Iptel.org

Optimizing the use of RTP proxy

Ladislav Andel
ladaan@iptel.org

31st August 2006

Contents

Chapter 1

Optimizing the use of RTP proxy

There are few considerations which have to be kept in mind to make this optimization successful. In the following sections of this document we will get through necessary capabilities needed on User Agent side and on the server side. The current development version of SER has been changed to facilitate the use of new features using Attribute Value Pairs (AVPs).

This approach makes the configuration of the SER server much easier and enables new possibilities for SIP end-points.

Before we start with the optimization we need to know what actually Attribute Value Pairs (AVPs) are. AVPs are a powerful tool for implementing services/preferences per uri/user/domain and they can be used directly from configuration script ser.cfg. It includes functions for interfacing database resources (loading/storing/removing), functions for swapping information between AVPs and SIP messages, function for testing/checking the value of an AVP.

These AVPs are accessible through **$** sign followed by a variable such as **$avp_attribute** which are loaded within configuration script. They can be accessible through GLOBAL, DOMAIN, USER or URI class. Furthermore, they are also divided into tracks such as FROM, TO or ALL. In other words there are two sets of AVPs, one for caller (FROM) and the other one for callee (TO). For example **$f.avp_attribute**, **$t.avp_attribute** or just simply **$avp_attribute**.

Within script can be defined for example
`$fu.avp = "abc"`
and checked as
`if ($foo == "abc") { ... }`.

More detailed info on AVPs reader can find at
<http://www.iptel.org/attribute_value_pairs_and_selects>

Basing on these AVPs SER knows how to treat each UA. For example what domain particular UA belongs to in multi-domain environment, what connectivity realm is in, what UA is behind NAT within SIP transaction, what language UA prefers etc.

1.1 SER NAT traversal extension

1.1.1 NAT flags - FLAG_NAT_UAS, FLAG_NAT_UAC

FLAG_NAT_UAC

The contact of UA will be marked as behind NAT if this flag is set before calling `save_contacts()` function in configuration script. The flag is set in contact row of location table in SER database.

Set by: `setflag("FLAG_NAT_UAC");` followed by `$f.nat = true;` which is later stored in Record-Route HF.

FLAG_NAT_UAS

This flag will be set by lookup function `lookup_contacts()` if a contact of UA is behind NAT

Checked by: `if (isflagset("FLAG_NAT_UAS"))`
If this condition is true the AVP attribute $t.nat = true is set and later stored in Record-Route HF.

These flags are declared at beginning of SER script:

Before modparam section

```
flags
   FLAG_NAT_UAC :1,
   FLAG_NAT_UAS :2;
```

If there are not specified flags with "flags: ...;" directive, flags will be 'automatically' registered by modparam. It is better to declare them, as ser.cfg gets easier readable.

Within modparam section must be these flags declared through registrar module

```
modparam("registrar", "save_nat_flag", "FLAG_NAT_UAC")
modparam("registrar", "load_nat_flag", "FLAG_NAT_UAS")
```

1.1.2 New AVP attributes in attr_types table

Connectivity realm membership

If User Agents are in the same connectivity realm then they can communicate with each other without using RTP proxy. It means they can reach each other directly (they are in the same network behind one NAT or reachable through VPN). This options can be set by typing the same name of connectivity realm in serweb's user account for both User Agents and avoid using RTP proxy.

For this functionality is introduced **connectivity_realm** AVP. This attribute is accessible within ser.cfg by **$f.connectivity_realm** and **$t.connectivity_realm**.

RTP proxy selection

If there is defined more RTP proxies in SER script then a user can choose the closest (geographically) RTP proxy from the given list in serweb's user account. For this reason is introduced attribute **$node**.

Support for Active/Passive role

For more detailed info, see 1.2.

1.2 UA capabilities

Important capabilities of UAs for NAT traversal optimization in ser.cfg are following:

- **Support for symmetric signalling**

  Currently, most of User Agents are supporting symmetric signalling. They would not work behind NAT if they do not support it.

  Function **force_rport()** has been used to keep the signalling symmetric.

- **Support for symmetric RTP**

  Ability to send RTP packets to source IP address and port of received RTP packet. UAs have to have this capability to work behind NAT.

- **Support for active/passive role**

  This role could also be called symmetric RTP, but it is based on active and passive attributes. For example when public UA is set to be passive then it waits until it receives the first RTP packet from its NATed peer and sends RTP to derived source IP address and port.

  For both symmetric RTP and active/passive direction attribute is introduced **sym_pass** AVP. This attribute is accessible within ser.cfg by **$f.sym_pass** and **$t.sym_pass**. Each user can select this option in serweb's user account if the user knows that its User Agent supports this feature and has public IP.

1.3 Configuration script ser.cfg

The configuration script, ser.cfg, is a part of every SER server and defines default behaviour. Server allows to users register with SER, proxy requests between User Agents, redirect SIP messages and lots and lots more. The possibilities of the script logic is almost endless.

Basically, it works this way. 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, otherwise a negative 404 reply is generated.

Requests in script execution are checked if they are from UA behind NAT and if yes, attributes **$f.nat** or **$t.nat** are set to *true.*

At the same time are loaded user's attributes from database where is specified if current users in dialog belongs to the same connectivity realm or if User Agent with public IP supports symmetric RTP and active/passive role.

Loading attributes from user_attrs table of SER database:

```
load_attrs("$fu", "$f.uid");
```
or
```
load_attrs("$tu", "$t.uid");
```

Similarly, there can be loaded attributes for URI as **$fr** or **$tr** from uri_attrs table or DOMAIN as **$fd** or **$td** from domain_attrs table by load_attrs function.

Requests outside served domain are always statefully forwarded.

Stateful forwarding is also required when a NAT is involved in SIP transactions and **Record-Route** header field is appended to SIP messages. AVP attributes can be stored in **Record-Route** for later transactions such as re-INIVITEs, BYE, CANCEL where could be also specified RTP proxy node used for the current call.

Note: Attributes loaded from database is necessary to assign to a different attribute in SER script if we want to store them in Record-Route HF. This is a reported bug, but before it is fixed, this note might save a little bit of time.

Figure 1.1 shows basically how requests are processed. Individual routes regarding NAT traversal are fully described later in this document.

1.4 Algorithm for using RTP proxy

1.4.1 route["send"]

Figure 1.2 shows **route("send")**. Within this route block requests are entering **route("nat.mangle")** and discovers if RTP proxy is needed for the initiating session. Route block **t_on_reply("nat.mangle")** is used only for replies passing SER proxy and also checks whether replies belonging to INVITE transactions need to be proxied by RTP proxy.

If any of UAs is behind NAT then SIP messages are record routed. In this route block we check if this flag was set and basing on this flag we store AVPs of both UAs and add this information to Record-Route header.

Then script forwards the message statefully.

1.4.2 route["nat.mangle"]

This route block does everything necessary to make UA behind NAT reachable. It includes rewriting contact header and forcing RTP proxy depending on User Agent's AVPs. The logic in this block is based on the value of FLAG_NAT_UAC and FLAG_NAT_UAS flags and User AVP attributes from database.

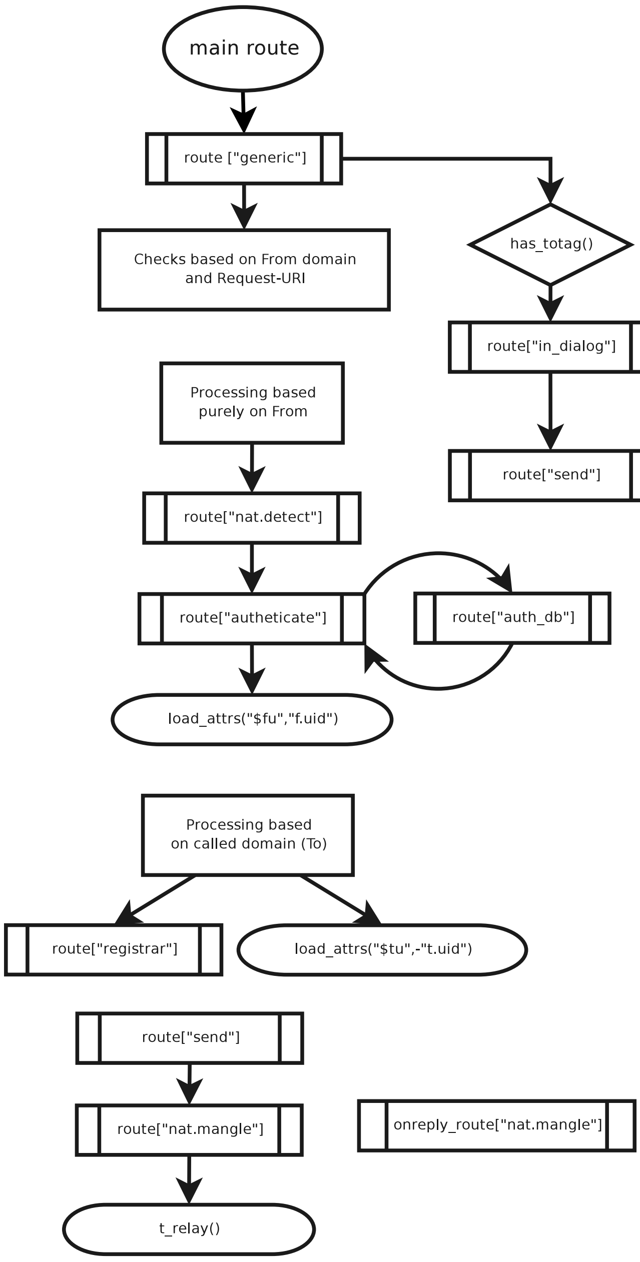
main route
route ["generic"]
Checks based on From domain and Request-URI
has_totag()
route["in_dialog"]
route["send"]
Processing based purely on From
route["nat.detect"]
route["autheticate"]
route["auth_db"]
load_attrs("$fu","f.uid")
Processing based on called domain (To)
route["registrar"]
load_attrs("$tu",-"t.uid")
route["send"]
route["nat.mangle"]
onreply_route["nat.mangle"]
t_relay()

