Fosdem 2018

Building a WebRTC Gateway

Experiment with WebRTC native API

https://github.com/jchavanton/rtc_gw

Julien Chavanton

Lead software engineer - voice routing @ flowroute.com

01/24/2018

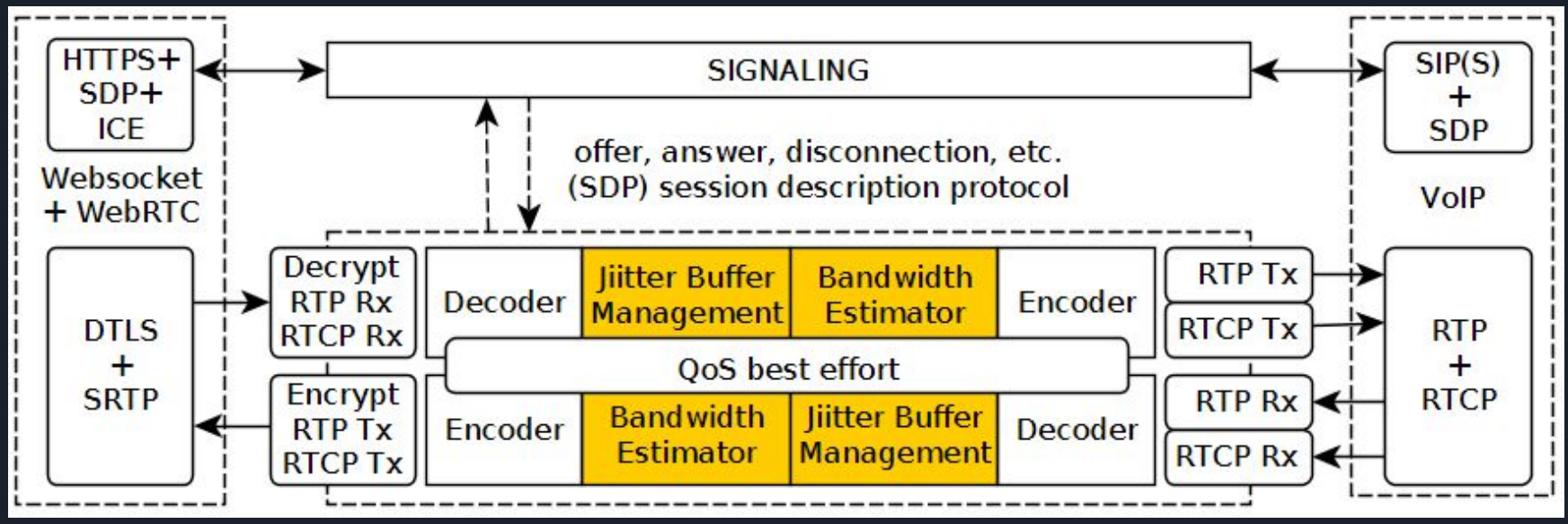
WebRTC Gateway

bridging WebRTC with VoIP

- ➔ WebRTC endpoints requires HTTPS, DTLS/SRTP, multiplexing of RTP/RTCP and support for ICE.
- ➔ VoIP does not require any of this, SIP is prefered for signaling and encryption is mostly using TLS on signaling socket and SRTP

WebRTC is focussing on connecting browsers running on laptops and mobile phones, achieving QoS best effort can be more challenging because users are everywhere connected over wireless networks.

With VoIP we do not always need to care about this, we sometimes assume that the users or switches are well connected to the Internet.



Using WebRTC to build a Gateway? 1/3

Native API (C++11) is evolving to accommodate out of browser usage

WebRTC update October 2017 - Path to 1.0 Update 

- "A shift of what was originally a web based API platform for real time communication into developers adopting it on mobile and building natively for it"
- “Injectable audio codecs and embedded device for the native environment”
- “Our mission: To enable rich, high-quality RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allow them all to communicate via a common set of protocols.”

WebRTC provides complex signal processing solutions, example :

AEC3 (Acoustic Echo Canceler) several improvements and testing being done on millions of calls. Other media libraries like **PJMedia** and **MediaStreamer2** are using part of code.

Example of integration of AEC-DA to mediastreamer2 (not official and outdated)
https://github.com/jchavanton/ms-aec-webrtc

However the fork makes it more expensive to test, maintain and update ...

Using WebRTC to build a Gateway? 2/3

NetEQ . The adaptive jitter buffer in WebRTC that can do fast accelerate, expand, clock skew compensation, etc.

