Kamailio World 2019

RTP Media Server

New Kamailio module in 5.3

Julien Chavanton

Lead software engineer - voice routing @ flowroute.com

02/24/2019

Introducing RMS RTP Media Server module Presentation agenda

In this presentation I will try to cover the following :

- ➔ Explain why I choose to implement this module in Kamailio.
- ➔ Share some details on problems encountered, as well somes design and performance considerations.
- ➔ Explain what we can do with the module and how it works.
- ➔ Since it is an non-default new module with dependencies, how can we install it and test it ?
- ➔ Clarify the current state of the module.

But Kamailio is a SIP proxy ?

philosophical objections !@#

Kamailio is not just a SIP proxy, I see it more as a SIP server with a rich ecosystem providing :

★ a very flexible and compliant SIP stack (RFC 3261 and others)
★ a scripting engine also supporting well known languages like Lua, Python, etc.
★ rich set of features implemented in various modules

The result of more than 15 years of development and testing.

---

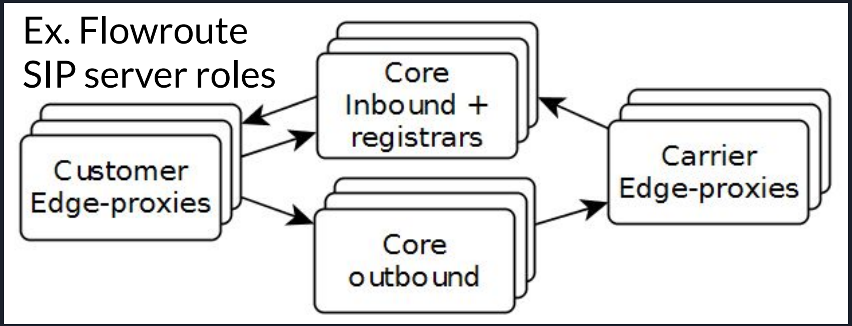
oRTP and mediastreamer2 are libraries, they are normally used with Belle-SIP, SofiaSIP, and in the past oSIP/eXosip.

★ **oRTP** is providing RTP endpoints compliant with (RFC 3550)
★ **MediaStreamer2** is implementing a framework for audio processing using graphs of filters, filters can be used to do various things. (even if it is written in C, it is quite a high level library)

These libraries were started in around 2000 they represent almost 20 years of development and testing.
They are maintained by Belledonne communications.
(Please note that Belledonne communications is not involved in the development of this module.)

Combining multiple open source projects in one application

People are often tempted to run only one or two Kamailio instances to do everything, simply because they can.
When using the RTP Media Server modules, performance will become more problematic, users will most likely have more than one Kamailio server.
Users can still run other Kamailio servers to do more specific roles like edge-proxies, registrars, push gateways etc.



When using the RMS module, Kamailio becomes an **application server** and we will inherit most the **existing features** provided by its core and most of its modules, without any need to rewrite, document and test all them.

