OsmoHLR - Bug #3793

GSUP message routing for inter-MSC handover

02/08/2019 01:26 PM - osmith

Status:	Resolved	**Start date:**	02/08/2019
Priority:	High	**Due date:**	
Assignee:	osmith	**% Done:**	100%
Category:			
Target version:			

Description

N osmo-mscs connected to one osmo-hlr forward messages to each other via GSUP, using the source_name and destination_name IEs introduced in #3774.

Neels wrote in the openbsc ML:

```
- Connect two GSUP clients (test programs?) to a running osmo-hlr instance, and
  to extend the gsup_server so that we can use the source_name/destination_name
  IEs to forward GSUP messages between the two clients. (I added these because
  I'm fairly certain there is no existing in-band IPA protocol bits that would
  allow such routing; but I only briefly skimmed it, wouldn't hurt if you could
  verify that again.) The aim is to have plain gsup_client API to allow me to
  "just send messages back and fort".

- The session_id/session_state: we still need to figure out how to avoid
  collisions in session_id numbers, as any peer may at any point re-use the
  same session_id. Vadim suggested using a TI flag that gets flipped between
  sender and receiver, but since there are more than two peers, I guess we
  should treat each session_id as subordinate to a Request's source_name. IOW,
  any peer owns the entire session_id number space itself for sending out
  Request message types, and a Response or Error message type echos this
  session_id back with the source_name then having become the destination_name;
  it is perfectly legal for two peers to use the same session_id and collisions
  are avoided by Request vs. Response/Error. Something like...

    MSC-A   MSC-B   MSC-B'
      ------->              FOO_REQUEST(source=alice, destination=bob, id=3, state=BEGIN)
      <-------              FOO_RESPONSE(source=bob, destination=alice, id=3, state=CONTINUE)
      ------->              BAR_REQUEST(source=alice, destination=bob, id=3, state=CONTINUE)
      <-------              BAR_RESPONSE(source=alice, destination=bob, id=3, state=CONTINUE)
      --------------->      FOO_REQUEST(source=alice, destination=fred, id=4, state=BEGIN)
      <---------------      FOO_ERROR(source=fred, destination=alice, id=4, state=CONTINUE)
      --------------->      END_REQUEST(source=alice, destination=fred, id=4, state=END)
      ------->              END_REQUEST(source=alice, destination=bob, id=3, state=END)
```

Related issues:

Related to OsmoMSC - Feature #3774: implement GSUP messages for inter-MSC han...	**Resolved**	**01/31/2019**

History

#1 - 02/08/2019 01:26 PM - osmith

- Related to Feature #3774: implement GSUP messages for inter-MSC handover added

#2 - 02/08/2019 01:28 PM - osmith

- Subject changed from IPA routing to GSUP message routing for inter-MSC handover

#3 - 02/11/2019 01:40 PM - osmith

- Status changed from New to In Progress

#4 - 02/12/2019 05:20 PM - osmith

- % Done changed from 0 to 10

#5 - 02/13/2019 05:53 PM - osmith

- % Done changed from 10 to 20

Here are the concepts I came up with and started to implement.

neels: what do you think, would this be a good API to work with?
laforge: FYI, if you are interested in the details.

osmo-hlr

First I thought, the proper location to forward messages would somewhere in gsup-server.c. But I've realized that this only handles the encoded message as sent on the wire, not the decoded version. We need to look at the decoded version, to figure out if it needs to be forwarded to another client.

So I've extended read_cb() in osmo_hlr.c with:

```
+       if (gsup.destination_name_len)
+               return read_cb_forward(msg, &gsup);
```

and added a read_cb_forward() function, which will do the following:

- check if all session IEs are present
- verify source name, destination name length (before we use string compare functions on them)
- check if source name matches where it comes from
- when any check of the above fails, abort
- otherwise send the *original* message to the new destination (without re-encoding it, so osmo-hlr may not remove IEs it does not know about if osmo-hlr is compiled against an older libosmocore version compared to what the clients were built against)

... and that's all there is to it from osmo-hlr side. I would do all the session checking etc. in the client.

libosmo-gsup-client

I've created new gsup_client_session.{c,h} files in src/gsupclient. Picking a good name wasn't trivial for me, because there are already different kinds of sessions in hlr_ussd.c. But I went with client_session, and kept almost everything in the client (vs. the other sessions, which are handled in the server), so it should be reasonable to differentiate.

