one of the domains I accept registrations for
          proxy_authorize
          done
    if not to me (I'm relaying for a local phone to an external address)
          proxy_authorize
          done
```

You also may want to apply additional restriction to how digest username relates to usernames claimed in From and To header fields. For example, the **check_to** action enforces the digest id to be equal to username in To header fields. That is good in preventing someone with valid credentials to register as someone else (e.g., sending a REGISTER with valid credentials of "joe" and To belonging to "alice"). Similarly, **check_from** is used to enforce username in from to equal to digest id.

> **Note:** There may be a need for a more complex relationship between From/To username and digest id. For example, providers with an established user/password database may wish to keep using it, whereas permitting users to claim some telephone numbers in From. To address such needs generally, there needs to be a 1:N mapping between digest id and all usernames that are acceptable for it. This is being addressed in

a newly contributed module "domain", which also addresses more generally issues of domain matching for multidomain scenarios.

Other operational aspect affecting the authentication policy is guarding PSTN gateways (see Section 3.2.3). There may be destinations that are given away for free whereas other destinations may require access control using group membership, to which authentication is a prerequisity.

3.2.12. Connecting to PBX Voicemail Using a Cisco Gateway

In some networks, administrators may wish to utilize their PBX voicemail systems behind PSTN gateways. There is a practical problem in many network settings: it is not clear for whom a call to voicemail is. If voicemail is identified by a single number, which is then put in INVITE's URI, there is no easy way to learn for whom a message should be recorded. PBX voicemails utilize that PSTN protocols signal the number of originally called party. If you wish to make the PBX voicemail work, you need to convey the number in SIP and translate it in PSTN gateways to its PSTN counterpart.

There may be many different ways to achieve this scenario. Here we describe the proprietary mechanism Cisco gateways use and how to configure ser to make the gateways happy. Cisco gateways expect the number of originally called party to be located in proprietary `CC-Diversion` header field. When a SIP INVITE sent via a PSTN gateway to PBX voicemail has number of originally called party in the header field, the voicemail system knows for whom the incoming message is. That is at least true for AS5300/2600 with Cisco IOS 12.2.(2)XB connected to Nortel pbxs via PRI. (On the other hand, 12.2.(7b) is known not to work in this scenario.)

ser needs then to be configured to append the `CC-Diversion` header field name for INVITEs sent to PBX voicemail. The following script shows that: when initial forwarding fails (nobody replies, busy is received, etc.), a new branch is initiated to the pbx's phone number. **append_urihf** is used to append the `CC-Diversion` header field. It takes two parameters: prefix, which includes header name, and suffix which takes header field separator. **append_urihf** inserts original URI between those two.

Example 3-11. Forwarding to PBX/Voicemail via Cisco Gateways

```
# $Id: ccdiversion.cfg,v 1.2 2003/06/03 03:18:12 jiri Exp $

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

loadmodule "modules/textops/textops.so"
loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"

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

route{
# if we do not get a positive reply, continue at reply_route[2]
t_on_failure("2");
# forward the request to all destinations in destination set now
t_relay();
}
```

```
failure_route[2] {
# request failed (no reply, busy, whatever) ... forward it again
# to pbx's voicemail at phone number 6000 via Cisco gateway at
# 192.168.10.10; append proprietary CC-Diversion header field with
# original request uri in it, so that the gateway and voicemail
# know, whom the request was originally intended for
append_branch("sip:6000@192.168.10.10");
append_urihf("CC-Diversion: ", "\r\n");
t_relay();
}
```

3.3. Troubleshooting

This section gathers practices how to deal with errors known to occur frequently. To understand how to watch SIP messages, server logs, and in genereal how to troubleshoot, read also Section 3.1.

1. SIP requests are replied by ser with "483 Too Many Hops" or "513 Message Too Large"

In both cases, the reason is probably an error in request routing script which caused an infinite loop. You can easily verify whether this happens by watching SIP traffic on loopback interface.

The most comon reason for misrouting is a failure to match local domain correctly. If a server fails to recognize a request for itself, it will try to forward it to current URI in believe it would forward them to a foreign domain. Alas, it forwards the request to itself again. This continues to happen until value of max_forwards header field reaches zero or the request grows too big. Solutions is easy: make sure that domain matching is correctly configured. In easy case, it is as simple as adding an alias configuration option. See Section 2.2.2.1 for more information how to get it right.

Other favorite misconfiguration is forgetting to rewrite a URI before a request is forwarded.

Also, the failure not to include loose routing in your scripts may lead to infinite loops. Make sure that you include the following script fragment immediately after request sanity checks:

Example 3-12. Processing of Loose Routes Must be Present

```
if (len_gt( max_len )) {
    sl_send_reply("513", "Message too big");
    break;
};

# DON'T FORGET THE FOLLOWING LINES
if (loose_route()) {
    t_relay();
    break;
};
```

2. Windows Messenger authentication fails.

The most likely reason for this problem is a bug in Windows Messenger. WM only authenticates if server name in request URI equals authentication realm. After a challenge is sent by SIP server, WM does not resubmit the challenged request at all and pops up authentication window again. If you want to authenticate WM, you need to set up your realm value to equal server name. If your server has no name, IP address can be used as realm too. The realm value is configured in scripts as the first parameter of all **{www|proxy}_{authorize|challenge}** actions.

3. Windows Messenger Reponds with "400 Bad Request".

That is most likely caused by a Messenger bug: it does not like ";lr" record-routing parameter. Turn on the option 'modparam("rr", "enable_full_lr", 1)' to use ";lr=true" instead -- Messenger will then process record routing.

4. Multiple phones register with a single address of record. The server complains during processing of incoming requests for this address: "ERROR: append_branch: max nr of branches exceeded".

There is a compile-time limitation on number of contacts that may be used for forking. To change the limit, update value of MAX_BRANCHES in config.h and recompile SER.

5. On a multihomed host, forwarded messages carry other interface in Via than used for sending, or messages are not sent and an error log is issued "invalid sendtoparameters one possible reason is the server is bound to localhost".

Set the configuration option `mhomed` to "1". ser will then attempt to calculate the correct interface. It's not done by default as it degrades performance on single-homed hosts or multi-homed hosts that are not set-up as routers.

6. I receive "ERROR: t_newtran: transaction already in process" in my logs.

That looks like an erroneous use of tm module in script. tm can handle only one transaction per request. If you attempt to instantiate a transaction multiple times, ser will complain. Anytime any of **t_newtran**, **t_relay** or **t_relay_to_udp** actions is encountered, tm attempts to instantiate a transaction. Doing so twice fails. Make sure that any of this commands is called only once during script execution.

7. I try to add an alias but **serctl** complains that table does not exist.

You need to run ser and use the command **lookup("aliases")** in its routing script. That's because the table of aliases is stored in cache memory for high speed. The cache memory is only set up when the ser is running and configured to use it. If that is not the case, serctl is not able to manipulate the aliases table.

8. I started ser with `children=4` but many more processes were started. What is wrong?

That's ok. The `children` parameter defines how many children should process each transport protocol in parallel. Typically, the server listens to multiple protocols and starts other supporting processes like timer or FIFO server too. Call serctl ps to watch running processes.

9. I decided to use a compiled version of ser but it does not start any more.

You probably kept the same configuration file, which tries to load modules from the binary distribution you used previously. Make sure that modules paths are valid and point to where you compiled ser. Also, watch logs for error messages "ERROR: load_module: could not open module".

Chapter 4. Application Writing

ser offers several ways to couple its functionality with applications. The coupling is bidirectional: ser can utilize external applications and external applications can utilize ser. An example of the former direction would be an external program determining a least-cost route for a called destination using a pricing table. An example of the latter case is a web application for server provisioning. Such an application may want to send instant messages, query all current user's locations and monitor server health. An existing web interface to ser, serweb, actually does all of it.

The easiest, language-independent way of using external logic from ser is provided by exec module. exec module allows ser to start external programs on receipt of a request. The programs can execute arbitrary logic and/or affect routing of SIP requests. A great benefit of this programming method is it is language-independent. Programmers may use programming languages that are effective or with which they are best familiar. Section 4.1 gives additional examples illustrating use of the exec module.

Another method for extending ser capabilities is to write new modules in C. This method takes deeper understanding of ser internals but gains the highest flexibility. Modules can implement arbitrary brand-new commands upon which ser scripts can rely on. Guidelines on module programming can be found in ser programmer's handbook available from iptel.org website.

To address needs of applications wishing to leverage ser, ser exports parts of its functionality via its built-in "Application FIFO server". This is a simple textual interface that allows any external applications to communicate with the server. It can be used to send instant messages, manipulate user contacts, watch server health, etc. Programs written in any language (PHP, shell scripts, Perl, C, etc.) can utilize this feature. How to use it is shown in Section 4.2.

4.1. Using exec Module

The easiest way is to couple ser with external applications via the *exec* module. This module allows execution of logic and URI manipulation by external applications on request receipt. While very simple, many useful services can be implemented this way. External applications can be written in any programming language and do not be aware of SIP at all. ser interacts with the application via standard input/output and environment variables.

For example, an external shell script may send an email whenever a request for a user arrives.

Example 4-1. Using exec: Step 1

