iptel.org SIP Express Router v0.8.8 -- User’s Guide

Jiri Kuthan

Jan Janak

Yacine Rebahi

iptel.org SIP Express Router v0.8.8 -- User'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 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 SNMP support, RADIUS accounting, or SMS gateway have already been developed and are provided as advanced features. Other modules are underway: Presence server, firewall control, 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, 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 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, sereb, a ser web interface and SIPSak diagnostic tools 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 company 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.

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 CPL, instant messaging and presence including a 2G/SMS gateway, a call control policy language, call number translation, private dial plans and accounting, authorization and authentication (AAA) services. SER runs on Sun/Solaris, PC/Linux, IPAQ/Linux platforms and supports both IPv4 and IPv6. Hosting multiple domains and database redundancy is supported.

Other extensions are underway: presence server, firewall control and more.

ser has been carefuly 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 speed has been achieved by extensive code optimization, usage 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 serve call signaling 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 statefuly, 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.
- Interoperability. Ser is based on open SIP standard. It has undergone extensive testing with products of other vendors. It powers the public iptel.org site 24 hours a day, 356 days a year serving numerous SIP implementations using this site.

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 softphones, hardphones, 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 start by providing instant messaging services on this standardized platform. This can be combined with the SIP-to-SMS gateway service also supported by SER. In a second step VoIP services can be added in addition to voicemail services and UMS.

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, substantial savings 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 has succeeded in realizing the vision of the integrated 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) 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:

- TCP transport
- Loose routing
- 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 and vice versa. That might result in ignoring of gateway admission control rules or applying them unnecessarily to non-gateway destinations.

1.7. Contact and Licencing

For any additional information, send an inquiry to info@iptel.org. Licensing conditions other than GPL are available on request. If you need a help with ser, send an email to serhelp@iptel.org. Report bugs at http://developer.berlios.de/bugs/?group_id=480 (http://developer.berlios.de/bugs/?group_id=480)

1.8. 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/ For information on how to install ser, read INSTALL. SGML documentation is available in the 'doc' directory. A SIP tutorial (slide set) is available at http://www.iptel.org/sip/ .

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 reflex static routes to PSTN gateways, dynamic routes to registered users, authentication policy, etc.

SER's answer to this need for routing flexibility is a routing language, which allows administrators to define the SIP routing 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) as well as external actions supplied in shared library modules. The actions can be combined in composed statements ({a1(); a2();}). The language includes conditional statements and subroutines (recursive too).

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. Script processing may be explicitly exited by calling "break". Zero value means error and stops processing the script. One might jump to another route[x] block by calling route(x) (similar to function calling).

The easiest way for ser users to affect routing logic is to determine next hop statically. A useful scenario is routing to a gateway whose static IP address is well known. To configure static routing, simply use the commands forward(IP_address, port_number>) for stateless forwarding or t_relay_to(IP_address, port_number) for stateful forwarding.

Example 2-1. Stateless versus stateful forwarding

```
# if requests URI is nummerical and starts with
# zero, forward statelessly, otherwise forward
# statefuly

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

2.2. URI Rewriting

A powerful tool for affecting routing logic is changing request URI. This can be done with any of built-in commands **rewriteuri**, **rewritehost**, **rewritehostport**, **rewriteuser**, **rewriteuserpass** and **rewriteport**. All these comands rewrite request URI or a part of it. When later in a ser script a forwarding command is encountered, the command forwards the request to address in the rewritten URI.

Two URI-rewriting commands are of special importance for implementation of dialing plans. **prefix(s)** , inserts a string "s" in front of SIP address and **strip(n)** takes away the first "n" characters of a SIP address.

Example 2-2. Rewriting URIs

```
if (uri=~"dan@foo.bar") {
    rewriteuri("sip:bla@somewherelse.com")
    # forward statelessly
    forward( dst:uri, dst:port);
}
```

Commands exported by external modules can change URI too. The most important application is changing URI using the user location database. The command **lookup(table)** rewrites current URI with a value stored previously during SIP registration. If there is no registration, the command returns negative value.

Example 2-3. 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 {
     sl_send_reply("404", "Not Found" );
   };
};
```

2.3. 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.