Receiving messages

Just like before, the client using libosmo-gsup-client uses osmo_gsup_client_create() with a callback function:

```
osmo_gsup_client_create(ctx, "msc-a", "127.0.0.1", OSMO_GSUP_PORT, read_cb_client, NULL);

int read_cb_client(struct osmo_gsup_client *client, struct msgb *msg)
{
    static struct osmo_gsup_message gsup_msg;
    osmo_gsup_decode(msgb_l2(msg), msgb_l2len(msg), &gsup_msg); /* error handling omitted */

    if (!osmo_gsup_client_session_sanity_check(&gsup_msg))
        goto error;

...
error:
    /* free up gsup_msg etc. */
}
```

The only new thing in the callback is the osmo_gsup_client_session_sanity_check() call. It does the following things:

- check if the GSUP message type needs to have session data attached or not (if not, skip everything else and return true)
- check if all session IEs are there
- verify source name, destination name length (before we use string compare functions on them)
- check if session state makes sense, depending on the message
- if the session state is "begin", try to create a new session (there's a global list of currently open sessions, register it there)
- check if the session is valid
- update the session's watchdog/guard timer
- when anything is wrong with this message, print a meaningful message to the log and return false

So when the sanity check went through, we are sure that the GSUP message is valid and can be used.

Sending messages

It should work roughly like this:

```
struct osmo_gsup_message gsup_msg = {0};
struct osmo_gsup_client_session *session = osmo_gsup_client_session_alloc(ctx, gsup_client, "msc-b");

gsup_msg.message_type = OSMO_GSUP_MSGT_E_PREPARE_HANDOVER_REQUEST;
strncpy(gsup_msg.imsi, "123456", sizeof(gsup_msg.imsi));
gsup_msg.an_apdu = /* ... */;
osmo_gsup_client_session_enc_send(session, &gsup_msg);
```

The first new function is osmo_gsup_client_session_alloc(), which adds a new entry to the client's global session list:

```
struct osmo_gsup_client_session {
    struct llist_head list;
    struct osmo_gsup_client *source;
    const char *destination_name;
    uint32_t session_id;
    enum osmo_gsup_session_state session_state;
};
```

The session_id is randomly generated, and tested against the existing list entries with the same source and destination to make sure that there are no duplicates (yes, performance could be optimized here by making one list per destination for example, but this isn't needed here, right?).

Then it gets sent out by the other new function osmo_gsup_client_session_enc_send(), which does some error checking, fills out all the session IEs and finally sends the message with osmo_gsup_client_enc_send:

```
int osmo_gsup_client_session_enc_send(const struct osmo_gsup_client_session *session,
                      struct osmo_gsup_message *gsup_msg)
{
    const char *source_name = session->source->ipa_dev->unit_name;

    if (gsup_msg->session_id || gsup_msg->session_state || gsup_msg->source_name || gsup_msg->source_name_len
        || gsup_msg->destination_name || gsup_msg->destination_name_len) {
        /* TODO: log error! */
        return -EINVAL;
    }

    gsup_msg->session_id = session->session_id;
    gsup_msg->session_state = session->session_state;
    gsup_msg->source_name = (uint8_t *)source_name;
    gsup_msg->source_name_len = strlen(source_name);
    gsup_msg->destination_name = (uint8_t *)session->destination_name;
    gsup_msg->destination_name_len = strlen(session->destination_name);

    return osmo_gsup_client_enc_send(session->source, gsup_msg);
}
```

Tests

My first approach was creating a C test, which starts the gsup server and two gsup clients, and tries to send messages between them. This would need read_cb() from hlr.c, so I figured it would be a good idea to build upon hlr.c, but replace main() with my own test main() function. This compiles and links properly now, but as I'm trying to run it, this does not seem like the best approach anymore. Because I need a lot of stuff from the original main() to get the server running properly, but the main loop would need to be replaced with testing logic, and in the end we aren't really running osmo-hlr's original gsup server anymore, but instead some modified version that may do quite a few things differently.