```
# send email if a request for user "jiri" arrives
if (uri=~"^sip:jiri@") {
     exec_msg("echo 'body: call arrived'|mail -s 'call for you' jiri");
}
```

In this example, the **exec_msg** action starts an external shell. It passes a received SIP request to shell's input. In the shell, **mail** command is called to send a notification by e-mail. The script however features several simplifications:

1. The email notification does not tell who was calling.
2. The logic is not general: it only supports one well-known user (jiri).
3. The logic is stateless. It will be executed on every retransmission.
4. It is a script fragment not explaining the context. This particular example may be for example used to report on missed calls.

All of these simplifications are addressed step-by-step in the following examples.

Example 4-2. Using exec: Step 2, Who Called Me

This example shows how to display caller's address in email notification. The trick is easy: process request received on shell programm's input and grep From header field.

```
#
# $Id: exec_s2.cfg,v 1.1 2003/01/05 20:37:51 jiri Exp $
#
# send a notificiation if a request for user jiri arrives
# ------------------ module loading ----------------------------------

loadmodule "modules/exec/exec.so"
loadmodule "modules/sl/sl.so"

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

# main routing logic

route{
# send email if a request for jiri arrives
if (uri=~"^sip:jiri@") {
exec_msg("(egrep -i '^(From|f):';
echo 'request received')|
mail -s 'request for you' jiri");
};

}
```

The following two figures show an example SIP request and email notification generated on its receipt.

```
INVITE sip:jiri@iptel.org SIP/2.0
Via: SIP/2.0/UDP 195.37.77.100:5040
Max-Forwards: 10
From: "alice" <sip:alice@iptel.org>;tag=76ff7a07-c091-4192-84a0-d56e91fe104f
To: <sip:jiri@iptel.org>
Call-ID: d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35
CSeq: 2 INVITE
Contact: <sip:123.20.128.35:9315>
Content-Type: application/sdp
Content-Length: 451

--- SDP payload snipped ---
```

email received:

```
Date: Thu, 12 Dec 2002 14:25:02 +0100
```

```
From: root <root@cat.iptel.org>
To: jiri@cat.iptel.org
Subject: request for you

From: "alice" <sip:alice@iptel.org>;tag=76ff7a07-c091-4192-84a0-d56e91fe104f
request received
```

There is another way to learn values of request header fields, simpler than use of **grep**. ser parses header fields and passes their values in environment variables. Their names correspond to header field names prefixed with "SIP_HF_".

```
# send email if a request for "jiri" arrives
if (uri=~"^sip:jiri@") {
     exec_msg("echo request received from $SIP_HF_FROM | mail -s 'request for you' jiri");
};
```

Moreover, several other values are passed in environment variables. `SIP_TID` is a token uniquely identifying transaction, to which the request belongs. `SIP_DID` includes to-tag, and is empty in requests creating a dialog. `SIP_SRCIP` includes IP address, from which the request was sent. `SIP_RURI` and `SIP_ORURI` include current request-uri and original request-uri respectively, `SIP_USER` and `SIP_OUSER` username parts of these. The following listing shows environment variables passed to a shell script on receipt of the previous message:

```
SIP_HF_MAX_FORWARDS=10
SIP_HF_VIA=SIP/2.0/UDP 195.37.77.100:5040
SIP_HF_CSEQ=2 INVITE
SIP_HF_FROM="alice" <sip:alice@iptel.org>;tag=76ff7a07-c091-4192-84a0-d56e91fe104f
SIP_ORUI=sip:jiri@iptel.org
SIP_HF_CONTENT_LENGTH=451
SIP_TID=3b6b8295db0835815847b1f35f3b29b8
SIP_DID=
SIP_RURI=iptel.org
SIP_HF_TO=<sip:jiri@iptel.org>
SIP_OUSER=jiri
SIP_HF_CALLID=d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35
SIP_SRCIP=195.37.77.100
SIP_HF_CONTENT_TYPE=application/sdp
SIP_HF_CONTACT=<sip:123.20.128.35:9315>
```

Example 4-3. Using exec: step 3, Make The Script Work For Anyone

A drawback of the previous example is it works only for one well-known user: request URI is matched against his SIP address and notification is sent to his hard-wired email address. In real scenarios, one would like to enable such a service for all users without enumerating their addresses in the script. The missing piece is translation of user's SIP name to his email address. This information is maintained in subscriber profiles, stored in MySQL by ser. To translate the username to email address, the executed script needs to query the MySQL database. That is what this example shows. First, an SQL query is constructed which looks up email address of user, for whom a request arrived. If the query does not return a valid email address, the script returns with an error status and ser script replies with "user does not exist". Otherwise an email notification is sent.