- Required to provide QoS best effort when facing Jitter
- Such Jitter Buffer Management (JBM) can be complex to implement and to test

Example using the native API you can configure it

```
RTC Configuration Example
webrtc::PeerConnectionInterface::RTCConfiguration config;
config.audio_jitter_buffer_max_packets = 100;
config.audio_jitter_buffer_fast_accelerate = true;
Result in:
(webrtcvoiceengine.cc:511): NetEq capacity is 100
(webrtcvoiceengine.cc:517): NetEq fast mode? 1
```

Microsoft Edge compatibility, support for edge ORTC

```
commit 7aee3d538c548a66843efcf415c9ac50274846eb
       Fix ortc_api circular deps.
       This will help keep ortc dependencies clean in the
future ...
```

Using WebRTC to build a Gateway? 3/3

BWE (BandWidth Estimator) for audio using adaptive codecs functionality

chrome was planning to use bandwidth estimation also to decide the audio bitrate to send (Planned for version 58).

compatible and symmetric behavior

(same code == same feature set on each side)

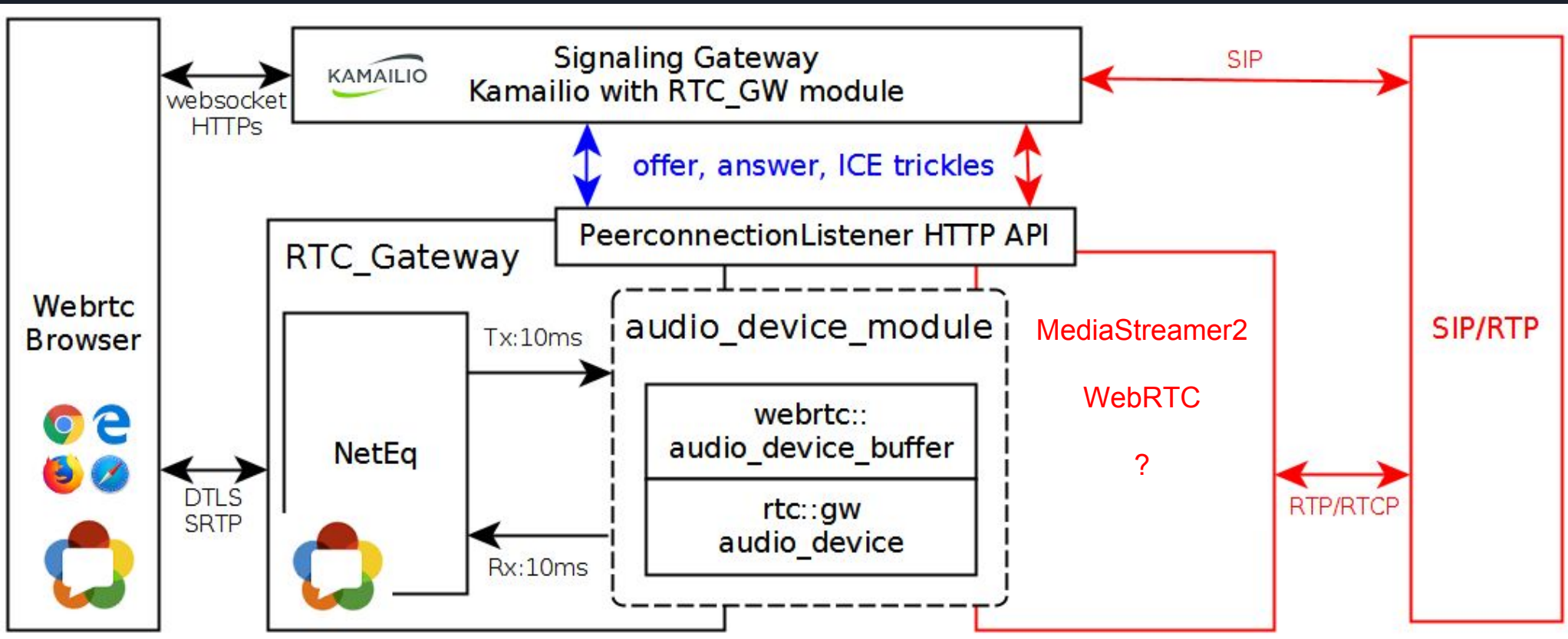
However I did not test it and it is still considered experimental

src/media/engine/webrtcvoiceengine.cc

```
void UpdateAllowedBitrateRange() {

// Note: This is an early experiment currently only supported by Opus.
    if (send_side_bwe_with_overhead_) {
    const int max_packet_size_ms =
        WEBRTC_OPUS_SUPPORT_120MS_PTIME ? 120 : 60;
```