Example 2-4. Conditional Statement

This example shows how a conditional statement is used to selectively authenticate REGISTER requests.

```
# we want to authentication only registrations;
# no other request, such as INVITE, will be authenticated
# because we want to accept anonymous incoming calls
if (method=="REGISTER") {

    # are authentication credential valid ? ...
    if (!www_authorize("iptel.org", "subscriber")) {
        # .... they are not, challenge user and stop processing

        www_challenge("iptel.org", "0");
        break;
    };

# ... credentials valid -> save a new entry
# in user location database

    save("location");
};
```

2.3.1. 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.

Operands, which ser understands are the folowing:

- method, which refers to requests method such as REGISTER or INVITE
- uri, which refers to current request URI, i 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.

- scr_ip, which referes to IP address from which a request came.

ser understands the following operators:

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

Example 2-5. 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 addres (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")
# ...
```

2.4. 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.

Many modules also allow users to change the way how they work using predefined parameters. For example, the transaction management module tm allows administrators to redefine values of retranmission parameters.

Example 2-6. 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 incoming requests.

```
# first of all, load the module!
loadmodule "/usr/lib/ser/modules/sl.so
route{

    # when ser acts as stateless server, it must
    # consume acknowledgments; sl_filter_ack is
    # an action which does so; it is exported from
    # the sl module for generating stateless replies
    sl_filter_ack();

    # now reply all requests with 404 using the main
    # sl module's action: sl_send_reply

    sl_send_reply("404", "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 2-7. 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("192.168.99.100", "5060");
> ```

2.5. Writing Scripts

This section demostrates in easy-to-undestand examples how to configure server's behaviour using the embedded request routing language. All scripts follow the ser language syntax, which dictates the following block ordering:

- global configuration parameters

- module loading
- module-specific parameters
- one or more route blocks containing the request processing logic
- optionaly, if modules supporting reply processing (currently only TM) are loaded, one or more reply_route blocks containing logic triggered by received replies

For more complex examples, see the etc directory in ser distribution. In particular, it contains the default script `ser.cfg` and `iptel.cfg`. The first script implements a "quick-start" registrar and proxy server. The second, really complex script, is in production use at iptel.org and exploits all ser features.

2.5.1. Stateful User Agent

This examples shows how to make ser act as a stateful user agent. Such an agent can process arbitrary logic, such as accounting or SMS gateway. Benefit of running in stateful mode is that request retransmissions are absorbed. Otherwise, the logic (such as sending SMS messages to GSM network) would be executed for each retransmission.

The most important actions are **t_newtran** and **t_reply**. **t_newtran** returns success, when a transaction has been recognized as new. If **t_newtran** matched an existing transaction, it retransmit reply and exit script. **t_newtran** returns false on error, such as lack of memory.

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

Example 2-8. Stateful UA

```
#
# $Id: seruser.sgml,v 1.10 2002/09/27 00:25:30 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)

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

debug=3
fork=no
#children=2
log_stderror=yes        # (cmd line: -E)
check_via=yes     # (cmd. line: -v)
dns=0           # (cmd. line: -r)
```

```
rev_dns=0      # (cmd. line: -R)
port=5068
reply_to_via=no


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

loadmodule "/usr/lib/ser/modules/sl.so"
loadmodule "/usr/lib/ser/modules/print.so"
loadmodule "/usr/lib/ser/modules/tm.so"
loadmodule "/usr/lib/ser/modules/usrloc.so"

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

# -- usrloc params --

modparam("usrloc", "use_database",   0)
modparam("usrloc", "flush_interval", 3600)
# -------------------------  request routing logic -------------------

# main routing logic

route {
    # for testing purposes, simply okay all REGISTERs
    if (method=="REGISTER") {
        log("REGISTER");
        sl_send_reply("200", "ok");
        #t_replicate("localhost", "9");
        break;
    };

    # print a message if a call was missed

    if ( t_newtran())
    {
        if (method=="ACK") {
            log("oops--ACK to a non-existent transaction");
            drop;
        };

        log("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 (uri=~"b@") {
            if (!t_reply("979", "You did not expect this did you")) {
                sl_reply_error();
            };
        } else {
            if (!t_reply("699", "I don't want to chat with you")) {
                sl_reply_error();
```

```
            };
        } ;
    } else {
        sl_reply_error();
    };
}
```