```
#
# $Id: exec_s3.cfg,v 1.2 2003/01/25 15:04:43 janakj Exp $
#
# email notification to email address from mysql database
```

```
#

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

loadmodule "modules/exec/exec.so"
loadmodule "modules/sl/sl.so"

# send email if a request arrives
route[0] {
     if (!exec_msg('
QUERY="select email_address from subscriber
where user=\"$SIP_OUSER\"";
EMAIL=`mysql  -Bsuser -pheslo -e "$QUERY" ser`;
if [ -z "$EMAIL" ] ; then exit 1; fi ;
echo "SIP request received from $SIP_HF_FROM for $SIP_OUSER" |
mail -s "request for you" $EMAIL ')) {
# exec returned error ... user does not exist
sl_send_reply("404", "User does not exist");
} else {
sl_send_reply("600", "No messages for this user");
};

}
```

Example 4-4. Adding Stateful Processing

The previously improved example still features a shortcoming. When a message retransmission arrives due to a nework mistake such as lost reply, the email notification is executed again and again. That happens because the script is stateless, i.e., no track of current transactions is kept. The script does not know whether a request is a new or a retransmitted one. Transaction management may be introduced by use of tm module as described in Section 2.7.2. In the script, **t_newtran** is first called to absorb requests retransmission -- if they occur, script does not continue. Then, as in the previous example, an exec module action is called. Eventually, a reply is sent statefully.

> **Note:** Note carefuly: it is important that the stateful reply processing (**t_reply**) is used as opposed to using stateless replies (**sl_send_reply**). Otherwise, the outgoing reply would not affect transactional context and would not be resent on receipt of a request retransmission.

```
#
# $Id: exec_s4.cfg,v 1.2 2003/01/25 15:04:43 janakj Exp $
#
# email notification to email address from mysql database
#

fork=no

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

loadmodule "modules/exec/exec.so"
loadmodule "modules/sl/sl.so"
```

```
loadmodule "modules/tm/tm.so"

# send email if a request arrives; process statefully
# to avoid multiple execution on request retransmissions
route[0] {
# stop script processing if transaction exists
if ( !t_newtran()) {
sl_reply_error();
break;
};

      if (!exec_msg('
QUERY="select email_address from subscriber
where user=\"$SIP_OUSER\"";
EMAIL=`mysql  -Bsuser -pheslo -e "$QUERY" ser`;
if [ -z "$EMAIL" ] ; then exit 1; fi ;
echo "SIP request received from $SIP_HF_FROM for $SIP_OUSER" |
mail -s "request for you" $EMAIL '))
{
# exec returned error ... user does not exist
# send a stateful reply
t_reply("404", "User does not exist");
} else {
t_reply("600", "No messages for this user");
};

}
```

Example 4-5. Full Example of exec Use

The last example iteration shows how to integrate the email notification on missed calls with the default ser script (see Section 2.7.1). It generates an email for every call invitation to an off-line user.

```
#
# $Id: exec_s5.cfg,v 1.2 2003/01/25 15:04:43 janakj Exp $
#
# simple quick-start config script
#
fork=no
log_stderror=yes
# ----------- global configuration parameters ------------------------

loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"
loadmodule "modules/usrloc/usrloc.so"
loadmodule "modules/registrar/registrar.so"
loadmodule "modules/exec/exec.so"

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

route{
# uri for my domain ?
```

```
if (uri==myself) {

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

# native SIP destinations are handled using our USRLOC DB
if (!lookup("location")) {
# proceed to email notification
if (method=="INVITE") route(1)
else sl_send_reply("404", "Not Found");
break;
};
};
# user found, forward to his current uri now
if (!t_relay()) {
sl_reply_error();
};
}

/* handling of missed calls */
route[1] {
# don't continue if it is a retransmission
if ( !t_newtran()) {
sl_reply_error();
break;
};
# external script: lookup user, if user exists, send
# an email notification to him
      if (!exec_msg('
QUERY="select email_address from subscriber
where user=\"$SIP_OUSER\"";
EMAIL=`mysql  -Bsuser -pheslo -e "$QUERY" ser`;
if [ -z "$EMAIL" ] ; then exit 1; fi ;
echo "SIP request received from $SIP_HF_FROM for $SIP_OUSER" |
mail -s "request for you" $EMAIL '))
{
# exec returned error ... user does not exist
# send a stateful reply
t_reply("404", "User does not exist");
} else {
t_reply("600", "No messages for this user");
};
break;
}
```

Production "missed calls" services may want to report on calls missed for other reasons than being off-line too. Particularly, users may wish to be reported calls missed due to call cancellation, busy status or a downstream failure. Such missed calls can be easily reported to syslog or mysql using the acc module (see Section 3.2.8). The other, more general way, is to return to request processing on receipt of a negative reply. (see Section 2.7.5). Before a request is forwarded, it is labeled to be re-processed in a **failure_route** on receipt of a negative reply -- this is what **t_on_failure** action is used for. It does not matter what caused the transaction to fail -- it may be unresponsive downstream server, server responding with 6xx, or server sending a 487 reply, because an INVITE was cancelled. When any such circumstances occur (i.e., transaction does not complete with a 2xx status code), **failure_route** is entered.

The following ser script reports missed calls in all possible cases. It reports them when a user is off-line as well as when a user is on-line, but INVITE transaction does not complete successfully.

```
#
# $Id: exec_s5b.cfg,v 1.5 2003/06/20 23:20:35 jiri Exp $
#
# simple quick-start config script
#
fork=no
log_stderror=yes
# ----------- global configuration parameters ------------------------

loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"
loadmodule "modules/usrloc/usrloc.so"
loadmodule "modules/registrar/registrar.so"
loadmodule "modules/exec/exec.so"

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

route{
# uri for my domain ?
if (uri==myself) {

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

# native SIP destinations are handled using our USRLOC DB
if (!lookup("location")) {
# proceed to email notification
if (method=="INVITE") route(1)
else sl_send_reply("404", "Not Found");
break;
};
};
# user found, forward to his current uri now; if any
# forwarding error occurs (e.g., busy or cancelled recevied
# from downstream), proceed to reply_route[1]
t_on_failure("1");
if (!t_relay()) {
sl_reply_error();
};
}

/* handling of missed calls */
route[1] {
# don't continue if it is a retransmission
if ( !t_newtran()) {
sl_reply_error();
break;
};
# external script: lookup user, if user exists, send
```

```
# an email notification to him
      if (!exec_msg('
QUERY="select email_address from subscriber
where username=\"$SIP_OUSER\"";
EMAIL=`mysql  -Bsuser -pheslo -e "$QUERY" ser`;
if [ -z "$EMAIL" ] ; then exit 1; fi ;
echo "SIP request received from $SIP_HF_FROM for $SIP_OUSER" |
mail -s "request for you" $EMAIL '))
{
# exec returned error ... user does not exist
# send a stateful reply
t_reply("404", "User does not exist");
} else {
t_reply("600", "No messages for this user");
};
break;
}

