OpenSER Admin Course

Configuration Basics

Voice System SRL

http://www.voice-system.ro

http://www.openser.org

Configuration file

- it is a text file (only file-system storage supported)
- no native support for advanced text processing (like define, include, etc), but very easy and simple to integrate with M4
- the file is loaded and processed only at startup (not used at runtime)
- currently there is no support for re-loading the configuration file at runtime – you need to restart the application; fortunately the most changeable data is configured via DB, where reload is possible.

Ex:

```
openser -f  openser.cfg
```

Config file structure

configuration file format....

```
# global options
debug=9
fork=no
log_stderror=yes
listen=192.168.1.34:5060
.....
```

```
# module loading
mpath="/usr/local/lib/openser/modules/"
loadmodule="tm.so"
loadmodule="registrar.so"
.....
```

```
# module parameters
modparam("registrar","append_branches",1)
modparam("tm", "fr_inv_timer", 30)
modparam("nathelper", "rtpproxy_sock", "/var/run/rtpproxy.sock")
.....
```

```
# initial request route
route{
   .....
   if (route(1)) {
   }
   .....
   route(x);
}
# additional routes – can be invoked from any other route
route[1]{
   if (uri=~"sip:[0-9]+@") {
      return(1); # return true on the above route
   } else if (uri=~"sip:[a-zA-Z]+@") {
      return(-1); # return false
   }
   sl_send_reply("400","Bad URI");
   exit;
}
......
```

```
route[x]{
   t_on_reply("1");
   t_on_failure("1");
   t_relay();
   exit
}
# onreply routes – triggered by TM each time a reply is received; no routing is available
onreply_route[1]{
   xlog("","reply received from $si (method=$rm)\n")
   if (method=="INVITE" && nat_uac_test("1"))
      fix_nated_contact();
}
# failure routes – triggered by TM each time a negative final reply is elected;
failure_route[1]{
   if (t_check_status("486")) {
      append_branch("sip:192.168.2.77:5060")
      t_relay();
   }
}
```

Options that controls the core and all modules. Most relevant:

- Listen interfaces

  listen=udp:192.168.2.2:5060

  listen=tcp:192.168.2.3:5066 #requires TCP support

  listen=tls:192.168.2.3 #requires TLS support; TLS default port is 5061

- Logging

  debug=3 #logging level

  memlog=3 #log level for memory related debugging

  log_stderror=no #use syslog and not standard error

  log_facility=LOG_LOCAL0

  log_name=”my-proxy” #default is argv[0]

- Protocol control

  disable_tcp=no

  disable_tcp=no

  NOTE : UDP cannot be disabled as it is mandatory by RFC

- Number of processes

  fork = yes # fork additional SIP works

  NOTE : if fork "no", only first UDP interface will be used, and no other protocol will be enabled (TCP and TLS).

  children = 4 # number of processes per UDP interface

  tcp_children = 6 # total numbers of TCP SIP worker processes

- Daemon options

  gid/group=sip # unix group

  uid/user=sip # unix user

  wdir=”/” # working directory

  chroot=”/usr/local/openser-1.2”

  disable_core_dump=no # enable core dumping

- SIP identity

  server_header=”My openser”

  # default is “OpenSer (<version> (<arch>/<os>))”

  server_signature = yes

  user_agent_header=”My openser”

- Miscellaneous

```
alias=”mydomain.sip”   # to set alias hostnames for the server
auto_aliases=no        # discover aliases via reversed DNS
avp_aliases=”my_avp=i:34”

disable_dns_failover = yes

sip_warning=yes   #add a debugging header in replies
```

- Loading a module

loadmodule “/usr/lib/openser/modules/tm.so”

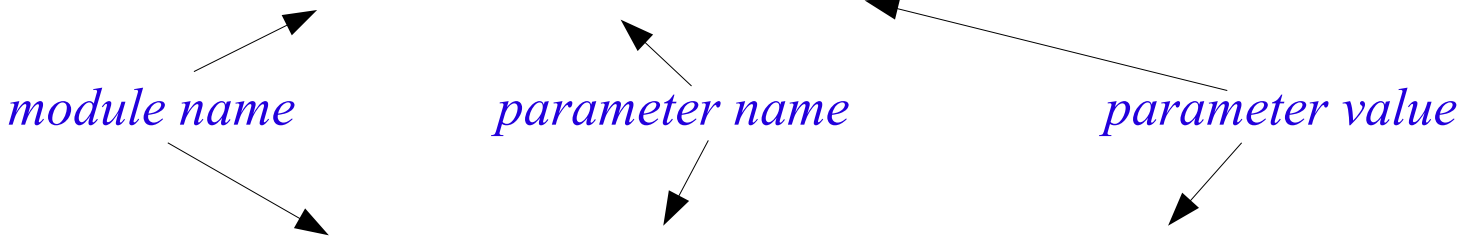
or

mpath=”/usr/lib/openser/modules/”

loadmodule “tm.so”