Figure 1.1: Main route - request routing

Request processing based on AVPs:

- **$f.nat** , **$t.nat**

  Indicates whether Caller($f.nat) or Callee($t.nat) is behind NAT. In re-INVITEs it is also necessary to check for $tr.nat since $t.nat does not see the value in AVP set because it is stored in URI class.

- **$f.connectivity_realm** , **$t.connectivity_realm**

  If both AVPs match then it indicates that UAs are in the same connectivity realm and RTP proxy will not be involved. Media streams will stay in their realm. For example UAs behind the same NAT or UAs in Virtual Private Network (VPN). Re-INVITEs also needs a special treatment here within dialog. **$con_realm** attribute is set if this

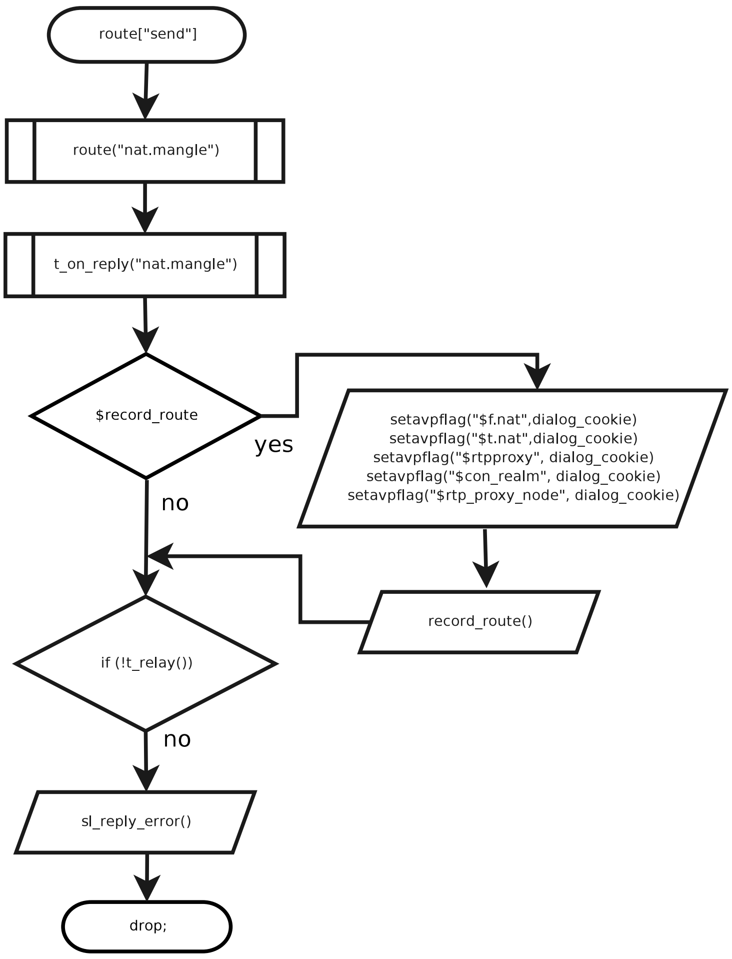
route["send"]
route("nat.mangle")
t_on_reply("nat.mangle")
$record_route
yes
setavpflag("$f.nat",dialog_cookie)
setavpflag("$t.nat",dialog_cookie)
setavpflag("$rtpproxy", dialog_cookie)
setavpflag("$con_realm", dialog_cookie)
setavpflag("$rtp_proxy_node", dialog_cookie)
no
record_route()
if (!t_relay())
no
sl_reply_error()
drop;

Figure 1.2: Send route - request routing

condition is true and is stored in **Record-Route** header for correct message processing within nat_mangle route.

- **$f.sym_pass** , **$t.sym_pass**

  Indicates if UA supports symmetric RTP and passive direction attribute. This AVP is important for UA with public IP and when only one NAT is involved in a SIP transaction. Because of the issue described here 1.3 there are assigned new attributes **$f.comedia** and **$t.comedia**. These new attributes are Record-Routed if needed.

- **$node**

  This attribute is loaded from database and says which RTP proxy node should be used for the session. **$rtp_proxy_node** attribute is assigned for storing in **Record-Route**. The reason is described in the note 1.3.

RTP proxies are defined in **modparam** line at the beginning of the script.

For example:
```
modparam("nathelper", "rtpproxy_sock", "unix:/var/run/rtpproxy.sock udp:5.6.7.8:2
```

Each proxy is chosen by index starting from default 0 selected per user's choice.

Figures 1.3 shows the actual algoritm.

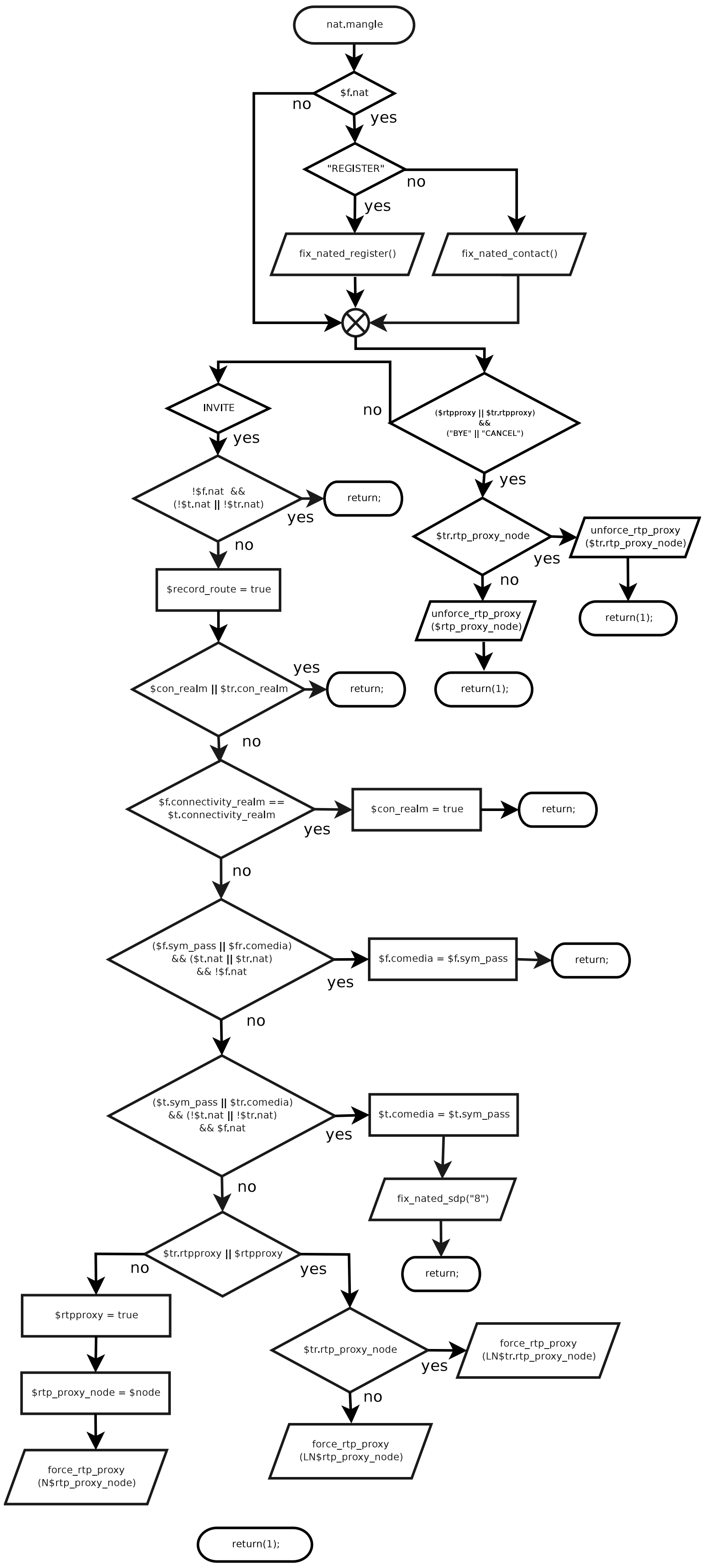
nat.mangle
$f.nat
no
yes
"REGISTER"
no
yes
fix_nated_register()
fix_nated_contact()
INVITE
no
($rtpproxy || $tr.rtpproxy) && ("BYE" || "CANCEL")
yes
!$f.nat && (!$t.nat || !$tr.nat)
return;
yes
no
$tr.rtp_proxy_node
unforce_rtp_proxy ($tr.rtp_proxy_node)
yes
no
unforce_rtp_proxy ($rtp_proxy_node)
return(1);
return(1);
$record_route = true
$con_realm || $tr.con_realm
yes
return;
no
$f.connectivity_realm == $t.connectivity_realm
$con_realm = true
return;
yes
no
($f.sym_pass || $fr.comedia) && ($t.nat || $tr.nat) && !$f.nat
$f.comedia = $f.sym_pass
return;
yes
no
($t.sym_pass || $tr.comedia) && (!$t.nat || !$tr.nat) && $f.nat
$t.comedia = $t.sym_pass
yes
fix_nated_sdp("8")
return;
no
$tr.rtpproxy || $rtpproxy
no
yes
$rtpproxy = true
$rtp_proxy_node = $node
force_rtp_proxy (N$rtp_proxy_node)
$tr.rtp_proxy_node
force_rtp_proxy (LN$tr.rtp_proxy_node)
yes
no
force_rtp_proxy (LN$rtp_proxy_node)
return(1);