failure_route[1] {
# just call exec, that's it
    exec_msg('
QUERY="select email_address from subscriber
where username=\"$SIP_OUSER\"";
EMAIL=`mysql  -Bsuser -pheslo -e "$QUERY" ser`;
if [ -z "$EMAIL" ] ; then exit 1; fi ;
echo "SIP request received from $SIP_HF_FROM for $SIP_OUSER" |
mail -s "request for you" $EMAIL ') ;
t_relay();
}
```

4.2. Application FIFO Server

Application FIFO server is a very powerful method to program SIP services. The most valuable benefit is it works with SIP-unaware applications written in any programming language. Textual nature of the FIFO interface allows for easy integration with a lot of existing programs. Today, ser's complementary web-interface, serweb, written in PHP, leverages the FIFO interface when displaying and changing user location records stored in server's memory. It uses this interface to send instant messages too, without any knowledge of underlying SIP stack. Another application relying on the FIFO interface is serctl, ser management utility. The command-line utility can browse server's in-memory user-location database, display running processes and operational statistics.

The way the FIFO server works is similar to how `/proc` filesystem works on some operating systems. It provides a human-readable way to access ser's internals. Applications dump their requests into the FIFO server and receive a status report when request processing completes. ser exports a lot of its functionality located in both the core and external modules through the FIFO server.

FIFO requests are formed easily. They begin with a command enclosed in colons and followed by name of file or pipe (relative to `/tmp/` path), to which a reply should be printed. The first request line may be followed by additional lines with command-specific parameters. For example, the **t_uac_dlg** FIFO command for initiating a transaction allows to pass additional header fields and message body to a newly created transaction. Each request is terminated by an empty line. Whole requests must be sent by applications atomically in a single batch to avoid mixing with requests from other applications. Requests are sent to pipe at which ser listens (filename configured by the `fifo` config file option).

An easy way to use the FIFO interface is via the serctl command-line tool. When called along with "fifo", FIFO command name, and optional parameters, the tool generates a FIFO request and prints request result. The following example shows use of this tool with the **uptime** and **which** commands. **uptime** returns server's running time, **which** returns list of available FIFO commands. Note that only the built-in FIFO command set is displayed as no modules were loaded in this example.

Example 4-6. Use of serctl to Access FIFO Server

```
[jiri@cat test]$ serctl fifo uptime
Now: Fri Dec  6 17:56:10 2002
Up Since: Fri Dec  6 17:56:07 2002
Up time: 3 [sec]

[jiri@cat test]$ serctl fifo which
ps
which
version
uptime
print
```

The request which the serctl command-line tool sent to FIFO server looked like this:

Example 4-7. uptime FIFO Request

```
:uptime:ser_receiver_1114
```

This request contains no parameters and consists only of command name enclosed in colons and name of file, to which a reply should be printed. FIFO replies consist of a status line followed by optional parameters. The status line consists, similarly to SIP reply status, of a three-digit status code and a reason phrase. Status codes with leading digit 2 (200..299) are considered positive, any other values indicate an error. For example, FIFO server returns "500" if execution of a non-existing FIFO command is requested.

Example 4-8. FIFO Errors

```
[jiri@cat sip_router]$ serctl fifo foobar
500 command 'foobar' not available
```

Example 4-9. Showing User Contacts Using serctl

Another example of use of FIFO is accessing server's in-memory user location database. That's a very powerful feature: web applications and other tools can use it to gain users access to the database. They can add new contacts (like permanent gateway destinations), remove and review users' whereabouts. The example here utilizes FIFO command **ul_show_contact** to retrieve current whereabouts of user "jiri".

```
[jiri@fox ser]$ serctl fifo ul_show_contact location jiri
<sip:195.37.78.160:14519>;q=0.00;expires=1012
```

The user location example demonstrates an essential feature of the FIFO server: extensibility. It is able to export new commands implemented in new modules. Currently, usrloc module exports FIFO commands for maintaining in-memory user location database and tm module exports FIFO commands for management of SIP transactions. See the example in `examples/web_im/send_im.php` for how to initiate a SIP transaction (instant message) from a PHP script via the FIFO server. This example uses FIFO command **t_uac_dlg**. The command is followed by parameters: header fields and message body. The same FIFO command can be used from other environments to send instant messages too. The following example shows how to send instant messages from a shell script.

Example 4-10. Sending IM From Shell Script

```
#!/bin/sh
#
# call this script to send an instant message; script parameters
# will be displayed in message body
#
# paremeters mean: message type, request-URI, outbound server is
# left blank ("."), required header fields From and To follow,
# then optional header fields terminated by dot and optional
# dot-terminated body

cat > /tmp/ser_fifo <<EOF
:t_uac_dlg:hh
NOTIFY
sip:receiver@127.0.0.1
.
From: sip:originator@foo.bar
To: sip:receiver@127.0.0.1
foo: bar_special_header
x: y
p_header: p_value
Contact: <sip:devnull@192.168.0.100:9>
Content-Type: text/plain; charset=UTF-8
.
Hello world!!!! $@
.
EOF
```

Example 4-11. Manipulation of User Contacts

The following example shows use of FIFO server to change user's contacts. This may be very practical, if for example a user wishes to set up his cell phone number as his temporary contact. The cell phone, which is behind a PSTN gateway, cannot register automatically using SIP. The user needs to set forwarding manually through some convenient web interface. The web interface needs to have the ability to upload new user's contacts to ser. This is what the **ul_add** FIFO command is good for. Paremeterized by user's name, table name, expiration time and weight, it allows external applications to introduce new contacts to server's in-memory user location table.

The example is borrowed from serweb, ser's web PHP-written interface. It consists of a short "stub" function which carries out all mechanics of FIFO communication and of forming the FIFO request.

```
/* construct and send a FIFO command; the command parameters $sip_address,
   $expires are PHP variables originating from an HTML form
 */
$fifo_cmd=":ul_add:".$config->reply_fifo_filename."\n".
$config->ul_table."\n". //table
$user_id."\n". //username
$sip_address."\n". //contact
$expires."\n". //expires
$config->ul_priority."\n\n"; //priority
$message=write2fifo($fifo_cmd, $errors, $status);

/* .......... snip .................. */

/* this is the stub function for communicating with FIFO server.
   it dumps a request to FIFO server, opens a reply FIFO and
   reads server's reply from it
*/
function write2fifo($fifo_cmd, &$errors, &$status){
global $config;

/* open fifo now */
$fifo_handle=fopen( $config->fifo_server, "w" );
if (!$fifo_handle) {
$errors[]="sorry -- cannot open fifo"; return;
}

/* create fifo for replies */
@system("mkfifo -m 666 ".$config->reply_fifo_path );

/* add command separator */
$fifo_cmd=$fifo_cmd."\n";

/* write fifo command */
if (fwrite( $fifo_handle, $fifo_cmd)==-1) {
    @unlink($config->reply_fifo_path);
    @fclose($fifo_handle);
$errors[]="sorry -- fifo writing error"; return;
}
@fclose($fifo_handle);

/* read output now */
@$fp = fopen( $config->reply_fifo_path, "r");
if (!$fp) {
    @unlink($config->reply_fifo_path);
$errors[]="sorry -- fifo reading error"; return;
}

$status=fgetS($fp,256);
if (!$status) {
    @unlink($config->reply_fifo_path);
```