- Setting module parameter

modparam(“tm”, ”fr_inv_timer”, 20)



modparam(“tm”, ”fr_inv_timer_avp”, ”$avp(tm_timeout)”)

- Multi-module parameters

modparam("usrloc", "db_url",
"mysql:openser@localhost/openser")
modparam("auth_db", "db_url",
"mysql:openser@localhost/openser")

≡

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

Routes section

- the routes contain the routing logic
- there may be multiple routes
- there are multiple types of routes:
  - request route : route[n] {...}
  - reply route: onreply_route[m] {...}
  - failure route: failure_route[x] {...}
  - branch route: branch_route[y] {...}
  - error_route: error_route {...}
- in routes you can use :
  - function exported by core or modules
  - keywords and pre-defined values from core
  - pseudo-variables exported by core or modules

Message flow in OpenSER

- **route -** Request routing block. It contains a set of actions to be taken for SIP requests.

The main ‘route’ block identified by ‘route{...}’ or ‘route[0]{...}’ is executed for each SIP request.

The implicit action after execution of the main route block is to drop the SIP request. To send a reply or forward the request, explicit actions must be called inside the route block.

```
route {
    if(is_method("OPTIONS")) {
      # send reply for each options request
      sl_send_reply("200", "ok");
      exit();
    }
    route(1);
}
```

```
route[1] {
      # forward according to uri
      forward();
 }
```

- **branch_route** - request's branch routing block. It contains a set of actions to be taken for each branch of a SIP request. It is executed only by TM module after it was armed via t_on_branch("branch_route_index").

```
route {
    lookup("location");
    t_on_branch("1");
    if(!t_relay()) {
        sl_send_reply("500",
                    "relaying failed");
    }
}
```

```
branch_route[1] {
    if(uri=~"10\.10\.10].10") {
        # discard branches that go
        #  to 10.10.10.10
        drop();
    }
}
```

- **failure_route** - failed transaction routing block. It contains a set of actions to be taken each transaction that received only negative replies (>=300) for all branches. The ‘failure_route’ is executed only by TM module after it was armed via t_on_failure(”failure_route_index”).

Note that in ‘failure_route’ is processed the request that initiated the transaction, not the reply .

```
route {
    lookup("location");
    t_on_failure("1");
    if(!t_relay()) {
        sl_send_reply("500", "relaying failed");
    }
}
failure_route[1] {
    if(is_method("INVITE")) {
        # call failed - relay to voice mail
        t_relay_to_udp("voicemail.server.com","5060");
    }
}
```

- **onreply_route** - reply routing block. It contains a set of actions to be taken for SIP replies.

The main ‘onreply_route’ identified by ‘onreply_route {...}’ or ‘onreply_route[0] {...}’ is executed for all replies received.

Certain ‘onreply_route’ blocks can be executed by TM module for special replies. For this, the ‘onreply_route’ must be armed for the SIP requests whose replies should be processed within it, via t_on_reply(”onreply_route_index”).