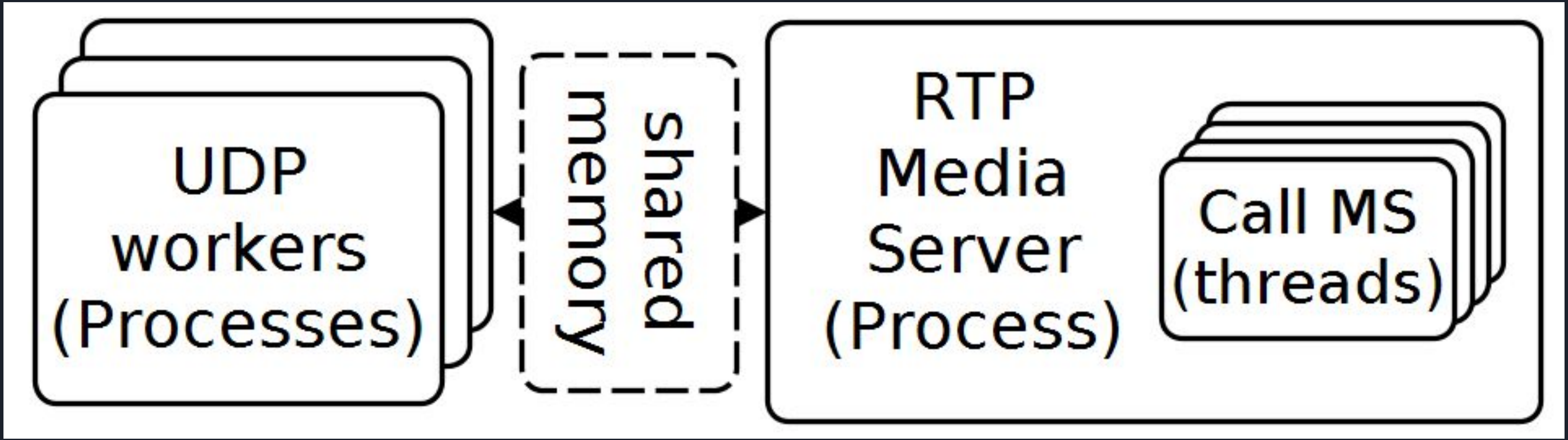
Kamailio is also handling everything related to **SIP** and is helping with SDP, the module is adding some SDP parsing as well as providing a **scripting engine**.

Even if there is already many existing solutions available, this is another way to use existing great projects for various use cases like IoT, IVR or other specific needs.

Kamailio is a multi-process application, this may be problematic with multi-threaded libraries.

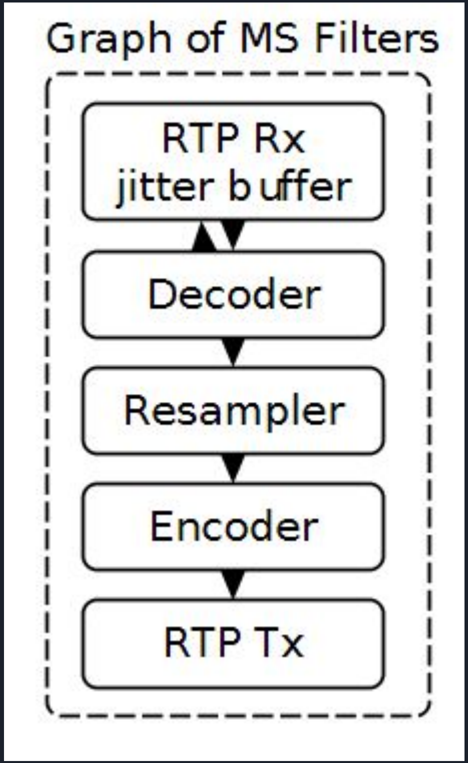
Mediastream2 is creating one thread per call "msticker".

When mixing mutliprocess and threads, we need to make sure all the memory allocated from distinct processes is shared and synchronized, this can sometime be another source of problem as we are making assumptions.
Shared memory allocation could supported in the libraries using wrapper around malloc/free, but I decided to centralize the interactions with with the threads in one process to make sure this will never be a problem.



Benefits of using a graph of filters

Performance report per filter



```
 AUDIO SESSION'S RTP STATISTICS
===========================================================
sent                                        313 packets
                                            0 duplicated packets
                                            53836 bytes


received                               299 packets
                                            0 duplicated packets
                                            51428 bytes


incoming delivered to the app               50568 bytes
incoming cumulative lost                    0 packets
incoming received too late                  0 packets
incoming bad formatted                      2 packets
incoming discarded (queue overflow)         0 packets
sent rtcp                                   0 packets
received rtcp                               2 packets
===========================================================
                FILTER USAGE STATISTICS
Name                  Count      Time/tick (ms)         CPU Usage
-----------------------------------------------------------
MSRtpSend             629        0.0188944              45.863
MSRtpRecv             629        0.0150079              36.4291
MSFilePlayer          629        0.0036385              8.83185
MSUlawEnc             627        0.00309774             7.49538
MSResample            627        0.000382067            0.92446
MSVoidSink            294        0.000401353            0.456181
MSUlawDec             0          0                      0
===========================================================
```

B2BUA : initial and in-dialog messages

When using the RMS module, Kamailio will connect and disconnect calls.
A call can be composed of 2 bridged “call legs”.