Figure 1.3: NAT.mangle route

1.4.3 onreply_route["nat.mangle"]

Similar approach is with responses passing the SIP proxy. Note that SIP messages must pass SIP proxy in purpose to have successful NAT traversal. In other words they are record routed.

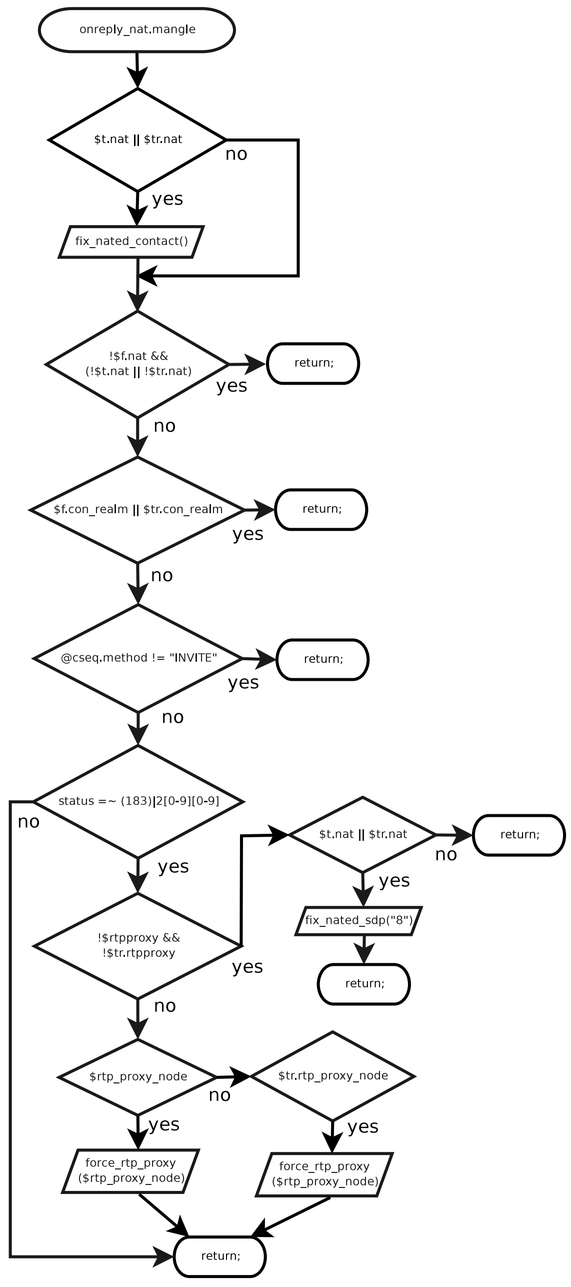

Figure 1.4: Onreply_NAT.mangle route - 2nd part

1.5 Example scenarios

Following figures show possible scenarios in real world deployment.

1.5.1 No NATs involved

Figure 1.5 shows when both UAs have public IP addresses. There is no need for any NAT mangling, because there will be no NAT detected.

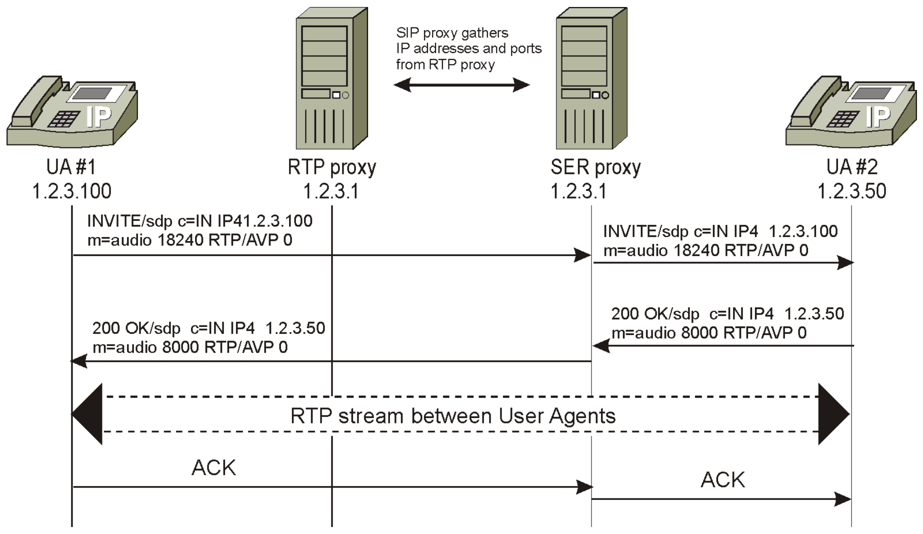

Figure 1.5: No NATs involved

1.5.2 One UA behind NAT, public UA with passive support

Following Figures 1.6 and 1.7 show that we can avoid using RTP proxy because public UA supports symmetric RTP and active/passive role. The AVP **sym_pass** for public UA is set to *true*.