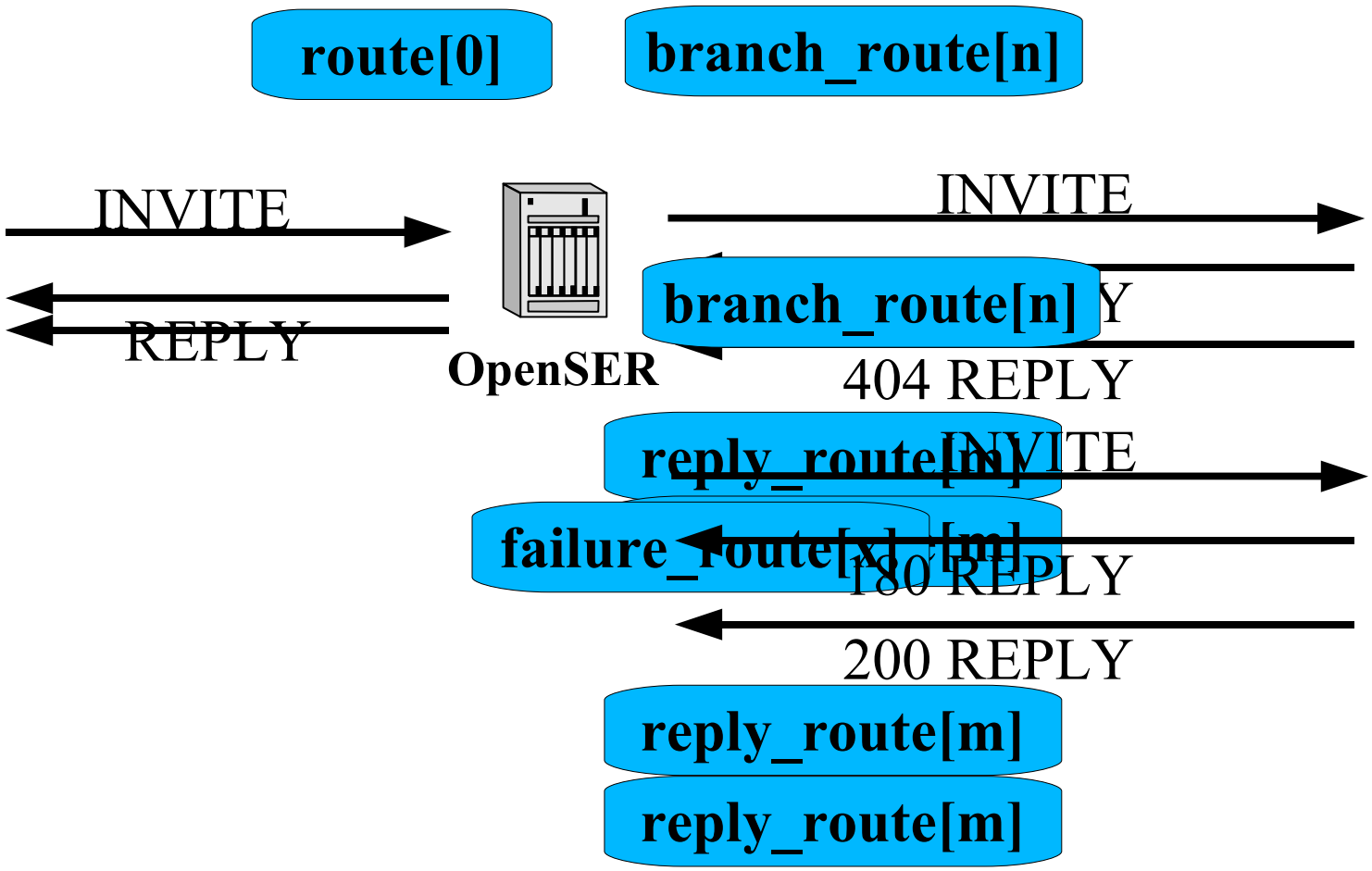
```
route {
    lookup("location");
    t_on_reply("1");
    if(!t_relay()) {
        sl_send_reply("500", "relaying failed");
    }
}
onreply_route[1] {
    if(status=~"1[0-9][0-9]") {
        log("provisional response\n");
    }
}
```