```
$errors[]="sorry -- fifo reading error"; return;
}

$rd=fread($fp,8192);
@unlink($config->reply_fifo_path);

return $rd;
}
```

See Section 6.5 for a complete listing of FIFO commands available with current ser distribution.

4.2.1. Advanced Example: Click-To-Dial

A very useful SIP application is phonebook with "click-to-dial" feature. It allows users to keep their phonebooks on the web and dial by clicking on an entry. The great advantage is that you can use the phonebook alone with any phone you have. If you temporarily use another phone, upgrade it permanently with another make, or use multiple phones in parallel, your phonebook will stay with you on the web. You just need to click an entry to initiate a call. Other scenario using "click-to-dial" feature includes "click to be connected with our sales representative".

There are basically two ways how to build such a feature: distributed and centralized. We prefer the distributed approach since it is very robust and leight-weighted. The "click-to-dial" application just needs to instruct the calling user to call a destination and that’s it. (That’s done using "REFER" method.) Then, the calling user takes over whereas the initating application disappears from signaling and is no longer involved in subsequent communication. Which is good because such a simple design scales well.

The other design alternative is use of a B2BUA [1] which acts as a "middleman" involved in signaling during the whole session. It is complex: ringing needs to be achieved using a media server, it introduces session state, mangling of SIP payloads, complexity when QoS reservation is used and possibly other threats which result from e2e-unfriendly design. The only benefit is it works even for poor phones which do not support REFER -- which should not matter because you do not wish to buy such.

So how does "distributed click-to-dial" application work? It is simple. The core piece is sending a REFER request to the calling party. REFER method is typically used for call transfer and it means "set up a call to someone else".

There is an issue -- most phones don’t accept unsolicited REFER. If a malicious user made your phone to call thirty destinations without your agreement, you would certainly not appreciate it. The workaround is that first of all the click-to-dial application gives you a "wrapper call". If you accept it, the application will send a REFER which will be considered by the phone as a part of approved communication and granted. Be aware that without cryptography, security is still weak. Anyone who saw an INVITE can generate an acceptable REFER.

Example 4-12. Call-Flow for Click-To-Dial Using REFER

```
        CTD                  Caller               Callee
            #1 INVITE
            ----------------->
                             ...
```

```
                    caller answers
    #2 200
    <-----------------
    #3 ACK
    ----------------->
    #4 REFER
    ----------------->
    #5 202
    <-----------------
    #6 BYE
    ----------------->
    #7 200
    <-----------------
                         #8 INVITE
                         ------------------>
                         #9 180 ringing
                         <------------------


#1 click-to-dial (CTD) is started and the "wrapper call" is initiated
INVITE caller
From: controller
To: caller
SDP: on hold

#2 calling user answes
200 OK
From: controller
To: caller

#3 CTD acknowledges
ACK caller
From controller
To: caller

#4 CTD initiates a transfer
REFER caller
From: controller
To: caller
Refer-To: callee
Refered-By: controller

#5 caller confirms delivery of REFER
202 Accepted
From: controller
To: caller

#6 CTD terminates the wrapper call -- it is no longer needed
BYE caller
From: controller
To: caller

#7 BYE is confirmed
200 Ok
From: controller
To: caller
```

```
#8 caller initates transaction solicited through REFER
INVITE callee
From: caller
To: callee
Referred-By: controller

#9 that's it -- it is now up to callee to answer the INVITE
180 ringing
From: caller
To: callee
```

Implementation of this scenario is quite straight-forward: you initiate INVITE, BYE and REFER transaction. Source code of the example written in Bourne shell is available in source distrubtion, in `examples/ctd.sh`. A PHP implementation exists as well as a part of serweb.

Example 4-13. Running the CTD Example

```
[jiri@cat examples]$ ./ctd.sh
destination unspecified -- taking default value sip:23@192.168.2.16
caller unspecified -- taking default value sip:113311@192.168.2.16
invitation succeeded
refer succeeded
bye succeeded
```

Notes

1. See `draft-ietf-sipping-3pcc-02.txt` for more details.

Chapter 5. Complementary Applications

5.1. serctl command-line tool

serctl is a command-line utility which allows to perform most of management tasks needed to operate ser: adding users, changing their passwords, watching server status, etc. Usage of utility is as follows:

Example 5-1. serctl usage

```
usage:
           * subscribers *
 serctl add <username> <password> <email> .. add a new subscriber (*)
 serctl passwd <username> <passwd> ......... change user's password (*)
 serctl rm <username> ...................... delete a user (*)
 serctl mail <username> .................... send an email to a user
 serctl alias show [<alias>] ............... show aliases
 serctl alias rm <alias> ................... remove an alias
 serctl alias add <alias> <uri> ............ add an aliases

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

           * usrloc *
 serctl ul show [<username>]................ show in-RAM online users
 serctl ul rm <username> ................... delete user's UsrLoc entries
 serctl ul add <username> <uri> ............ introduce a permanent UrLoc entry
 serctl showdb [<username>] ................ show online users flushed in DB

           * server health *
 serctl monitor ........................... show internal status
 serctl ps ................................ show runnig processes
 serctl fifo .............................. send raw commands to FIFO

   Commands labeled with (*) will prompt for a MySQL password.
   If the variable PW is set, the password will not be prompted.
```

Note: Prior to using the utility, you have to first set the environment variable `SIP_DOMAIN` to locally appropriate value (e.g., "foo.com"). It is needed for calculation of user credentials, which depend on SIP digest realm. (see also MSM Authentication Issue)

Example 5-2. Example Output of Server Watching Command sc monitor

```
[cycle #: 2; if constant make sure server lives and fifo is on]
Server: Sip EXpress router(0.8.8 (i386/linux))
Now: Thu Sep 26 23:16:48 2002
Up Since: Thu Sep 26 12:35:27 2002
Up time: 38481 [sec]

Transaction Statistics
Current: 0 (0 waiting) Total: 606 (0 local)
Replied localy: 34
Completion status 6xx: 0, 5xx: 1, 4xx: 86, 3xx: 0,2xx: 519

Stateless Server Statistics
200: 6218 202: 0 2xx: 0
300: 0 301: 0 302: 0 3xx: 0
400: 0 401: 7412 403: 2 404: 1258 407: 116 408: 0 483: 0 4xx: 25      500: 0 5xx: 0
6xx: 0
xxx: 0
failures: 0

UsrLoc Stats
Domain Registered Expired
'aliases' 9 0
'location' 29 17
```

5.2. Web User Provisioning -- serweb

To make provisioning of user accounts convenient, a web front-end to ser, serweb has been developed. serweb, a PHP-written web application, allows users to apply for new ser accounts, and maintain these. Users can manipulate their contacts, keep a phone-book with SIP addresses, change password, send instant SIP messages, and more. Administrators can manipulate any accounts and grant or revoke user privileges.

serweb is freely available from berlios site at http://developer.berlios.de/cvs/?group_id=500 (http://developer.berlios.de/cvs/?group_id=500). Installation takes unpacking tarball to a safe destination at web server (better not in the HTML tree) and configuring `config.php` accordingly to local conditions.

Running serweb can be seen at iptel.org's SIP site. Just create and use a SIP account at http://www.iptel.org/user/

5.3. Voicemail

5.3.1. Introduction

The voicemail system provides ser with voice announcement and recording capabilities. Voice messages may then be mailed to the called user. The system relies on ser for implementing the SIP stack and communicate with it through FIFO. It implements the dialog and media handling as described in RFC 3264 (An Offer/Answer Model with the Session Description Protocol) and RFC 1889 (Real time transport protocol) to realize its goal.

5.3.2. Advantages

- Anyone deploying ser and VoIP should profit from this 'ready-to-run' application. It plugs into ser as easy as configuring the database location, announce file path and SMTP server address.
- Further, voicemail integrates the most popular free codecs (G.711ulaw, G.711alaw and GSM 06.10) and its own SMTP client, which means that you don't need to install anything else as ser and voicemail.
- If you want your voicemail system to support other codecs, a simple plugin system with SDK alows you to integrate them fast and simply (see the basis plugins for examples).

5.3.3. Technical limitations

- The sound conversion engine doesn't support yet resampling. It means that input and ouput files have to be compatible with the sampling rate of the codec. All codecs included with the distribution work at 8kHz, which means that all the input and output files MUST be sampled at the rate of 8kHz.
- At the moment, voicemail only support the Microsoft Wav file format with PCM 16 bit, Mu-law and A-law 8 bit encoding.

5.3.4. Compilation and installation

- Configure Ser to fit your needs. You can refer to voicemail example config file to know what your configuration file should include. Note that voicemail uses subsriber database table to determine recepient's email address. Read the README file in the vm module directory for complete description of the functions and variables that are used by voicemail and how they work.
- Finally, compile the voicemail application:

  ```
  [~/voicemail]$ cd ortp-0.5.0
  [~/voicemail/ortp-0.5.0]$ ./configure
  [~/voicemail/ortp-0.5.0]$ make all
  [~/voicemail/ortp-0.5.0]$ cd ..
  [~/voicemail]$ cd plug-in/gsm/gsm-????
  [~/voicemail/plug-in/gsm/gsm-????]$ make all
  [~/voicemail/plug-in/gsm/gsm-????]$ cd ../..
  [~/voicemail]$ make all
  ```

  You can then start voicemail with following command **ans_machine** and look if the default fit your needs. If not, type **ans_machine -h** to see how to change the default parameters. If ans_machine is not started or can't be joined while ser tries to communicate with it, the caller will become a '500 internal server error' with a comment saying what the trouble is.

5.3.5. Example ser Config File

Example 5-3. Example ser Config File

```
#
# $Id: ser.cfg,v 1.11 2003/07/03 12:17:56 rco Exp $
#
# iptel's voicemail config script
#

# ----------- global configuration parameters ------------------------

debug=7         # debug level (cmd line: -dddddddddd)
fork=no

log_stderror=yes # (cmd line: -E)
check_via=no # (cmd. line: -v)
dns=no # (cmd. line: -r)
rev_dns=no # (cmd. line: -R)
port=5060
children=4
fifo="/tmp/ser_fifo"

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

loadmodule "modules/sl/sl.so"
loadmodule "modules/tm/tm.so"
loadmodule "modules/maxfwd/maxfwd.so"
loadmodule "modules/rr/rr.so"
loadmodule "modules/textops/textops.so"
loadmodule "modules/vm/vm.so"
loadmodule "modules/dbtext/dbtext.so"

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

modparam("voicemail", "db_url","/home/rco/cvs/ser/sip_router/modules/vm/db")

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

# main routing logic

route{

# initial sanity checks -- messages with
# max_forwars==0, or excessively long requests
if (!mf_process_maxfwd_header("10")) {
sl_send_reply("483","Too Many Hops");
break;
};
if (len_gt( max_len )) {
sl_send_reply("513", "Message too big");
break;
```

```
};

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

# Make MSN Messenger happy...
if (method=="REGISTER") {
sl_send_reply("200","ok");
break;
};

if (uri == myself) {

# Voicemail specific configuration - begin

if(method=="ACK" || method=="INVITE" || method=="BYE" || method=="REFER"){

if(t_newtran()){

t_reply("100","Trying -- just wait a minute !");

if(method=="INVITE" || method=="REFER"){
log("**************** vm start - begin ******************\n");
if( uri =~ "conference" ){
if(!vm("/tmp/am_fifo","conference")){
log("could not contact conference server\n");
t_reply("500","could not contact conference server");
};
}
else if( uri =~ "echo" ){
if(!vm("/tmp/am_fifo","echo")){
log("could not contact echo\n");
t_reply("500","could not contact echo");
};
}
else {
if(!vm("/tmp/am_fifo","voicemail")){
log("could not contact voicemail\n");
t_reply("500","could not contact voicemail");
};
};
log("**************** vm start - end ******************\n");
break;
};

if(method=="BYE"){
log("**************** vm end/refer - begin ******************\n");
if(!vm("/tmp/am_fifo","bye")){
log("could not contact the media server\n");
t_reply("500","could not contact the media server");
};
log("**************** vm end/refer - end *******************\n");
```

```
break;
};
}
else {
     log("could not create new transaction\n");
     sl_send_reply("500","could not create new transaction");
};
};

# Voicemail specific configuration - end
}
else {
if (!t_relay()) {
sl_reply_error();
};
};
}
```

5.3.6. Availabilty, report bugs, contact the author

Ser's Voicemail's home page is hosted at http://sems.berlios.de. A snapshot may be downloaded directly from the CVS tree. A pre-configured version of ser including voicemail will be soon available (from version 0.8.11). Bugs can be reported at the voicemail's home page. If you want to contact the author, use the contact email at the home page.

Chapter 6. Reference

6.1. Core Options

Core options are located in beginning of configuration file and affect behaviour of the server.

- `debug` - Set log level, this is number between 0 and 9. Default is 0.
- `fork` - If set to yes, the server will spawn children. If set to no, the main process will be processing all messages. Default is yes.

  > **Note:** Disabling child spawning is useful mainly for debugging. When `fork` is turned off, some features are unavailable: there is no attendant process, no pid file is generated, and server listens only at one address. Make sure you are debugging the same interface at which ser listens. The easiest way to do so is to set the interface using `listen` option explicitly.

- `log_stderror` - If set to yes, the server will print its debugging information to standard error output. If set to no, **syslog** will be used. Default is no (printing to syslog).
- `listen` - list of all IP addresses or hostnames SER should listen on.

  > **Note:** This parameter may repeat several times, then SER will listen on all addresses. For example, the following command-line options (equivalent to "listen" config option) may be used: **ser -l foo -l bar -p 5061 -l x -l y** will listen on foo:5060, bar:5061 & x:5061 & y:5061

- `alias` - Add IP addresses or hostnames to list of name aliases. All requests with hostname matching an alias will satisfy the condition "uri==myself".
- `dns` - Uses dns to check if it is necessary to add a "received=" field to a via. Default is no.
- `rev_dns` - Same as dns but use reverse DNS. Default is no.
- `port` - Listens on the specified port (default 5060). It applies to the last address specified in listen and to all the following that do not have a corresponding "port" option.
- `maxbuffer` - Maximum receive buffer size which will not be exceeded by the auto-probing procedure even if the OS allows. Default value is MAX_RECV_BUFFER_SIZE, which is 256k.
- `children` - Specifies how many processes should be started for each transport protocol. Running multiple children allows a server to server multiple requests in parallel when request processing block (e.g., on DNS lookup). Note that ser typically spawns additional processes, such as timer process or FIFO server. If FIFO server is turned on, you can watch running processes using the serctl utility.
- `check_via` - Turn on or off Via host checking when forwarding replies. Default is no.
- `syn_branch` - Shall the server use stateful synonym branches? It is faster but not reboot-safe. Default is yes.
- `memlog` - Debugging level for final memory statistics report. Default is L_DBG -- memory statistics are dumped only if `debug` is set high.
- `sip_warning` - Should replies include extensive warnings? By default yes, it is good for trouble-shooting.

- `fifo` - FIFO special file pathname, for example "/tmp/ser_fifo". Default is no filename -- no FIFO server is started then. We recommend to set it so that accompanying applications such as serweb or serctl can communicate with ser.
- `fifo_mode` - Permissions of the FIFO special file.
- `server_signature` - Should locally-generated messages include server's signature? By default yes, it is good for trouble-shooting.
- `reply_to_via` - A hint to reply modules whether they should send reply to IP advertised in Via. Turned off by default, which means that replies are sent to IP address from which requests came from.
- `user | uid` - uid to be used by the server.
- `group | gid` - gid to be used by the server.
- `mhomed` -- enable calculation of outbound interface; useful on multihomed servers, ser .
- `loadmodule` - Specifies a module to be loaded (for example "/usr/lib/ser/modules/tm.so")
- `modparam` - Module parameter configuration. The commands takes three parameters:
  - *module* - Module in which the parameter resides.
  - *parameter* - Name of the parameter to be configured.
  - *value* - New value of the parameter.

6.2. Core Commands

Route Blocks and Process Control

- **route[number]{...}** - This marks a "route block" in configuration files. route blocks are basic building blocks of ser scripts. Each route block contains a sequence of SER actions enclosed in braces. Multiple route blocks can be included in a configuration file. When script execution begins on request receipt, route block number 0 is entered. Other route blocks serve as a kind of sub-routines and may be entered by calling the action **route(n)**, where n is number of the block. The action **break** exits currently executed route block. It stops script execution for route block number 0 or returns to calling route block otherwise.

  Example 6-1. route