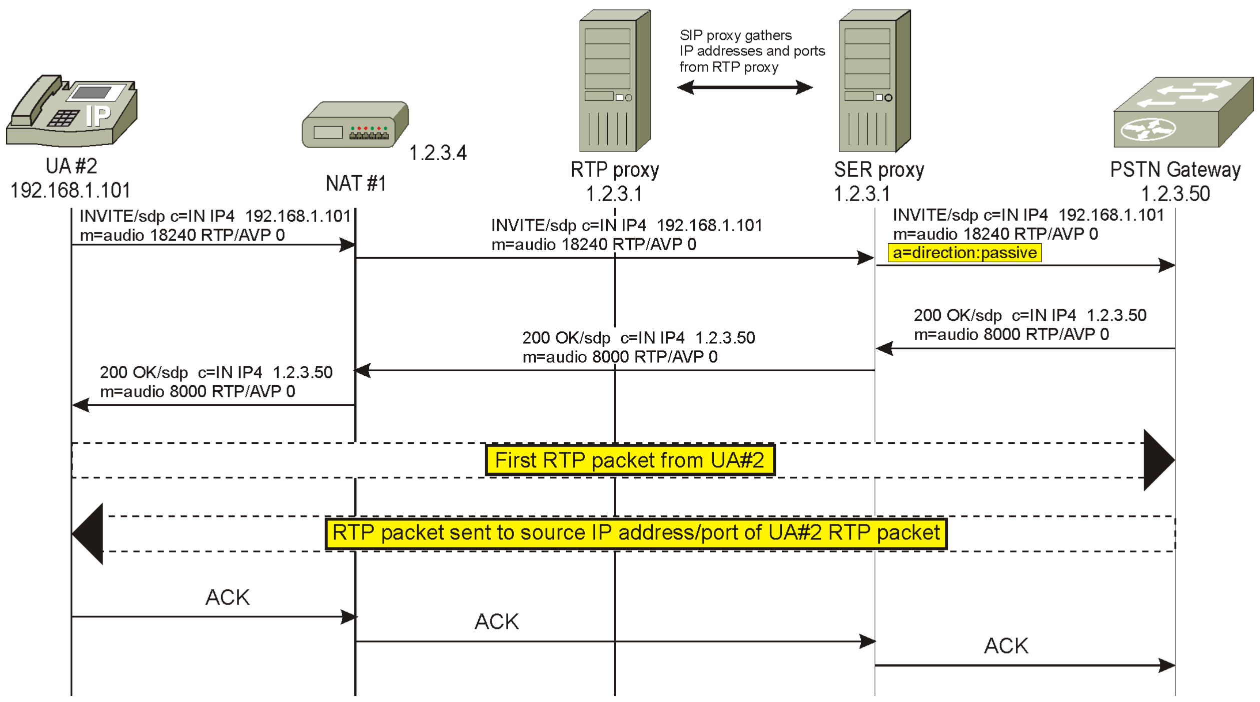

Figure 1.6: INVITE sent from NATed User Agent

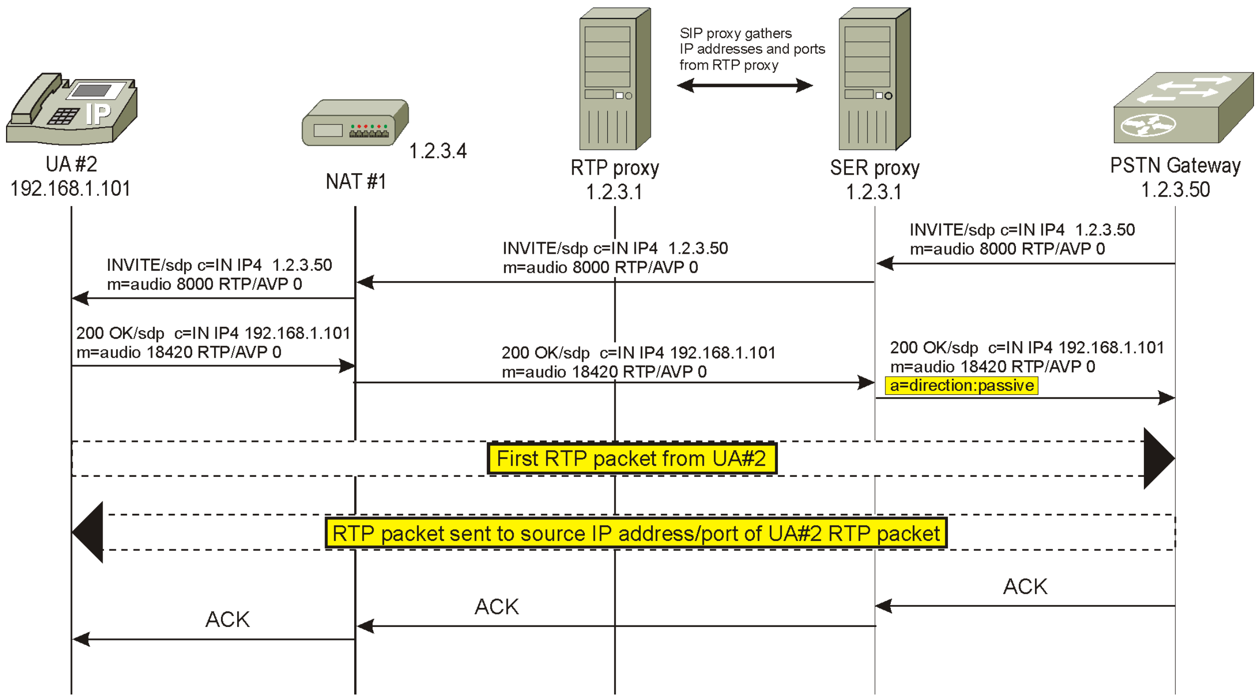

Figure 1.7: INVITE sent from public User Agent

1.5.3 One UA behind NAT, public UA without passive role support

Figure 1.8 shows when RTP proxy is needed. In this case public UA does not support symmetric RTP and active/passive role. The AVP **sym_pass** equals to *false* as default.

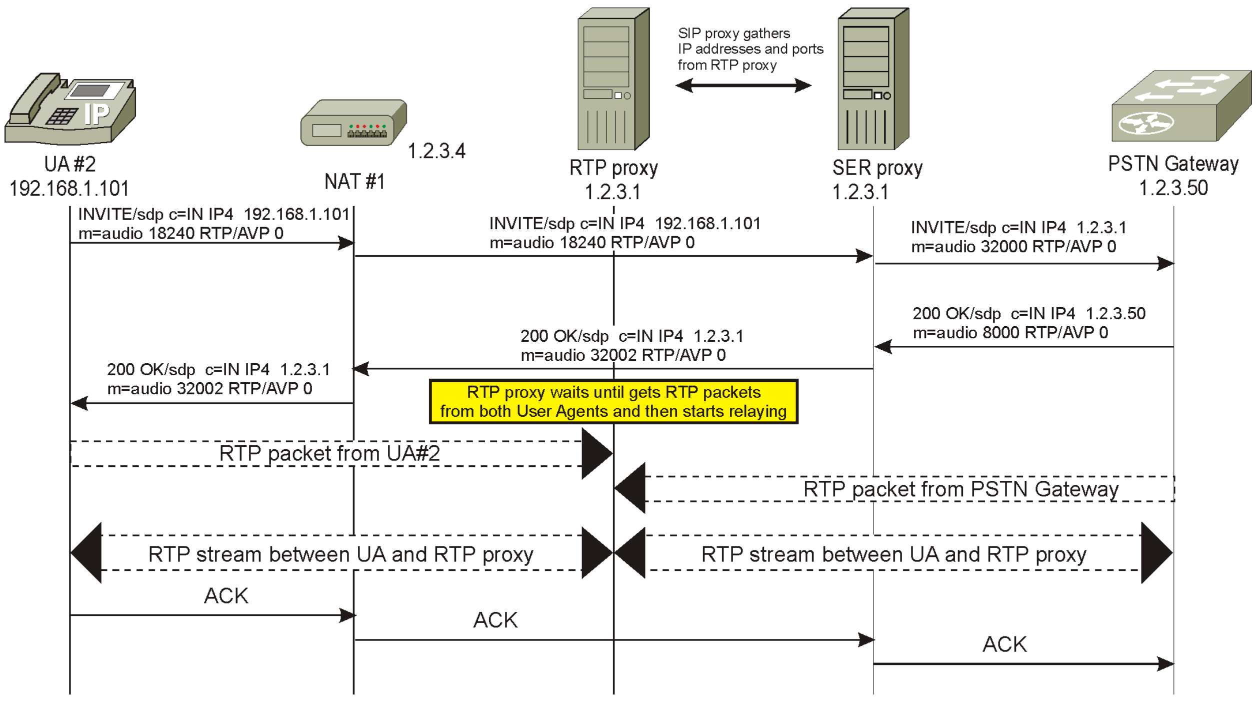
SIP proxy gathers IP addresses and ports from RTP proxy
UA #2
192.168.1.101
NAT #1
1.2.3.4
RTP proxy
1.2.3.1
SER proxy
1.2.3.1
PSTN Gateway
1.2.3.50
INVITE/sdp c=IN IP4 192.168.1.101
m=audio 18240 RTP/AVP 0
INVITE/sdp c=IN IP4 192.168.1.101
m=audio 18240 RTP/AVP 0
INVITE/sdp c=IN IP4 1.2.3.1
m=audio 32000 RTP/AVP 0
200 OK/sdp c=IN IP4 1.2.3.50
m=audio 8000 RTP/AVP 0
200 OK/sdp c=IN IP4 1.2.3.1
m=audio 32002 RTP/AVP 0
200 OK/sdp c=IN IP4 1.2.3.1
m=audio 32002 RTP/AVP 0
RTP proxy waits until gets RTP packets from both User Agents and then starts relaying
RTP packet from UA#2
RTP packet from PSTN Gateway
RTP stream between UA and RTP proxy
RTP stream between UA and RTP proxy
ACK
ACK
ACK

Figure 1.8: One UA behind NAT, public UA without passive role support

1.5.4 Two UAs behind the same NAT

Figure 1.9 shows when two UAs are behind the same NAT and are in the same connectivity realm. Both UAs have the same **connectivity_realm** AVP set to **realm1** which means RTP packets are sent directly between them.