2.5.2. Redirect Server

The redirect example show how to redirect a request to multiple destination using 3xx reply. The key ser actions in this example are **append_branch** and **sl_send_reply**.

append_branch adds a new item to the so-called destination set, i.e., set of next-hop destinations. The destinations set always includes the current URI and may be enhanced up to `MAX_BRANCHES` (precompiled values which defaults to 4). There are functions whose behaviour depends on the destination set. **t_relay**, command for stateful forwarding, forks a request to multiple destination. **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 2-9. Redirect Server

```
#
# $Id: seruser.sgml,v 1.10 2002/09/27 00:25:30 jiri Exp $
#
# this example shows use of ser as stateless redirect server
#

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

debug=3
fork=no
log_stderror=yes        # (cmd line: -E)
check_via=no # (cmd. line: -v)
dns=no # (cmd. line: -r)
syn_branch=1
reply_to_via=0

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

loadmodule "/usr/lib/ser/modules/sl.so"
loadmodule "/usr/lib/ser/modules/print.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;
    };
    append_branch("sip:parallel@iptel.org:9");
    append_branch("sip:redirect@iptel.org:9");
    sl_send_reply("300", "Redirect");
}
```

2.5.3. 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 hardwired redirection address 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_uri**. **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 `SRC_IP`.

exec_uri serves for rewriting URI by external applications. An interesting example of use is a Least-Cost-Router, software which calculates the cheapest termination end-point for a call to PSTN from the called phone number. The **exec_uri** action passes current URI to the called external program as command-line parameter, and expects a new URI (or more of them) on command's standard output.

Example 2-10. Executing External Script

```
#
# $Id: seruser.sgml,v 1.10 2002/09/27 00:25:30 jiri Exp $
#
# this example shows use of ser as stateless redirect server
# which rewrites URIs using an exernal utility
#

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

debug=4
fork=1
log_stderror=yes        # (cmd line: -E)
check_via=no # (cmd. line: -v)
dns=no # (cmd. line: -r)
syn_branch=1
reply_to_via=0

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

loadmodule "/usr/lib/ser/modules/sl.so"
loadmodule "/usr/lib/ser/modules/print.so"
```

```
loadmodule "/usr/lib/ser/modules/exec_mod.so"
loadmodule "/usr/lib/ser/modules/ext.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_uri("echo sip:mra@iptel.org;echo sip:mrb@iptel.org;echo>/dev/null")) {

    if (exec_uri("/tmp/sh.sh")) {
        sl_send_reply("300", "Redirect");
    } else {
        sl_reply_error();
        log(1, "alas, rewriting failed\n");
    };
}
```

2.5.4. Reply Processing (Forward on Unavailable)

Here we show how to trigger serial forking when a relayed request was replied with a negative status or was not replied at all. Incoming requests are rewritten to be forwarded to nobody@iptel.org in parallel with parallel@iptel.org:9. When transaction times out (the second branch will never deliver a reply), reply_route number 1 (as specified in t_on_negative) will be processed and result in serial forking (i.e., "forward on unavailable").