So I'm trying to run the osmo-hlr binary now, with a C test that creates two clients with the libosmo-gsup-client functions I've described above.

I created a prepare.sh and cleanup.sh shell script, which would run osmo-hlr in the background with the example config, save the pid, and kill it afterwards. In testsuite.at it looks like this:

```
AT_SETUP([gsup_client_session])
AT_KEYWORDS([gsup_client_session])
cat $abs_srcdir/gsup_client_session/gsup_client_session_test.ok > expout
cat $abs_srcdir/gsup_client_session/gsup_client_session_test.err > experr
$abs_srcdir/gsup_client_session/prepare.sh "$abs_top_builddir" "$abs_top_srcdir"
AT_CHECK([$abs_top_builddir/tests/gsup_client_session/gsup_client_session_test], [], [expout], [experr])
$abs_srcdir/gsup_client_session/cleanup.sh "$abs_top_builddir"
AT_CLEANUP
```

However, this doesn't work as expected: the testsuite will hang until osmo-hlr gets killed :\

My conclusion is, that we should rather do something like the external python tests do; create a python script that will run osmo-hlr, but instead of accessing the vty, run the test program and redirect its output. Then kill osmo-hlr when the test program quits, and pass the test program's exit code. Maybe something like that:

```
$ export DAEMON="osmo-hlr -c=..."
$ some-wrapper.py -- path/to/test_program
```

From what I can tell, we don't have anything similar in osmo-python-tests.git, so next week I would start with creating that.

#6 - 02/18/2019 05:56 PM - osmith

- % Done changed from 20 to 30

> My conclusion is, that we should rather do something like the external python tests do; create a python script that will run osmo-hlr, but instead of accessing the vty, run the test program and redirect its output. Then kill osmo-hlr when the test program quits, and pass the test program's exit code.

Created osmo_interact_exec.py in osmo-python-tests.git (see osmith/osmo-interact-exec branch, patch for gerrit coming when the rest of this issue is ready).

```
$ scripts/osmo_interact_exec.py -h
usage: osmo_interact_exec.py [-h] -r RUN_APP_STR -p PORT [-O OUTPUT_PATH] -c
                             CMD_STR

Wait until a given application listens for TCP connections, then run C tests or
other programs against it.

Example:
  osmo_interact_exec.py \
    -r 'osmo-hlr -c /etc/osmocom/osmo-hlr.cfg -l /tmp/hlr.db' \
    -p 4222 \
    -c 'tests/gsup_client_session/gsup_client_session_test'

optional arguments:
  -h, --help            show this help message and exit
  -r RUN_APP_STR, --run RUN_APP_STR
                        command to run to launch application to test,
                        including command line arguments. If omitted, no
                        application is launched.
  -p PORT, --port PORT  Port to reach the application at.
  -O OUTPUT_PATH, --output OUTPUT_PATH
                        Write command results to a file instead of stdout.('-O
                        -' writes to stdout and is the default)
  -c CMD_STR, --command CMD_STR
                        Run this command (before reading input files, if any).
                        multiple commands may be separated by ';'
```

I can call this from testsuite.at, and it successfully compares against a file with the expected output. The test gets skipped if ./configure was called without --enable-external-tests. There's the usual make target to regenerate the expected output.

Next up is writing a working test and further implementing the API as laid out above.

#7 - 02/19/2019 09:27 AM - laforge

osmith wrote:

> My first approach was creating a C test, which starts the gsup server and two gsup clients, and tries to send messages between them.