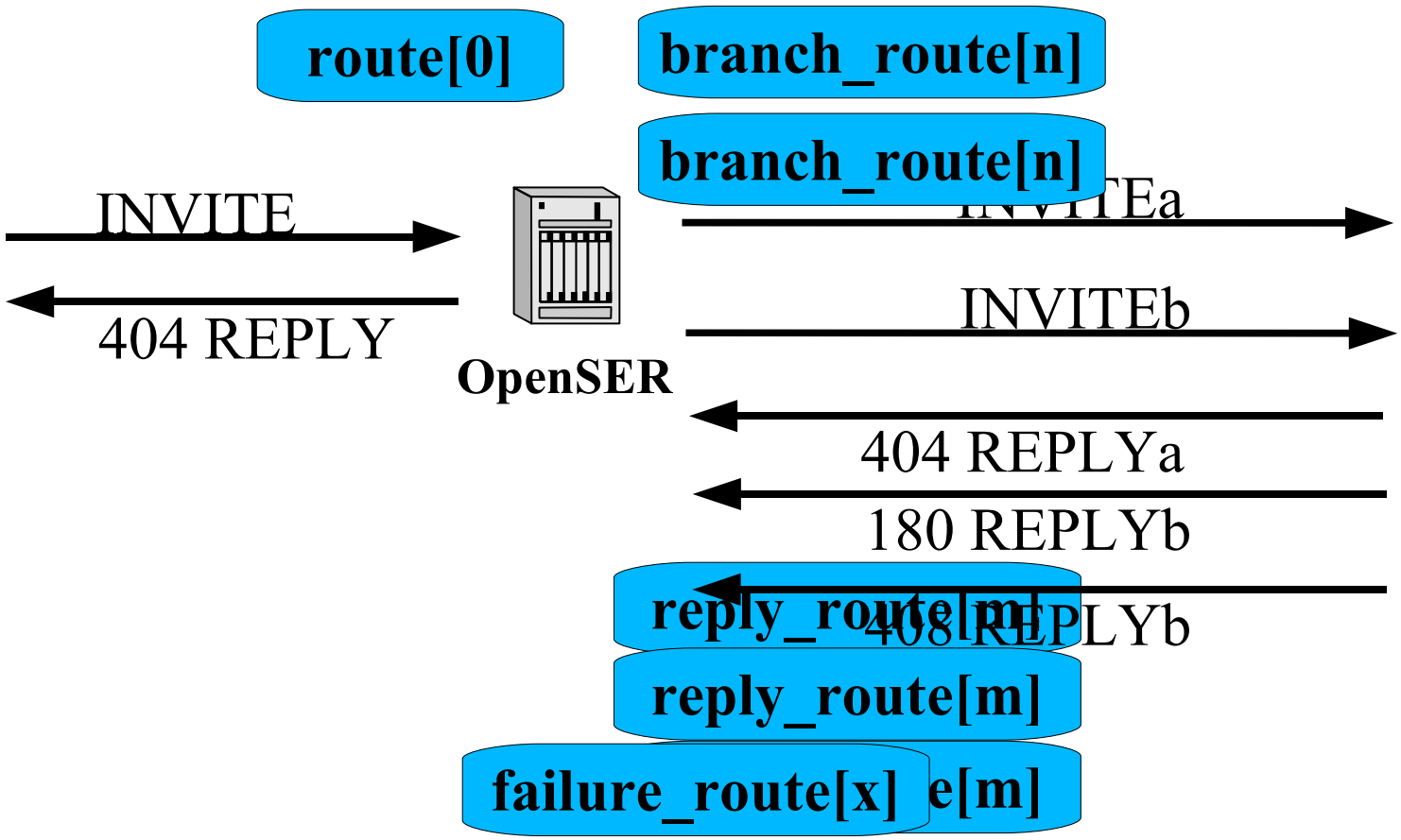
- **error_route** - the error route is executed automatically when a parsing error occurred during SIP request processing. This allow the administrator to decide what to do in case of error.

In error_route, the following pseudo-variables are available to get access to error details:

- $(err.class) - the class of error (now is '1' for parsing errors)
- $(err.level) - severity level for the error
- $(err.info) - text describing the error
- $(err.rcode) - recommended reply code
- $(err.rreason) - recommended reply reason phrase

```
error_route {
    xlog("--- error route class=$(err.class) level=$(err.level)
          info=$(err.info) rcode=$(err.rcode) rreason=$(err.rreason) ---
    \n");
    xlog("--- error from [$si:$sp]\n+++++\n$mb\n++++\n");
    sl_send_reply("$err.rcode", "$err.rreason");
    exit;
}
```


route[0]
branch_route[n]
INVITE
INVITE
REPLY
OpenSER
branch_route[n]
404 REPLY
reply_route[m]
INVITE
failure_route[x]
180 REPLY
200 REPLY
reply_route[m]
reply_route[m]


route[0]
branch_route[n]
branch_route[n]
INVITE
INVITEa
404 REPLY
INVITEb
OpenSER
404 REPLYa
180 REPLYb
reply_route[m]
408 REPLYb
reply_route[m]
failure_route[x]
e[m]

Scripting in OpenSER

- core functions
  - no restriction about the number of parameters (0....N)
  - parameters can be strings or integers
  - Ex:
    - forward(“udp:192.168.3.55:5060”);
    - setflag(1);

- module functions
  - can have maximum 2 parameters
  - they take only string parameters
  - Ex:
    - sl_send_reply("200","OK");
    - is_method("INVITE")

- **predefined values to be used in tests:**
  - **INET / INET6**
  - **TCP / TLS / UDP**
  - **myself** - it is a reference to the list of local IP addresses, hostnames and aliases that has been set in OpenSER configuration file.
- **keyword**
  - af / proto
  - dst_ip / dst_port
  - src_ip / src_port
  - method / status / retcode
  - uri / from_uri / to_uri