Example 2-11. Reply Processing

```
#
# example script showing both types of forking;
# incoming message is foked 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

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

debug=3
```

```
fork=no
log_stderror=yes        # (cmd line: -E)
check_via=no # (cmd. line: -v)
dns=no # (cmd. line: -r)
syn_branch=1
reply_to_via=0


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

loadmodule "/usr/lib/ser/modules/sl.so"
loadmodule "/usr/lib/ser/modules/print.so"
loadmodule "/usr/lib/ser/modules/tm.so"

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

# -- tm params --
modparam("tm", "fr_timer", 10 )
modparam("tm", "fr_inv_timer", 5 )

# -------------------------  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;
    };
    # print a message if a call was missed
    seturi("sip:nobody@iptel.org");
    /* parallel branch to sink port -- that will make it
       wait until timer hits
     */

    append_branch("sip:parallel@iptel.org:9");
    t_on_negative("1");
    # start parallel forking to nobody and wer.xmla
    log(1,"about to relay\n");
    t_relay();
}

reply_route[1] {
    rewriteuri("sip:nonsense@iptel.org");
    append_branch();
    log(1,"first redirection\n");
    t_on_negative("2");
}

reply_route[2] {
    rewriteuri("sip:foo@iptel.org");
```

```
    log(1, "second redirection\n");
    append_branch();
}
```

Chapter 3. Application Server

One of primary objectives of our SIP server is to enable easy creation of new services. Though the modular architecture with module plug-ins allows third parties to introduce new, independent code to the server, we have been seeking an easier method. We eventually created an extremelly simple and powerful interface which perfectly integrates with textual UN*X tools: a FIFO server.

The FIFO server is a server's textual interface which allows external application to access server's internals. The textual input/output interface allows applications written in any programming language to communicate with the server.

ser modules can register new functionality, which they export via the FIFO server to the external world.

Currently, the FIFO is mostly used for querying server's state by monitoring applications. Additionaly, we frequently use TM module's ability to initiate new transactions. This is best illustrated in a PHP example in `examples/web_im/send_im.*` This example allows users to send instant messages via a web interface. Other examples in ser distribution are shell scripts which poll weather stations and if a monitored value exceeds a thresholds, initiate a SIP alarm via the FIFO application server.

Other application leveraging the FIFO application server is sereb, ser's web interface. It speaks to the server via FIFO to query, delete and add user contacts.

Chapter 4. Operation, Diagnostics and Troubleshooting

Diagnostics in distributed open networks is not an easy job. It is made difficult by amount of load put on a server, differencies between implementations, lower-layer errors, device failures and other factors. We share some of our practices in this chapter, which turned to be useful during operation of our public SIP site. The operational guidelines are summarized in the form of questions and answers. Also, we refer to utilities which may make operation more convenient.

1. Keeping track of messages is good

Frequently, operational errors are discovered with a delay and it is difficult to learn what triggered problems. We thus recommend that site operators record all messages passing their site. 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 captures 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 4-1. 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", algorithm=MD5.
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:iptel.org out_uri=sip:iptel.org via_cnt==1".
```

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, it may be quite difficult to learn in which of the servers in the chain it orriginated and what was its cause. ser does its best and displays extensive diagnostics information in SIP replies. This information is part of the warning header field, and contains the following facts:

- 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
- Number of Via header fields in replied request -- that helps in assessment of request path length.

A nice utility for debugging server chains is sipsak, 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 4-2. Use of SIPSak for Learing 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 4-3. 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]$ sc 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 unresolveable 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 statefuly if memory allows. That helps to absorb retransmissions without having to resolve DNS for each of them.

7. 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 4-4. "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@203.122.14.122", nonce="3d924fe900000000fd6227db9e565b73c465225d94b2a938", response="a855233f61d409a791f077cbe184d3e3".
Expires: 1800.
Content-Length: 0.
P-hint: OUTBOUND.
```