The question is **what** do you want to test. In general, except for some historical differences in a few cases, the C tests that are part of [lib]osmo* should only test individual functions. As soon as it comes to testing a given implementation remotely via some protocol, we use the TTCN-3 tests.

Do you want to test an API? Do you want to test a protocol implementation on how it interacts remotely?

In the "worst" case, I can imagine a test that combines both parts: Something like a small C program using the library to create a "test binary", which is then exercised via the protocol remotely from a TTCN-3 testsuite.

> So I'm trying to run the osmo-hlr binary now, with a C test that creates two clients with the libosmo-gsup-client functions I've described above.

I think this sounds like you want two sets of TTCN-3 tests:

- one where the TTCN-3 code behaves as HLR and you have some C client connecting to it
- one where the TTCN-3 code behaves as client and you have the standard osmo-hlr

The GSUP code we have can already behave in those two roles and is used that way: Emulate the HLR side when testing OsmoMSC, and emulate the client when testing OsmoHLR. Whatever new messages/IEs you are defining need to be added to GSUP_Types.ttcn anyway.

#8 - 02/19/2019 11:00 AM - osmith

laforge wrote:

> The question is **what** do you want to test. In general, except for some historical differences in a few cases, the C tests that are part of [lib]osmo* should only test individual functions. As soon as it comes to testing a given implementation remotely via some protocol, we use the TTCN-3 tests.
>
> Do you want to test an API? Do you want to test a protocol implementation on how it interacts remotely?

I would like to test the client-side API directly in a C program, inspired by what neels wrote:

> Connect two GSUP clients (test programs?) to a running osmo-hlr instance, and to extend the gsup_server so that we can use the source_name/destination_name IEs to forward GSUP messages between the two clients. [...] The aim is to have plain gsup_client API to allow me to "just send messages back and fort".

So... this is *not* a function test with everything stubbed out except for the pure function logic. It is on a slightly higher level, testing the client-side API, but still not at the protocol level because the interesting part is only the client side. I had tried to stub out the whole GSUP server, but that seemed to be more complex than just running the osmo-hlr binary, hence the osmo_interact_exec.py script.

Would it be fine to keep a test like this for the client-side functions, and then add or extend existing TTCN3 tests for protocol testing?
Or should I do what you proposed instead with the two sets of TTCN3 test, and possibly create additional C function tests?

(If you don't mind: in any case, I would like to continue with the testing stuff I have now locally, to make the API work and pass the tests, and then refactor the testing code as desired.)

> The GSUP code we have can already behave in those two roles and is used that way: Emulate the HLR side when testing OsmoMSC, and emulate the client when testing OsmoHLR. Whatever new messages/IEs you are defining need to be added to GSUP_Types.ttcn anyway.

Nice. I did not write any TTCN3 tests yet, but now that you say it, I would extend GSUP_Types.ttcn accordingly once we are sure that the messages are working for the handover. (Neels asked to keep the GSUP messages patch WIP until he is actually using them.)

#9 - 02/22/2019 12:34 PM - osmith

Neels gave me the feedback to make the routing work first, before getting into the session management stuff. I've simplified the patch accordingly and submitted it as WIP:
https://gerrit.osmocom.org/#/c/osmo-hlr/+/13006/

#10 - 04/02/2019 03:09 AM - neels

- Blocks Feature #3893: update wireshark dissector for new inter-MSC GSUP messages added

#11 - 04/02/2019 03:21 AM - neels

Noticed a bug in error reporting; marked in gerrit code review, and a fix is also on branch neels/gsup_router in osmo-hlr.
http://git.osmocom.org/osmo-hlr/commit/?h=neels/gsup_router&id=b34ac5621973da3d5cbd5a36e75767d3fb7788b5

#12 - 05/10/2019 09:46 AM - laforge

- Blocks deleted (Feature #3893: update wireshark dissector for new inter-MSC GSUP messages)

#13 - 05/20/2019 09:33 AM - osmith

- Status changed from In Progress to Resolved

- % Done changed from 30 to 100

The patch was merged.