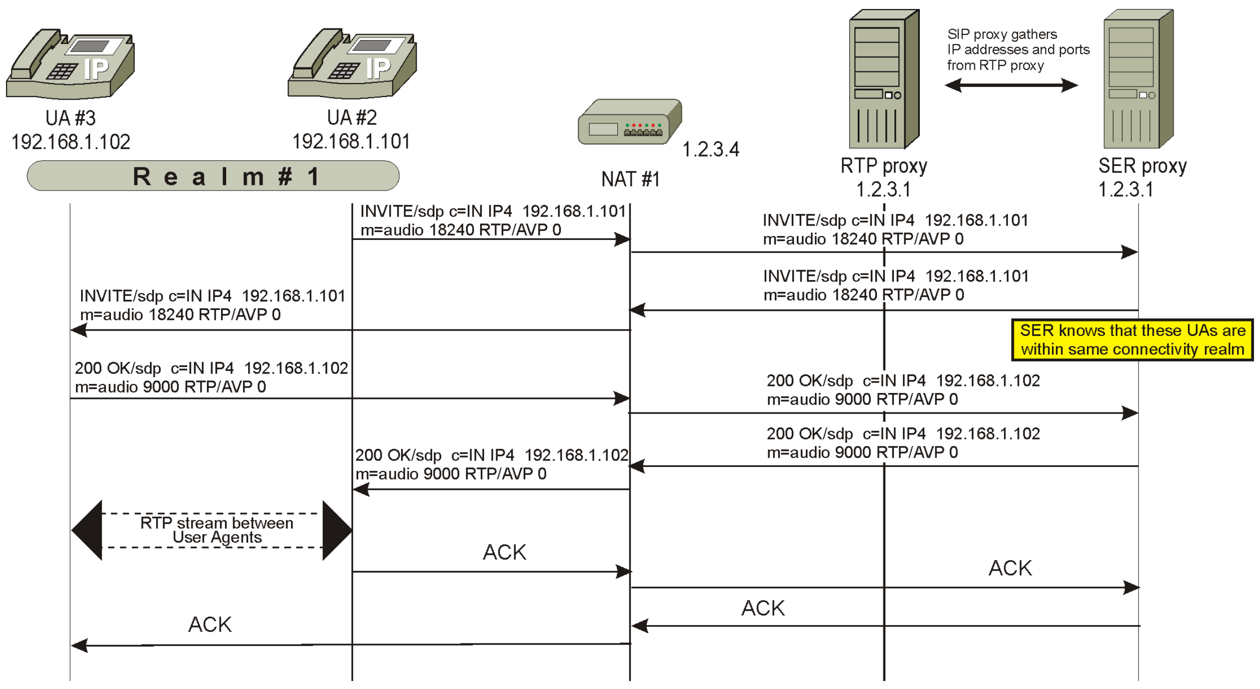
UA #3
192.168.1.102
UA #2
192.168.1.101
Realm #1
NAT #1
1.2.3.4
RTP proxy
1.2.3.1
SER proxy
1.2.3.1
SIP proxy gathers IP addresses and ports from RTP proxy
INVITE/sdp c=IN IP4 192.168.1.101 m=audio 18240 RTP/AVP 0
INVITE/sdp c=IN IP4 192.168.1.101 m=audio 18240 RTP/AVP 0
INVITE/sdp c=IN IP4 192.168.1.101 m=audio 18240 RTP/AVP 0
INVITE/sdp c=IN IP4 192.168.1.101 m=audio 18240 RTP/AVP 0
SER knows that these UAs are within same connectivity realm
200 OK/sdp c=IN IP4 192.168.1.102 m=audio 9000 RTP/AVP 0
200 OK/sdp c=IN IP4 192.168.1.102 m=audio 9000 RTP/AVP 0
200 OK/sdp c=IN IP4 192.168.1.102 m=audio 9000 RTP/AVP 0
200 OK/sdp c=IN IP4 192.168.1.102 m=audio 9000 RTP/AVP 0
RTP stream between User Agents
ACK
ACK
ACK
ACK

Figure 1.9: Two User Agents behind the same NAT

1.5.5 Two UAs behind two NATs but within same connectivity realm

This approach will work when a network tunnel(e.g. VPN, IPsec) is between UAs and of course **connectivity_realm** AVP is set to **realm3**. Figure 1.10 shows this scenario.

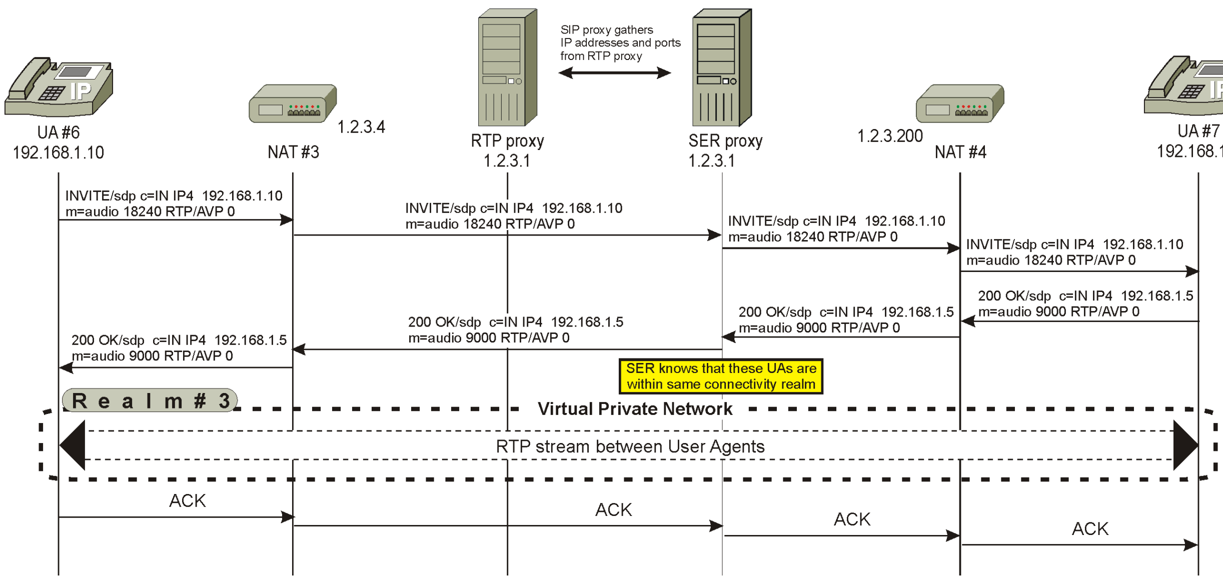
SIP proxy gathers IP addresses and ports from RTP proxy
UA #6 192.168.1.10
NAT #3
1.2.3.4
RTP proxy 1.2.3.1
SER proxy 1.2.3.1
1.2.3.200
NAT #4
UA #7 192.168.1.5
INVITE/sdp c=IN IP4 192.168.1.10 m=audio 18240 RTP/AVP 0
200 OK/sdp c=IN IP4 192.168.1.5 m=audio 9000 RTP/AVP 0
SER knows that these UAs are within same connectivity realm
R e a l m # 3
Virtual Private Network
RTP stream between User Agents
ACK

Figure 1.10: Two UAs behind two NATs but within same connectivity realm

1.5.6 Two UAs behind two NATs (different connectivity realms)

Figure 1.11 shows when RTP proxy has to be used at all times.

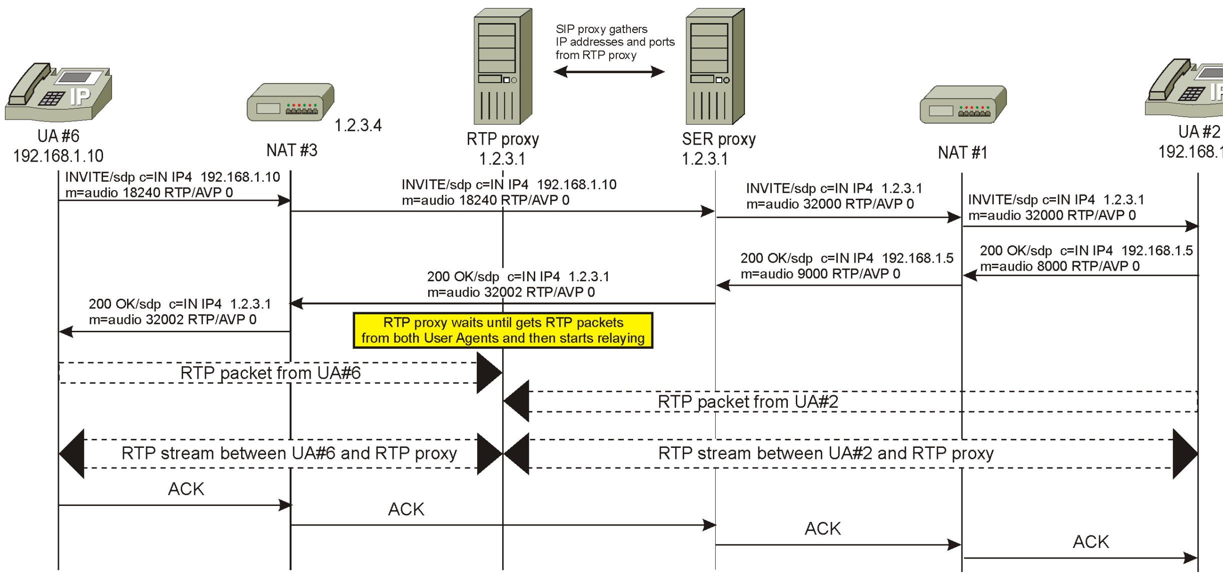
SIP proxy gathers IP addresses and ports from RTP proxy
UA #6 192.168.1.10
NAT #3
1.2.3.4
RTP proxy 1.2.3.1
SER proxy 1.2.3.1
NAT #1
UA #2 192.168.1.5
INVITE/sdp c=IN IP4 192.168.1.10 m=audio 18240 RTP/AVP 0
INVITE/sdp c=IN IP4 1.2.3.1 m=audio 32000 RTP/AVP 0
200 OK/sdp c=IN IP4 192.168.1.5 m=audio 8000 RTP/AVP 0
200 OK/sdp c=IN IP4 192.168.1.5 m=audio 9000 RTP/AVP 0
200 OK/sdp c=IN IP4 1.2.3.1 m=audio 32002 RTP/AVP 0
RTP proxy waits until gets RTP packets from both User Agents and then starts relaying
RTP packet from UA#6
RTP packet from UA#2
RTP stream between UA#6 and RTP proxy
RTP stream between UA#2 and RTP proxy
ACK

Figure 1.11: Two User Agents behind two NATs (different connectivity realms)

Chapter 2

Implementation Details

2.1 Augmented ser.cfg

In this section are described route blocks necessary for NAT traversal in configuration script.