```
route[0] {
        # call routing block number 2
route(2);
}

route[2] {
    forward("host.foo.bar", 5060);
}
```

- **failure_route** is used to restart request processing when a negative reply for a previously relayed request is received. It is only used along with tm module, which stores the original requests and can return to their processing later. To activate processing of a **failure_route** block, call the TM action **t_on_failure(route_number)** before calling **t_relay**. When a negative reply comes back, the desired **failure_route** will be entered and processing of the original request may continue.

The set of actions applicable from within **failure_route** blocks is limited. Permitted actions are URI-manipulation actions, logging and sending stateful replies using **t_reply**.

Example 6-2. failure_route

```
failure_route[1] {
    # for some reason, the original forwarding attempt
    # failed, try at another URI
    append_branch("sip:nonsense@iptel.org");
    # if this new attempt fails too, try another failure_route
    t_on_failure("2");
t_relay();
}
```

- The action **break** exits currently executed route block. It stops script execution for route block number 0 or returns to calling route block otherwise.

  Note: We recommend to use **break** after any request forwarding or replying. This practice helps to avoid erroneous scripts that continue execution and mistakenly send another reply or forward a request to another place, resulting in protocol confusion.

  Example: break;

- **route(n)** - call routing block route[n]{...}; when the routing block n finishes processing, control is passed back to current block and processing continues.
- **if (condition) statement** - Conditional statement.

  Example 6-3. Use of if

```
if (method=="REGISTER) {
    log("register received\n");
};
```

- **if - else** - If-Else Conditional statement.

  Example 6-4. Use of if-else

```
if (method=="REGISTER) {
    log("register received\n");
} else {
    log("non-register received\n");
};
```

Flag Manipulation

- **setflag** - Set flag in the message.

Example: setflag(1);

- **resetflag** - Reset flag in the message.

  Example: resetflag(1);

- **isflagset** - Test whether a particular flag is set.

  Example 6-5. isflagset

  ```
  if (isflagset(1)) {
      ....
  };
  ```

Manipulation of URI and Destination Set

- **rewritehost | sethost | seth** - Rewrite host part of the Request URI.

  Example: sethost("foo.bar.com");

- **rewritehostport | sethostport | sethp** - Rewrite host and port part of the Request URI.

  Example: sethostport("foo.bar.com:5060");

- **rewriteuser | setuser | setu** - Rewrite or set username part of the Request URI.

  Example: setuser("joe");

- **rewriteuserpass | setuserpass | setup** - Rewrite or set username and password part of the Request URI.

  Example: setuserpass("joe:mypass");

- **rewriteport | setport | setp** - Rewrite or set port of the Request URI.

  Example: setport("5060");

- **rewriteuri | seturi** - Rewrite or set the whole Request URI.

  Example: seturi("sip:joe@foo.bar.com:5060");

- **revert_uri** - Revert changes made to the Request URI and use original Request URI.

  Example: revert_uri();

- **prefix** - Add prefix to username in Request URI.

  Example: prefix("123");

- **strip** - Remove first n characters of username in Request URI.

  Example: strip(3);

- **append_branch** - Append a new destination to destination set of the message.

  Example 6-6. Use of append_branch

  ```
  # redirect to these two destinations: a@foo.bar and b@foo.bar
  # 1) rewrite the current URI
  rewriteuri("sip:a@foo.bar");
  # 2) append another entry to the destination ser
  append_branch("sip:b@foo.bar");
  # redirect now
  sl_send_reply("300", "redirection");
  ```

Message Forwarding

- **forward(uri, port)** - Forward the request to given destination statelessly. The uri and port parameters may take special values 'uri:host' and 'uri:port' respectively, in which case SER forwards to destination set in current URI. All other elements in a destination set are ignored by stateless forwarding.

  Example: forward("foo.bar.com"); # port defaults to 5060

- **send** - Send the message as is to a third party

  Example: send("foo.bar.com");

Logging

- **log([level], message)** - Log a message.

  Example: log(1, "This is a message with high log-level set to 1\n");

  Logging is very useful for troubleshooting or attracting administrator's attention to unusual situations. ser reports log messages to syslog facility unless it is configured to print them to `stderr` with the `log_stderr` configuration option. Log messages are only issued if their log level exceeds threshold set with the `debug` configuration option. If log level is omitted, messages are issued at log level 4.

Miscellaneous

- **len_gt** - If length of the message is greater than value given as parameter, the command will return 1 (indicating true). Otherwise -1 (indicating false) will be returned. It may take 'max_len' as parameter, in which case message size is limited to internal buffer size BUF_SIZE (3040 by default).

  Example 6-7. Use of len_gt