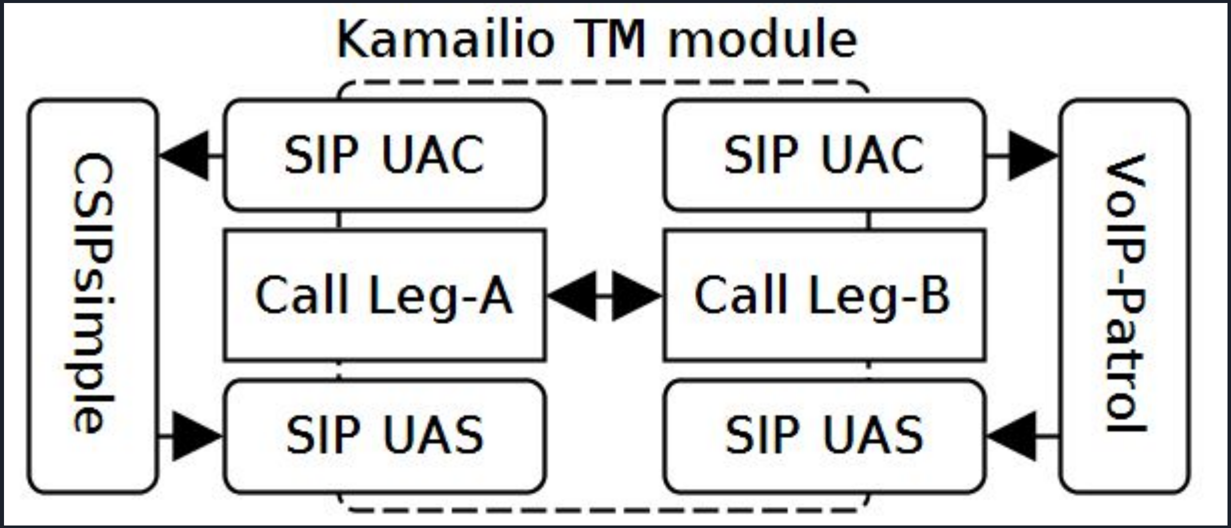
It is not possible to do all of this with a SIP proxy (I was not able to imagine a solution), fortunately Kamailio is already equipped to behave as a UAS and as a UAC, using flexible helper functions provided by the TM module API.

The RMS module will save the required Dialog state.
Note: the Dialog module can do many things but was not designed to accommodate this use case, it should be usable in conjunction with the RMS module.

```
// TM  API
load_tm_api(&tmb)

// User-Agent Client
tmb.new_dlg_uac()
tmb.t_request_within()

// User-Agent Server
UAS  tmb.t_reply_with_body()
```



Performance considerations

No load tests were done so far, only several active calls.

Concerning the libraries oRTP and MediaStreamer2, both of them are already being used in other server side components and are expected to be perform well.

oRTP : HP OCMP server with 2000 calls on x86 servers in 2006-2009
oRTP and MediaStreamer2 : both used in FlexiSIP

About the RMS module, there may be a concern in making sure the RMS process, will handle all the events generated by all calls and script interaction. But this should not be difficult to address, given the fact that other bottleneck will trigger before related to encoding, resampling and packet per second.