At the beginning of SER script, exactly after load_module section is necessary to place lines

These are recommended:

```
flags
   FLAG_NAT_UAC :1,
   FLAG_NAT_UAS :2;
```

This one is necessary:

```
avpflags dialog_cookie;
```

In modparam section it is necessary place following:

```
   modparam("registrar", "save_nat_flag", "FLAG_NAT_UAC")
   modparam("registrar", "load_nat_flag", "FLAG_NAT_UAS")
```

In general route we must check if TO user is behind NAT while initiating a call.

```
if (!lookup_contacts("location")) {
    sl_reply("483", "Temporarily Unavailable");
    break;
}
if (isflagset("FLAG_NAT_UAS"))
    $t.nat = true;
else
    $t.nat = false;
```

lookup_contacts() function load the contact details of callee including our NAT flag.

2.1.1 route["nat.detect"]

NAT detection is executed early in the script and its purpose is to detect whether the UAC is behind NAT. This block does no NAT mangling, this is postponed until the end of the script. This block only sets FLAG_NAT_UAC flag if the UAC is behind NAT and assigns a boolean value to **$f.nat** attribute.

```
366 route["nat.detect"]
367 {
368
369     if (src_ip == 127.0.0.1) return;
370
371     if (is_present_hf("^Record-Route:")) return;
372
373     if (nat_uac_test("3")) {
374         force_rport();
375         setflag("FLAG_NAT_UAC");
376         $f.nat = true;
377     }
378     else $f.nat = false;
379     return;
380 }
```

Line 369
Skip spiralling requests.

Line 371
Skip requests that already passed a proxy.

Lines 373 - 378
In nat_uac_test if flag 1 is set, the "received" test is used – IP address in Via is compared against source IP address of signalling. If flag 2 is set, Contact header field is searched for occurrence of RFC1918 addresses. Both flags can be bitwise combined, the test returns true if any of the tests identified a NAT.

Function `force_rport()` will include `"received="` and `"rport="` parameters in Via header and the content of parameters will be the source IP address and port of the received request. The function `set_flag("FLAG_NAT_UAC")` will mark the UA as behind NAT and sets the AVP to `$f.nat = true`.
If the condition is not true then sets NAT AVP to `$f.nat = false`.

Line 379
Returns back to where this route block was called from.

2.1.2 route["send"]

This route block forward requests and does NAT mangling if necessary before forwarding.

```
route["send"]
342 {
343
344    route("nat.mangle");
345    t_on_reply("nat.mangle");
346
347     if ($record_route) {
348         setavpflag("$f.nat", "dialog_cookie");
349         setavpflag("$t.nat", "dialog_cookie");
350         setavpflag("$rtpproxy", "dialog_cookie");
351         setavpflag("$con_realm", "dialog_cookie");
352         setavpflag("$rtp_proxy_node", "dialog_cookie");
353         setavpflag("$f.comedia", "dialog_cookie");
354         setavpflag("$t.comedia", "dialog_cookie");
355
356         record_route();
357     }
358
359     if (!t_relay()) {
360         sl_reply_error();
361         drop;
362     }
363 }
```

Line 344
Before relaying NAT mangling is performed.

Lines 345
NAT mangling for replies.

Lines 347 - 357
If record route AVP was set within nat.mangle route block then `Record-Route` header is appended with `"avp="` parameter including the value of AVPs **$f.nat, $t.nat, $rtpproxy, $con_realm, $rtp_proxy_node, $f.comedia** and **$t.comedia**.

Lines 359 - 362
Message is forwarded statefully.

2.1.3 onreply_route[”nat.mangle”]

This route block does everything necessary to make UAS behind NAT reachable, that includes rewriting contact of SIP replies and forcing RTP proxy if necessary. The logic in this block is based on the value of FLAG_NAT_UAC and FLAG_NAT_UAS flags, AVP attributes loaded from database and recovered from Record-Route HF.

```
382 onreply_route["nat.mangle"]
383 {
384
385     if ($t.nat || $tr.nat) {
386         xlog("L_ERR","LXLOG: Onreply - fixing NATed contact");
387         fix_nated_contact();
388     }
389
390     if (!$f.nat && (!$t.nat || !$tr.nat)) return;
391
392     if ($con_realm || $tr.con_realm) return;
393
394     if (@cseq.method != "INVITE") return;
395     if ((status =~ "(183)|2[0-9][0-9]") && !search("^Content-Length: 0")) {
396
397         if (!$rtpproxy && !$tr.rtpproxy) {
398
399             if ($t.nat || $tr.nat) {
400                 fix_nated_sdp("8");
401                 return;
402             }
403             else return;
404         }
405
406         if ($rtp_proxy_node)
407             eval_push("x:N%$rtp_proxy_node");
408
409         else if ($tr.rtp_proxy_node)
410             eval_push("x:N%$tr.rtp_proxy_node");
411
412         force_rtp_proxy("@eval.pop[-1]");
413
414     }
415 return;
416 }
```

Lines 385 - 388
If recipient is behind NAT then the Contact in reply message is rewritten by source IP address/port.

Line 390
Checks if either UA is behind NAT.

Line 392
If UAs within transaction are in the same connectivity realm then there is no need to do anything with SDP content and SIP request is returned back to **"send"** route block. Otherwise other checks are necessary. The value is taken from **Record-Route** header field

Line 394
Checks if the reply belongs to INVITE transaction, otherwise returns back to **"send"** route block.

Lines 397 - 414
Checks if the reply is 183 or 2xx and also if the SIP message contents SDP body. If yes then it proceeds to the check if AVP attribute **$rtpproxy** was set to *true* in nat.mangle route. **$tr.rtpproxy** is checked for occurrence of this value in TO track of URI class. (This usually happen with re-INVITEs or BYE when roles of caller and callee are swapped) The value reads from **"avp="** parameter in **Record-Route** header field.

If the condition is *false* then we know that one of UA supports symmetric RTP and active/passive role. If it is recipient of this reply then we call **fix_nated_sdp("8")** function which append new line **a=direction:passive** in SDP document of 2xx reply and return back to **"send"** route block. Otherwise returns back directly.

If the condition is *true* RTP proxy is forced and picked the RTP proxy which was selected by caller.

Following function will just store a string "N0", "N1" or "N2" etc. which is combined with value of **$rtp_proxy_node** attribute. This attribute carries the number(index) of the particular RTP proxy.

```
eval_push("x:N%$rtp_proxy_node");
```

Select **@eval.pop[-1]** returns the stored string by **eval_push()** function.

```
force_rtp_proxy("@eval.pop[-1]");
```

Line 415
Returns back to **"send"** route block.