Ex:

```
if (proto==UDP  && af==INET) {...}
if (method==”INVITE” && uri=~”sip:[0-9]+@”) {...}
```

Pseudo-variable marker is character $

- **information reference (form message or openser)**

Ex: $ci – reference to message callid

$ru / $rU / $rd – reference to RURI / usernamer / domain of RURI

$ml – reference to message length

$rm – reference to request method

$pp – reference to process PID

$Tf – reference to current time formated as string

NOTE: most of the information reference pseudo variables are read-only; exceptions are:

- $ru* / $du - request and destination URI
- more in upcoming 1.3.0, e.g., $br - branch

- **Headers (also information reference)**

$(hdr(name)[N]) - represents the body of the N-th header identified by 'name'. If [N] is omitted then the body of the first header is printed. The first header is got when N=0, for the second N=1, a.s.o. To print the last header of that type, use -1. No white spaces are allowed inside the specifier (before }, before or after {, [,] symbols). When N='*', all headers of that type are printed.

*** *the above format is valid for 1.3, and N can be a PV as well*

- **AVPs**

- **$(avp(id)[N])** - represents the value of N-th AVP identified by 'id'.

- The 'id' can be:
  - "[si]:name" - name is the id of an AVP; 's' and 'i' specifies if the id is string or integer. If missing, it is considered to be string.
  - "name" - the name is an AVP alias

IMPORTANT: AVPs are hooked on messages or transactions. In stateful mode, same AVP can be accessed when processing any message related to the same transaction. Also, an AVP can have multiple values.

Script variables

$var(name) - refers to variables that can be used in configuration script, having integer or string value. This kind of variables are faster the AVPs, being referenced directly to memory location. The value of script variables persists over the processing of SIP messages, being specific per each OpenSER process.

- $var(a) = 2; -- sets the value of variable 'a' to integer '2'
- $var(a) = "2"; -- sets the value of variable 'a' to string '2'
- $var(a) = 3 + (7&(~2));
- $var(a) = "sip:" + $au + "@" + $fd; -- compose a value from authentication username and From URI domain

IMPORTANT: script variables exists only during script execution (including sub-routes). Once the script ended, they will be lost. Script variables are much, much faster than AVPs. A script variable can have only one value.

different transformations can be applied to Pseudo-Variables:

- String transformations
  - **{s.len}**
  - **{s.int}**
  - **{s.substr,offset,length}**
  - **{s.select,index,separator}**

  Ex: $var(len) = $(fU{s.len})
- URI transformations
  - **{uri.user}**
  - **{uri.host}**
  - **{uri.params}**

  Ex: $var(name) = $(avp(my_uri){uri.user})

- **Parameters list transformation**
  - **{param.value,name}** - return the value of parameter ‘name’

  Example:

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

  - **{param.name,index}** - return the name of parameter at position ‘index’.

  Example:

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

- **if** - IF-ELSE statement

Example of usage:

```
if(is_method("INVITE"))
{
    log("this sip message is an invite\n");
} else {
    log("this sip message is not an invite\n");
}
```

- **switch** - SWITCH statement - it can be used to test the value of a pseudo-variable.

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

```
# switch of R-URI username
    switch($rU)
    {
        case "101":
            log("destination number is 101\n");
        break;
        case "102":
            log("destination number is 102\n");
        break;
        case "103":
        case "104":
            log("destination number is 103 or 104\n");
        break;
        default:
            log("unknown destination number\n");
    }
}
```

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

- **Assignment** – these can be done like in C, via '=' (equal). The following pseudo-variables can be used in left side of an assignment:
  - AVPs - to set the value of an AVP
  - script variables - to set the value of a script variable
  - $ru - to set R-URI
  - $rd - to set domain part of R-URI
  - $rU - to set user part of R-URI
  - $du - to set dst URI
  - more on upcoming 1.3.0

$var(a) = 123;

String operations

For strings, ‘+’ is available to concatenate.

```
$var(a) = "test";
$var(b) = "sip:" + $var(a) + "@" + $fd;
```

- **Arithmetic operations**

  For numbers, one can use:

  + : plus

  - : minus

  / : divide

  * : multiply

  % : modulo

  | : bitwise OR

  & : bitwise AND

  ^ : bitwise XOR

  ~ : bitwise NOT

$var(a) = 4 + (7 & (~2));