Fictive scenario highlighting the relation between CPS and calls

CPS call per second	ASR answered ratio	ACD average call duration	Active calls
50	**50%**	**60 seconds**	**1500**

Answer and play files 1/2

Routing script example

```
loadmodule "tm"
loadmodule "tmx"
loadmodule "rtp_media_server"
modparam("rtp_media_server", "log_file_name", "/var/log/rms_transfer.log");

event_route[rms:after_play] {
   rms_hangup();
}

event_route[rms:start] {
   rms_play("/opt/voice_files/OSR_us_000_0010_8k.wav", "rms:after_play");
}

route {
   if (is_method("INVITE") && !has_totag()) {
      rms_answer("rms:start");
   }

   if(rms_dialog_check()) // return true if this message is in a dialog handled by rms
      rms_sip_request();  // Handle in-dialog message
}
```

Answer and play files 2/2

Media Streamer2 graph example

Bellow we detail the graphs for a call when playing a file :

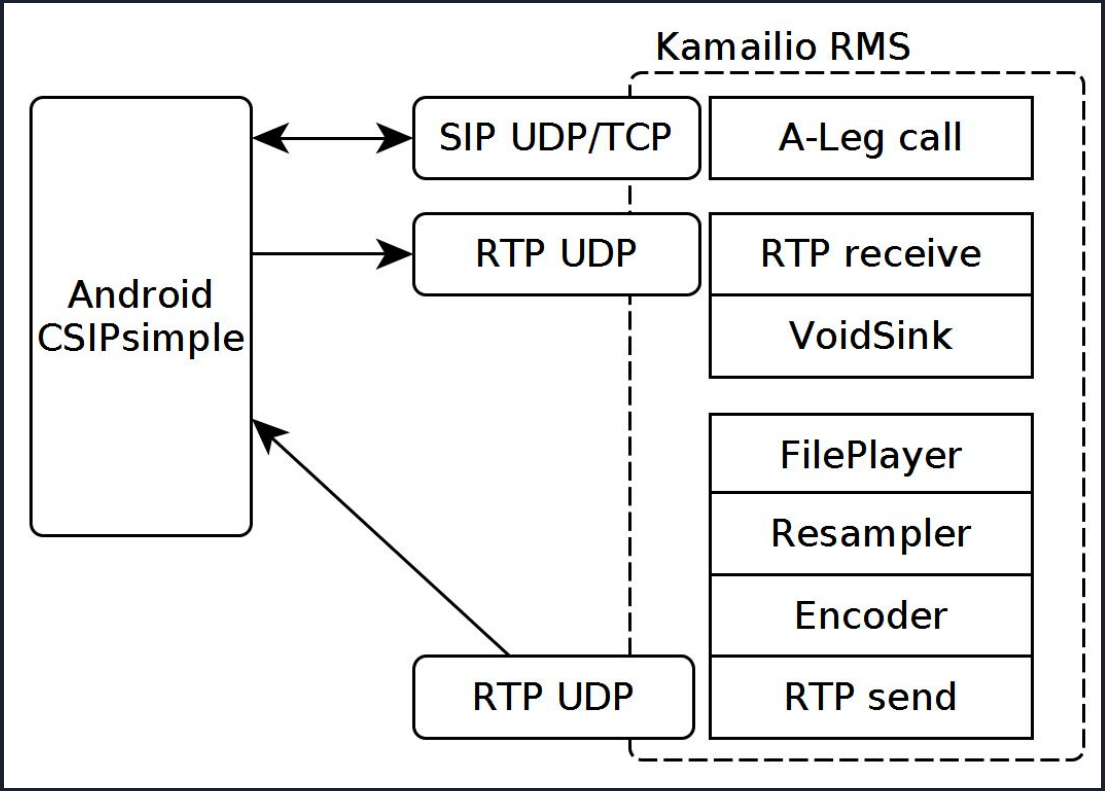
- We have one receiving and one sending graph, each having a chain of filters.
- For each call we have one thread, in each thread we run a ticker that will execute graphs every 10ms.

```
int rms_start_media(call_leg_media_t *m,
char *file_name) {

 MSConnectionHelper h;
 ...

 // receiving graph
 ms_start(&h);
 ms_link(&h, m->ms_rtprecv, -1, 0);
 ms_link(&h, m->ms_voidsink, 0, -1);

 // sending graph
 ms_start(&h);
 ms_link(&h, m->ms_player, -1, 0);
 ms_link(&h, m->ms_resampler, 0, 0);
 ms_link(&h, m->ms_encoder, 0, 0);
 ms_link(&h, m->ms_rtpsend, 0, -1);
```