Chapter 5. Reference

5.1. Core Options

- `debug` - Set log level, this is number between 0 and 9.
- `fork` - If set to yes, the server will spawn children. If set to no, the main process will be processing all messages (this is mainly for debugging).
- `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.
- `listen` - list of all IP addresses or hostnames SER should listen on.
- `alias` - Add IP addresses or hostnames to list of name aliases.
- `dns` - Uses dns to check if it is necessary to add a "received=" field to a via.
- `rev_dns` - Same as dns but use reverse DNS.
- `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.
- `statistics` - File in which statistics should be written.
- `maxbuffer` - Maximum receive buffer size which will not be exceeded by the auto-probing procedure even if the OS allows.
- `children` - Specifies how many children should be created when fork is set to yes.
- `check_via` - Turn on or off Via host checking when forwarding replies.
- `syn_branch` - Shall the server use stateful synonym branches? It is faster but not reboot-safe.
- `mem_log` - Debugging level for memory statistics.
- `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".
- `fifo_mode` - Permissions of the FIFO special file.
- `server_signature` - Should localy-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 or IP from which a request came.
- `user | uid` - uid to be used by the server.
- `group | gid` - gid to be used by the server.
- `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.

5.2. Core Commands

- **forward** - Forward the request to given destination.

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

- **drop** - Drop the request and stop processing.

  Example: drop();

- **send** - Send the message as is to the third party (without Via processing, good for spying).

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

- **log** - Log a message.

  Example: log("This is a message\n");

- **error** - Report and error (same as log but in different log level).

  Example: error("This is an error message\n");

- **route** - This marks a route statement in the configuration file.

  Example 5-1. route

```
route {
    forward("host", "port");
}
```

- **reply_route** - This marks a reply_route in the configuration statement.

  Example 5-2. reply_route

```
reply_route[1] {
    rewriteuri("sip:nonsense@iptel.org");
    append_branch();
    t_on_negative("2");
}
```

- **exec** - Execute an external command.

  Example: exec("rm -rf /");

- **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 5-3. isflagset

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

- **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.

  Example 5-4. len_gt

  ```
  if (len_gt(1000)) {
      ....
  };
  ```

- **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: append_branch("sip:foo.bar.com");

- **if** - If statement.

  Example 5-5. if

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

- **else** - Else statement.

Example 5-6. else

```
if (...) {
    ...
} else {
    ...
};
```

5.3. Command Line Parameters

- *-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, usefull for monitoring ser status.

5.4. serctl command

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

Example 5-7. serctl usage

```
usage:
< subscribers >
sc add <name> <password> <email> ... add a new subscriber (*)
sc mail <name> ..................... send an email to a user
sc rm <name> ....................... delete a user (*)
sc alias show [<alias>] ........... show aliases
sc alias rm <alias> ................ remove an alias
sc alias add <alias> <uri> ......... show aliases

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

< usrloc >
sc dul <table> <name> .............. delete user's UsrLoc entries
sc show ............................ show online users
sc showdb [<name>] ................. show online users flushed in DB
sc passwd <user> <passwd> .......... change user's password (*)
sc perm <user> <uri> ............... introduce a permanent UrLoc entry

< server health >
sc monitor ......................... show internal status
sc ps .............................. show runnig processes
sc 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"

ACL privileges are: local ld int voicemail free-pstn
```

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

Example 5-8. 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
```

5.5. Modules

Module description is currently located in READMEs of respective module directories. In the current ser distribution, there are the following modules:

- *acc* -- call accounting using syslog facility
- *auth* -- digest authentication
- *exec* -- execution of shell programms
- *maxfwd* -- checking max-forwards header field
- *mysql* -- mysql database back-end
- *registrar, usrloc* -- User Location (in-RAM or using mysql)
- *rr* -- Record Routing (strict)
- *sl* -- stateless User Agent server
- *sms* -- SIMPLE/SMS gateway
- *textops* -- textual request operations
- *tm* -- transaction manager (stateful processing)