2.1.4 route["nat.mangle"]

This route block does everything necessary to make UAS behind NAT reachable, that includes rewriting contact and forcing RTP proxy if necessary. The logic in this block is based on the value of FLAG_NAT_UAC and FLAG_NAT_UAS flags

```
418 route["nat.mangle"]
419 {
420
421     if ($f.nat) {
422         if (method == "REGISTER") {
423             fix_nated_register();
424         }
425         else {
426             fix_nated_contact();
427         }
428     }
429
430     if (($tr.rtpproxy || $fr.rtpproxy) &&
    (method == "BYE" || method == "CANCEL")) {
431
432         if ($tr.rtp_proxy_node)
433            unforce_rtp_proxy("$tr.rtp_proxy_node");
434         else
435            unforce_rtp_proxy("$rtp_proxy_node");
436
437     }
438     else if (method == "INVITE") {
439         if (!$f.nat && (!$t.nat || !$tr.nat)) return;
440
441         $record_route = true;
442
443         if ($con_realm || $tr.con_realm) return;
444         if ($f.connectivity_realm == $t.connectivity_realm) {
445             $con_realm = true;
446             return;
447         }
448
449         if (($f.sym_pass || $fr.comedia) && ($t.nat || $tr.nat) && !$f.nat) {
450             $f.comedia = $f.sym_pass;
451             return;
452         }
453         if (($t.sym_pass || $tr.comedia) && $f.nat && (!$t.nat || !$tr.nat)) {
454             $t.comedia = $t.sym_pass;
455             fix_nated_sdp("8");
456             return;
457         }
458
459         if ($tr.rtpproxy || $rtpproxy) {
460             if ($tr.rtp_proxy_node)
461                 eval_push("x:LN%$tr.rtp_proxy_node");
462             else
463                 eval_push("x:LN%$rtp_proxy_node");
464      }
465         else {
466             $rtpproxy = true;
467             $rtp_proxy_node = $node;
468
469             eval_push("x:N%$rtp_proxy_node");
470         }
471         force_rtp_proxy("@eval.pop[-1]");
472     }
473 return 1;
474 }
```

Lines 421 - 428
If INVITE is received from UA behind NAT then replace IP address/port in Contact header with source IP address/port of the SIP request. Otherwise if it is REGISTER then **fix_nated_register()** stores source IP address/port into location database and appends "received" parameter to Contact header.

Lines 430 - 437
If RTP proxy has been forced and the request is BYE or CANCEL then unforce session from the particular RTP proxy which had been selected when initiating call.

Line 438
If the request is INVITE do several checks if RTP proxy is necessary.

Line 439
If any of UAs in transaction is behind NAT then go on, otherwise return back to **"send"** route block.

Line 441
Record routing is forced if either UA is behind NAT.

Lines 443-447
If UAs within transaction are in the same connectivity realm then there is no need to do anything with SDP content and SIP request is returned back to **"send"** route block. Otherwise other checks are necessary. Why is also used **$con_realm** see 1.3

Lines 449 - 457
Checks if either UA supports symmetric RTP and active/passive role. If any UA does then it checks if INVITE is coming from the UA behind NAT and append new line `a=direction:passive` at the end of SDP document by calling `fix_nated_sdp("8")` function. Then it return to **"send"** route block. Otherwise it returns without appending direction attribute.

Lines 459 - 464
This section is for re-INVITEs : If RTP proxy was forced during initiating a new call then here is performed a check for attributes **$rtpproxy** and **$rtp_proxy_node**. This will use the same RTP proxy as was chosen while initiating the call. Parameter L performs lookup which only rewrite SDP when corresponding session is already existing in RTP proxy. Parameter N followed by AVP attribute is stored index(integer) of a particular RTP proxy defined at the beginning of this script in modparam.

Lines 465 - 472
For initial call: sets the **$rtpproxy** attribute to *true* and assign different AVP attribute for **$rtp_proxy_node** to make sure it is stored in Record-Route HF. **$node** comes from user_attrs table (reported at bugs.sip-server.net).

2.2 Data in database

User attributes(AVPs) used within configuration script are stored in **user_attrs** table of SER database. Among them are stored NAT traversal key attributes.

2.2.1 Adding new user attributes - Attribute Value Pairs

```
./ser_db add user_attrs [column=value] ...
```

2.2.2 Columns in user_attrs table

flags - If it is set to 1 the AVP attribute takes effect in ser.cfg.
type - 0 for integer, 2 for string
uid - user ID as it is in table **credentials**
name - name of AVP attribute
value - value of AVP attribute

2.2.3 Showing user_attrs table values

```
./ser_db -t show user_attrs

+-------+------+-------+---------------------+--------------------+
| flags | type | uid   | value               | name               |
+-------+------+-------+---------------------+--------------------+
| 0     | 2    | UA1   | 2006-04-08 12:12:00 | datetime_created   |
| 0     | 2    | UA2   | 2006-04-09 16:00:11 | datetime_created   |
| 0     | 2    | UA3   | 2006-04-13 09:51:36 | datetime_created   |
| 1     | 2    | UA3   | realm1              | connectivity_realm |
| 1     | 2    | UA2   | realm1              | connectivity_realm |
| 1     | 2    | UA6   | realm3              | connectivity_realm |
| 1     | 2    | UA7   | realm3              | connectivity_realm |
| 1     | 0    | UA1   | 1                   | sym_pass           |
| 1     | 2    | UA6   | 0                   | node               |
| 1     | 2    | UA7   | 1                   | node               |
+-------+------+-------+---------------------+--------------------+
```

2.2.4 Summary - Using Attribute Value Pairs

Since AVPs are set as we want, now the important part is using them in SIP transactions. Each set of AVPs is loaded within configuration script. There is one set for caller (**$f.** attributes) and one set for callee (**$t.** attributes). Within script **load_attrs("$fu","f.uid")** loads callers attributes and **load_attrs("$tu","t.uid")** loads callee attributes.

Then we can check the value of AVP for example by:

```
if ($fu.sym_pass == 1) {
...
}
else {
...
}
```

Conclusion

NAT devices bring a lot of drawbacks to SIP communication which was described earlier in this document. RTP proxy solves the worst scenarios on the Internet but brings few drawbacks too. The goal of this document is to optimize the use of RTP proxy based on knowledge of UA capabilities. New version of SIP Express Router introduced new features such as Attribute Value Pairs which makes the optimization possible. User Agent's AVPs are stored in database and fetched within SIP message processing in configuration script **ser.cfg**. Particularly, SER knows which UA is behind NAT and what capabilities supports. Depending on these AVPs SER determine whether RTP proxy is needed for the session or not. The following Table 2.1 shows what scenarios were optimized in terms of using RTP proxy.

Table 2.1: Results of optimization

Using RTP proxy	before	optimized
No NAT	No	No
One UA behind NAT and public UA with passive role support	Yes	No
One UA behind NAT and public UA with no passive role support	Yes	Yes
Two UAs behind the same NAT within the same connectivity realm	Yes	No
Two UAs behind the two NATs but within the same connectivity realm	Yes	No
Two UAs behind the two NATs but in different realms	Yes	Yes

The importance of avoiding the use of RTP proxy will come especially in large-scale deployments. RTP proxy is becoming a quite problem for Voice over IP service providers with more than 10000 subscribers. Therefore the optimization will save a lot on resources such as network bandwidth, performance of physical servers and so on.

Furthermore, it is possible to pick a particular RTP relay from a given set of relays based on the preferences of the caller from serweb interface or through command line tools. This will lower delay between User Agents to minimum if a correct RTP proxy is chosen depending on geographical location.