Bridging calls 1/2

Routing script example

```
loadmodule "tm"
loadmodule "tmx"
loadmodule "rtp_media_server"
modparam("rtp_media_server", "log_file_name", "/var/log/rms_transfer.log");

event_route[rms:bridged] {
    xnotice("[rms:bridged] ...\n");
    // not much can be done from here, maybe we need to implement new commands:
    // rms_sleep();
    // rms_dtmf("09123456789+*", "rms:dtmf");
}

route {
   if (is_method("INVITE") && !has_totag()) {
      var(target) = "sip:" + $rU + "@domain.void:5060;";
      rms_bridge("$var(target)", "rms:bridged");// 2nd argument is the call back route
   }

   if(rms_dialog_check()) // return true if this message is in a dialog handled by rms
      rms_sip_request();  // Handle in-dialog message
}
```

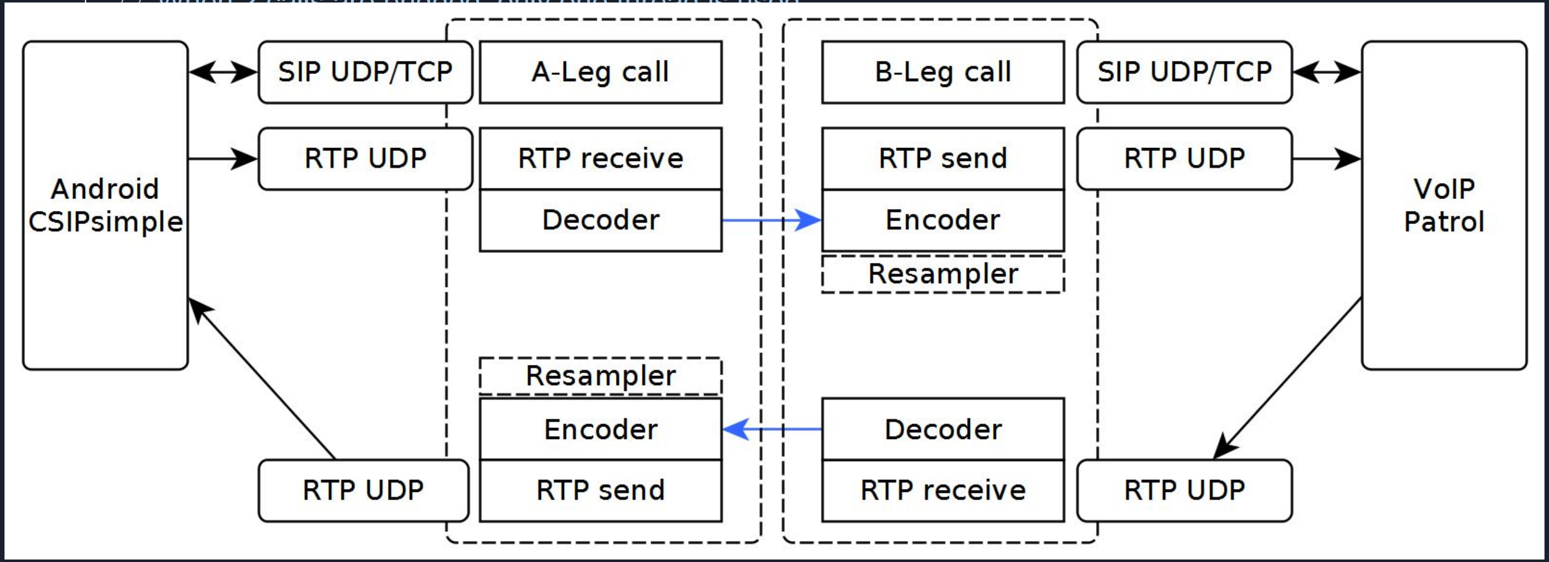
Bridging calls 2/2 MediaStreamer2 graph example

```
int rms_bridge(call_leg_media_t *m1, call_leg_media_t *m2) {
        MSConnectionHelper h;
        m1->ms_ticker = rms_create_ticker(NULL);

        // A-leg : inbound call graph
        ms_connection_helper_start(&h);
        ms_connection_helper_link(&h, m1->ms_rtprecv, -1, 0);
        ms_connection_helper_link(&h, m2->ms_rtpsend, 0, -1);

        // B-leg : outbound call graph
        ms_connection_helper_start(&h);
        ms_connection_helper_link(&h, m2->ms_rtprecv, -1, 0);
        ms_connection_helper_link(&h, m1->ms_rtpsend, 0, -1);

        ms_ticker_attach_multiple(
             m1->ms_ticker, m1->ms_rtprecv, m2->ms_rtprecv,
NULL);
        return 1;
} // When 2 calls are bridged, only one thread is used
```