  ```
  # deny all requests larger in size than 1 kilobyte
  if (len_gt(1024)) {
      sl_send_reply("513", "Too big");
      break;
  };
  ```

6.3. Command Line Parameters

> **Note:** Command-Line parameters may be overridden by configuration file options which take precedence over them.

- *-h* - Displays a short usage description, including all available options.
- *-c* - Performs loop checks and computes branches.
- *-r* - Uses dns to check if it is necessary to add a "received=" field to a via.
- *-R* - Same as -r but uses reverse dns.
- *-v* - Turns on via host checking when forwarding replies.
- *-d* - Turns on debugging, multiple -d increase debugging level.
- *-D* - Runs ser in the foreground (it doesn't fork into daemon mode).
- *-E* - Sends all the log messages to stderr.
- *-V* - Displays the version number.
- *-f config-file* - Reads the configuration from "config-file" (default ./ser.cfg).
- *-l address* - Listens on the specified address. Multiple -l mean listening on multiple addresses. The default behaviour is to listen on all the ipv4 interfaces.
- *-p port* - Listens on the specified port (default 5060). It applies to the last address specified with -l and to all the following that do not have a corresponding -p.
- *-n processes-no* - Specifies the number of children processes forked per interface (default 8).
- *-b max_rcv_buf_size* - Maximum receive buffer size which will not be exceeded by the auto-probing procedure even if the OS allows.
- *-m shared_mem_size* - Size of the shared memory which will be allocated (in Megabytes).
- *-w working-dir* - Specifies the working directory. In the very improbable event that will crash, the core file will be generated here.

- *-t chroot-dir* - Forces ser to chroot after reading the config file.
- *-u uid* - Changes the user id under which ser runs.
- *-g gid* - Changes the group id under which ser runs.
- *-P pid-file* - Creates a file containing the pid of the main ser process.
- *-i fifo-path* - Creates a fifo, useful for monitoring ser status.

6.4. Modules

Module description is currently located in READMEs of respective module directories. `README-MODULES` lists all available modules, including their maturity status. In the current ser distribution, there are the following modules:

- *acc* -- call accounting using syslog facility. RADIUS and mysql support can be compiled in. Depends on tm.
- *auth, auth_db, auth_radius* -- digest authentication. Depends on sl and mysql.
- *domain* -- checks URIs whether they belong in a list of served domains or not.
- *ENUM* -- E.164 phone number resolution using ENUM.
- *exec* -- execution of shell programs.
- *group, group_radius* -- checks membership of users in groups
- *jabber* -- gateway between SIMPLE and Jabber instant messaging. Depends on tm and mysql.
- *maxfwd* -- checking max-forwards header field.
- *msilo* -- message silo. Store for undelivered instant messages. Depends on tm and mysql.
- *mysql* -- mysql database back-end.
- *nathelper* -- facilitates NAT traversal for symmetric SIP phones such as ATA.
- *pa* -- presence agent
- *registrar, usrloc* -- User Location database. Works in in-memory mode or with mysql persistence support. Depends on sl, and on mysql if configured for use with mysql.
- *rr* -- Record Routing (strict and loose)
- *sl* -- stateless User Agent server.
- *sms* -- SIMPLE/SMS gateway. Depends on tm. Takes special hardware.
- *textops* -- textual database back-end.
- *tm* -- transaction manager (stateful processing).
- *uri, uri_radius* -- checks digest identity against header URIs or a database list

The most frequently used actions exported by modules are summarized in Table 6-1. For a full explanation of module actions, see documentation in respective module directories in source distribution of ser.

Command	Modules	Parameters	Comments

Table 6-1. Frequently Used Module Actions

Command	Modules	Parameters	Comments
append_hf	textops	header field	append a header field to the end of request's header
check_from	uri	none	check if username in from header field matches authentication id
check_to	uri	none	check if username in To header field matched authentication id
exec_dset	exec	command name	execute an external command and replace destination set with its output
exec_msg	exec	command name	execute an external command and pass received SIP request to its input
is_user	uri	user id	returns true if request credentials belong to a user
is_user_in	auth	user, group	check if user is member of a group; user can be gained from request URI ("Request-URI"), To header field ("To"), From header field ("From") or digest credentials ("Credentials")
lookup	usrloc	table name	attempt to translate request URI using user location database; returns false if no contact for user found;
loose_route	rr	none	process loose routes in requests
mf_process_maxfwd_header	maxfwd	default max_forwards value	return true, if request's max_forwards value has not reached zero yet; if none is included in the request, set it to value in parameter
proxy_authorize	auth	realm, subscriber table	returns true if requests contains proper credentials, false otherwise
proxy_challenge	auth	realm, qop	challenge user to submit digest credentials; qop may be turned off for backwards compatibility with elderly implementations

Command	Modules	Parameters	Comments
record_route	rr	none	record-route a request
replace	textops	RegExp, Substitute	find the first occurence of a string matching the regular expression in header or body and replace it with a substitute
replace_all	textops	RegExp, Substitute	find all occurences of a string matching the regular expression in header or body and replace it with a substitute
save	usrloc	table name	for use in registrar: save content of Contact header fields in user location database and reply with 200
search	textops	regular expression	search for a regular expression match in request header of body
sl_send_reply	sl	status code, reason phrase	reply a request statelessly
t_relay	tm	none	stateful forwarding to locations in current destination set
t_on_failure	tm	failure_route number	set failure_route block which shall be entered if stateful forwarding fails
t_replicate	tm	host, port number	replicate a request to a destination

6.5. FIFO Commands Reference

This section lists currently supported FIFO commands. Some of them are built-in in ser core, whereas others are exported by modules. The most important exporters are now tm and usrloc module. tm FIFO commands allow users to initiate transactions without knowledge of underlying SIP stack. usrloc FIFO commands allow users to access in-memory user-location database. Note that that is the only way how to affect content of the data-base in real-time. Changes to MySql database do not affect in-memory table unless ser is restarted.

Table 6-2. FIFO Commands

Command	Module	Parameters	Comments
ps	core	none	prints running ser processes
which	core	none	prints list of available FIFO commands

Command	Module	Parameters	Comments
arg	core	none	prints list of command-line arguments with which ser was started
pwd	core	none	prints ser's working directory
version	core	none	prints version of ser
uptime	core	none	prints ser's running time
sl_stats	sl	none	prints statistics for sl module
t_stats	tm	none	print statistics for tm module
t_hash	tm	none	print occupation of transaction table (mainly for debugging)
t_uac_dlg	tm	method, request URI, outbound URI (if none, empty line with a single dot), dot-line-terminated header fields, optionaly dot-line terminated message body.	initiate a transaction. From and To header fields must be present in header field list, so does Content-Type if body is present. If CSeq, CallId and From-tag are not present, ephemeral values are generated. Content_length is calculated automatically if body present.
ul_stats	usrloc	none	print usrloc statistics
ul_rm	usrloc	table name, user name	remove all user's contacts from user-location database
ul_rm_contact	usrloc	table name, user name, contact	remove a user's contact from user-location database
ul_dump	usrloc	none	print content of in-memory user-location database
ul_flush	usrloc	none	flush content of in-memory user-location cache in persistent database (MySQL)
ul_add	usrloc	table name, user name, contact, expiration, priority (q)	insert a contact address in user-location database
ul_show_contact	usrloc	table, user name	show user's contact addresses in user-location database

6.6. Used Database Tables

ser includes MySQL support to guarantee data persistence across server reboots and storage of users' web environment. The data stored in the database include user profiles, access control lists, user location, etc. Note that users are not supposed to alter the data directly, as it could introduce inconsistency between data on persistence storage and in server's memory. The following list enumerates used tables and explains their purpose.

- subscriber -- table of users. It includes user names and security credentials, as well as additional user information.
- reserved -- reserved user names. serweb does not permit creation of accounts with name on this list.
- phonebook -- user's personal phonebooks. Accessible via serweb.
- pending -- table of unconfirmed subscription requests. Used by serweb.
- missed_calls -- table of missed calls. Can be fed by acc modules if mysql support is turned on. Displayed by serweb.
- location -- user contacts. Typically updated through ser'r registrar functionality.
- grp -- group membership. Used by auth module to determine whether a user belongs to a group.
- event -- allows users to subscribe to additional services. Currently unused.
- aliases -- keeps track of alternative user names.
- active_sessions -- keeps track of currently active web sessions. For use by serweb.
- acc -- keeps track of accounted calls. Can be fed by acc module if mysql support is turned on. Displayed by serweb.
- config -- maintains attribute-value pairs for keeping various information. Currently not used.
- silo -- message store for instant messages which could not have been delivered.
- version -- keeps version number of each table definition.