Appendix A

Configuration script - ser.cfg

```
#
#
#
# rtp proxy optimizing config script
#

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

debug=3        # debug level (cmd line: -dddddddddd)
fork=yes
log_stderror=no  # (cmd line: -E)
#memlog=5 # memory debug log level
#log_facility=LOG_LOCAL0 # sets the facility used for logging (see syslog(3))
children = 1

check_via=no        # (cmd. line: -v)
dns=no           # (cmd. line: -r)
rev_dns=no      # (cmd. line: -R)

listen=127.0.0.1:5060
listen=1.2.3.4:5060  # your IP address

loadmodule "/usr/local/lib/ser/modules/maxfwd.so"

# Database drivers
loadmodule "/usr/local/lib/ser/modules/mysql.so"

loadmodule "/usr/local/lib/ser/modules/sl.so"

# AVP
loadmodule "/usr/local/lib/ser/modules/avp.so"
loadmodule "/usr/local/lib/ser/modules/avp_db.so"
loadmodule "/usr/local/lib/ser/modules/avpops.so"

loadmodule "/usr/local/lib/ser/modules/domain.so"

# Authentication
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"

# Registrar
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"

loadmodule "/usr/local/lib/ser/modules/gflags.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"

# NAT traversal
loadmodule "/usr/local/lib/ser/modules/nathelper.so"

loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"
loadmodule "/usr/local/lib/ser/modules/xlog.so"

# Management
loadmodule "/usr/local/lib/ser/modules/xmlrpc.so"

loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"

loadmodule "/usr/local/lib/ser/modules/acc_syslog.so"
loadmodule "/usr/local/lib/ser/modules/eval.so"

  # ----------------- setting script FLAGS -----------------------------
  flags
      FLAG_NAT_UAC    :1,
      FLAG_NAT_UAS    :2;

avpflags dialog_cookie;

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

modparam("usrloc|domain|avp_db|uri_db|auth_db|gflags|msilo|speeddial|acc_db", "db_url", "mysql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("domain", "db_mode", 1)
modparam("domain", "load_domain_attrs", yes)
modparam("gflags", "load_global_attrs", yes)
modparam("usrloc", "db_mode", 1)

modparam("registrar", "save_nat_flag", "FLAG_NAT_UAC")
modparam("registrar", "load_nat_flag", "FLAG_NAT_UAS")
# modparam("usrloc|registrar", "use_domain", 1)

# modparam("nathelper", "natping_interval", 30)
# modparam("nathelper", "ping_nated_only", 1)


# If using more RTP proxies they are addressed by index starting from index 0
modparam("nathelper", "rtpproxy_sock", "unix:/var/run/rtpproxy.sock udp:5.6.7.8:22222")

modparam("acc_syslog", "early_media", no)
modparam("acc_syslog", "failed_transactions", no)
modparam("acc_syslog", "report_ack", no)
modparam("acc_syslog", "report_cancels", no)
modparam("acc_syslog", "log_flag", "acc.log")
modparam("acc_syslog", "log_missed_flag", "acc.missed");

modparam("rr", "enable_full_lr", 1)    # testing purpose


route {
  ######### Generic processing independend of domains ########
   route("generic");

   lookup_domain("$fd","@from.uri.host");
    # lookup_domain("From");



   lookup_domain("$td","@ruri.host");
   # lookup_domain("Request-URI");

# First of all check for 3rd party relaying attempts. This must be at
# The beginning of the script, because rewrite actions below can change
# local destination domain to 3rd party domain. In such a case 3rdparty
# Inbound calls would match this condition (if it was at the end of
# the configuration file) and reject an otherwise valid call.
   if (!$f.did && !$t.did) {

           sl_reply("400", "No Relaying");
           drop;
   }

# Processing based purely on From domain, that includes authentication
# and identity verification, caller profile loading, NAT testing
   if ($f.did) {
           route("nat.detect");     # NAT detection comes here, this section sets flag FLAG_NAT_UAC

           route("authenticate");





           if (!lookup_user("From")) {
           # if (!lookup_user("$fu", "@from.uri.user")) {
                   sl_reply("400", "Fake Identity");
                   drop;
           }
           load_attrs("$fu", "$f.uid");
   }

######## Processing based on the called domain ########
   if ($t.did) {
                   if (method == REGISTER) {
                           route("registrar");
                             exit(1);
            }








                   # redirect
                   if ($f.did) {
                       ;
    }

    # Once we get here, the request can be either from one of our
    # domains or from a 3rd party domain. Thus we can put actions
    # independent of the initiator of the request here.

    # 1) Access control
    # 2) ENUM
    # 3) Test for PSTN gateways and other SIP servers that belong
    #    to the same domain

    #route("enum");
```

```
        # Test whether $t.did has changed

        # We have finished all the destination rewrites here, the request
        # is neither for another server in our domain nor for a 3rd party
        # domain, thus try to lookup end-user in the user location database.

                        if (!lookup_user("Request-URI")) {
                                sl_reply("404", "Not Found");
                                drop;
                        }

                        # User found, load his profile
                        load_attrs("$tu", "$t.uid");



                        if (!lookup_contacts("location")) {

                                sl_reply("483", "Temporarily Unavailable");

                                drop;
                        }
                        if (isflagset("FLAG_NAT_UAS")) {

                                $t.nat = true;
                        }
                        else $t.nat = false;
                }

                route("send");
}


route["generic"]
{


        # Handle XMLRPC requests first
        if (method == "POST" || method == "GET") {
                route("xmlrpc");
                drop;
        }

        if (!process_maxfwd("10")) {
                sl_reply("483", "Too Many Hops");
                drop;
        }

        if (msg:len >= max_len) {
                sl_reply("513", "Message Too Large");
                 drop;
        }

        loose_route();

     ######## Mid-dialog requests need special treatment ########
        if (has_totag()) {
                route("in_dialog");
                drop;
        }
}


#
```

```
# Handling of in-dialog requests
#
route["in_dialog"]
{

       # dump_attrs();
       route("send");
       drop;
}


#
# Handle REGISTER requests
#
route["registrar"]
{

       # Make sure this is domain our registrar is responsible for

       if (!lookup_domain("$td","@to.uri.host")) {
       # if (!lookup_domain("To")) {
               sl_reply("404", "Domain Not Local For Registrar");
       drop;
       }


    # Make sure that this is one of our users
       # if (!lookup_user("$tu", "@to.uri.user")) {
       if (!lookup_user("To")) {
               sl_reply("404", "Registrant Not Found");
                drop;
       }

       # Load registrant's profile
       load_attrs("$tu", "$t.uid");
       # load_attrs("t.uid");




    # Do whatever is needed here based on registrant's profile

       if (!save_contacts("location")) {
               sl_reply("400", "Invalid REGISTER Request");
               drop;
       }
}


#
# Digest authentication using database
#
route["auth_db"]
{

       if (!proxy_authenticate("@to.uri.host", "credentials")) {
               append_to_reply("%$digest_challenge");
               sl_reply("407", "Proxy Authentication Required");
               drop;
       }


}

```

```


#
# Authentication stanza
#
route["authenticate"]
{


    # Test for trusted peers here that will not be authenticated

       if (proto == TLS) {
               route("auth_tls");
       } else {
               route("auth_db");
               # route("auth_radius");
       }
}


#
# XML-RPC based management interface
#
route["xmlrpc"]
{


       create_via();

       # XML-RPC requests must be secured using TLS
       if (proto != TCP) {
                sl_reply("490", "Invalid Transport Protocol");
               drop;
       }

       dispatch_rpc();
       drop;
}


############### Forward blocks ###############


#
# Forward the request, do NAT mangling if necessary before
# forwarding
#
route["send"]
{

       route("nat.mangle");
       t_on_reply("nat.mangle");

       if ($record_route) {
               setavpflag("$f.nat", "dialog_cookie");
               setavpflag("$t.nat", "dialog_cookie");
               setavpflag("$rtpproxy", "dialog_cookie");
               setavpflag("$con_realm", "dialog_cookie");
               setavpflag("$rtp_proxy_node", "dialog_cookie");
               setavpflag("$f.comedia", "dialog_cookie");
               setavpflag("$t.comedia", "dialog_cookie");

               record_route();
       }
```

```

        if (!t_relay()) {
                sl_reply_error();
                drop;
        }
}


route["nat.detect"]
{

        if (src_ip == 127.0.0.1) return;

        if (is_present_hf("^Record-Route:")) return;

        if (nat_uac_test("3")) {
                force_rport();
                setflag("FLAG_NAT_UAC");
                $f.nat = true;
        }
    else $f.nat = false;
    return;
}

onreply_route["nat.mangle"]
{

        if ($t.nat || $tr.nat) {

                fix_nated_contact();
        }

        if (!$f.nat && (!$t.nat || !$tr.nat)) return;

        if ($con_realm || $tr.con_realm) return;

        if (@cseq.method != "INVITE") return;
        if ((status =~ "(183)|2[0-9][0-9]") && !search("^Content-Length: 0")) {

                if (!$rtpproxy && !$tr.rtpproxy) {

                        if ($t.nat || $tr.nat) {
                                fix_nated_sdp("8");
                                return;
                        }
                        else return;
                }

                if ($rtp_proxy_node)
                        eval_push("x:N%$rtp_proxy_node");

                else if ($tr.rtp_proxy_node)
                        eval_push("x:N%$tr.rtp_proxy_node");

                force_rtp_proxy("@eval.pop[-1]");

        }
return;
}

route["nat.mangle"]
{

        if ($f.nat) {
                if (method == "REGISTER") {
                        fix_nated_register();
                 }
                 else {
                        fix_nated_contact();
                 }
        }

        if (($tr.rtpproxy || $fr.rtpproxy) && (method == "BYE" || method == "CANCEL")) {

                if ($tr.rtp_proxy_node)
                        unforce_rtp_proxy("$tr.rtp_proxy_node");
                else
                        unforce_rtp_proxy("$rtp_proxy_node");

        }
        else if (method == "INVITE") {
                if (!$f.nat && (!$t.nat || !$tr.nat)) return;

                $record_route = true;

                if ($con_realm || $tr.con_realm) return;
                if ($f.connectivity_realm == $t.connectivity_realm) {
                        $con_realm = true;
                        return;
                }

                if (($f.sym_pass || $fr.comedia) && ($t.nat || $tr.nat) && !$f.nat) {
                        $f.comedia = $f.sym_pass;
                        return;
                }
                if (($t.sym_pass || $tr.comedia) && $f.nat && (!$t.nat || !$tr.nat)) {
                        $t.comedia = $t.sym_pass;
                        fix_nated_sdp("8");
                        return;
                }

                if ($tr.rtpproxy || $rtpproxy) {
                        if ($tr.rtp_proxy_node)
                                eval_push("x:LN%$tr.rtp_proxy_node");
                        else
                                eval_push("x:LN%$rtp_proxy_node");
                }
                else {
                        $rtpproxy = true;
                        $rtp_proxy_node = $node;

                        eval_push("x:N%$rtp_proxy_node");
                }
                force_rtp_proxy("@eval.pop[-1]");
        }
return 1;
}
```