Bridging after playing

Routing script example

```
event_route[rms:bridge] {
   $var(target) = "sip:123@voip.void:5060;";
      rms_bridge("$var(target)", "rms:bridged"));
   }
}

event_route[rms:bridged] {
   xnotice("[rms:bridged] ...\n");
}

event_route[rms:play_bridge] {
   rms_play("/opt/voice_files/Bach_10s_8000.wav", "rms:bridge");
}

route {
   if (is_method("INVITE") && !has_totag()) {
      rms_answer("rms:play_bridge");
   }

   if(rms_dialog_check()) // return true if this message is in a dialog handled by rms
      rms_sip_request();  // Handle in-dialog message
}
```

SDP parsing OFFER / ANSWER
Codecs and other informations

Kamailio libraries are providing some SDP parsing

../../core/parser/sdp/sdp.h

But only limited functionalities

More was needed to be to handle SDP Offer/Answer.

Some basic functionalities was implemented in rms_sdp.c

Currently only PCMU / PCMA are supported, support other codecs, like Opus, is currently disabled, more work is required in SDP parsing and oRTP/MS2 Payload .

The good news is that most free codecs and many non-free codecs are available in MediaStreamer2 and I know for a fact that they are well integrated.
Including a g.729 codec completely written by belledonne communication.
http://www.linphone.org/technical-corner/bcg729

Testing using the Docker provided files

```
git clone https://github.com/kamailio/kamailio.git
cd kamailio/src/modules/rtp_media_server/
```

Build the container image

```
cd docker
# see the file Dockerfile for the all commands used to build on Debian
docker build . -t rtp_media_server
```

Configure

```
# set your IP address in “example_config/kamailio.cfg “ listen line
docker cp config_example/kamailio.cfg rtp_media_server:/etc
```

Run

```
#start the container
./rtp_media_server.sh
# connect to the shell in the container
docker exec -it  rtp_media_server bash
# start Kamailio
kamailio -m 64 -D -dd -f /etc/kamailio.cfg
```

Current state of this module

Pre Alpha (New) : still some missing features

- Implement DTMF detection in most states and commands
- add recording command
- proper early media handling
- Enough SDP handling to support for free/non-free codecs

Unstable : Many edge cases are probably not covered.

Moving to Alpha : A proper testing strategy must be included, this must be addressed by including tests that can be automated.
I will most likely be using docker and a compliant test software like PJSIP/voip_patrol with a well defined test framework like Python Behave.

Thank you for listening !

Feel free to send me questions or comments
jchavanton@gmail.com
I

Thanks to Flowroute for sponsoring my trip to Kamailio World !

If you have any questions or feedback about Flowroute, let me know, I am curious to know more on why you would use Flowroute or not.

One reason to use Flowroute SIP trunking :

Flowroute has developed strategic partnerships with carriers to deliver optimized connections to provide direct media delivery. This reduces the number of “hops” and points of failure to deliver improved inbound and outbound voice quality. No resellers, no aggregators, no unnecessary chances for issues to arise.