Current State of Work In Progress



RTC_GW Source code

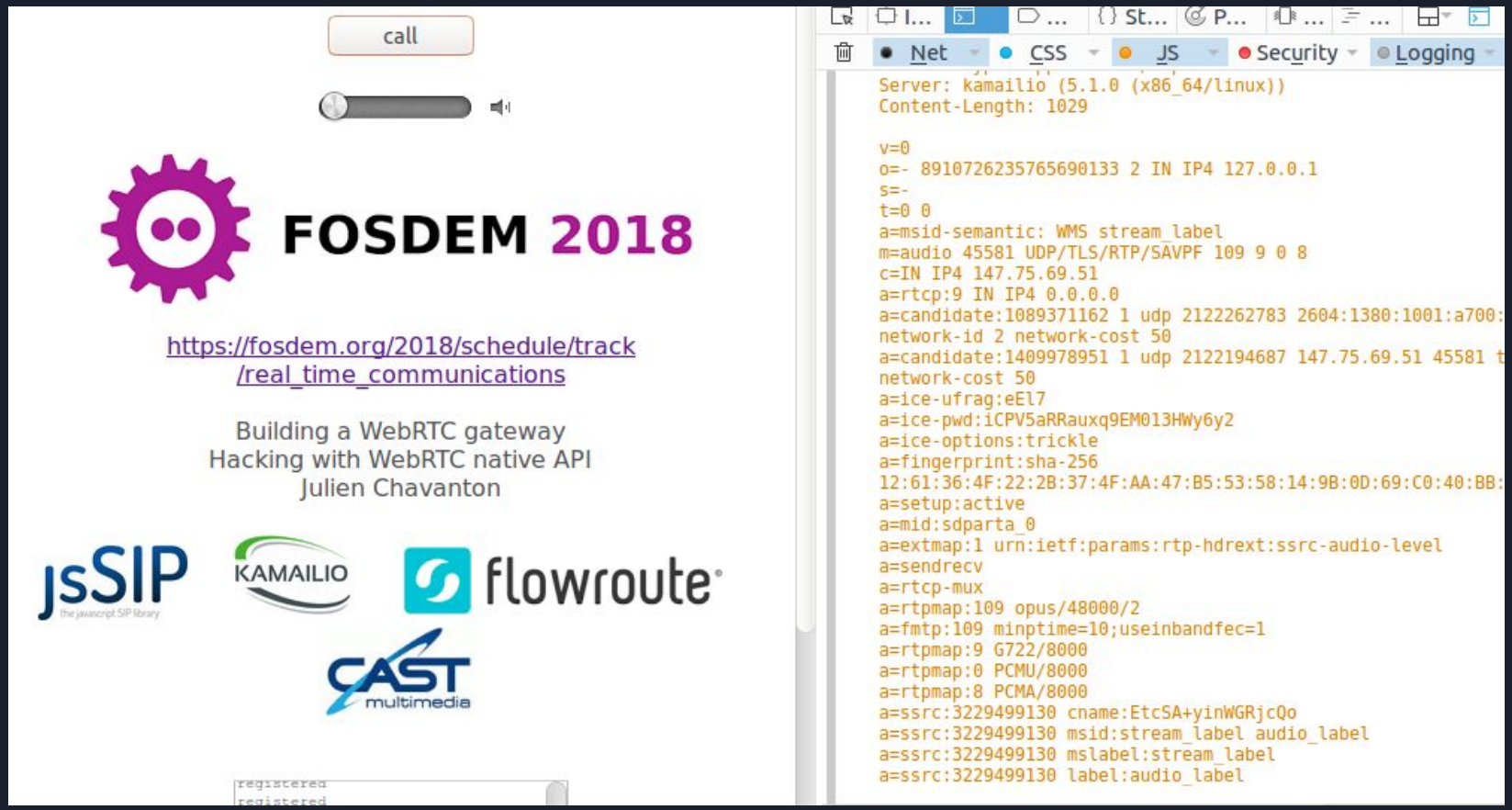
https://github.com/jchavanton/rtc_gw

The source code is mainly an experiment to verify what can be done with the current API or what could be missing (it does contain several quick and dirty hacks).

The code can be build from a separate set of files and will work without any modification to WebRTC (no need to fork), we can use the latest version and it should work as the API remain stable.

peer_connection_listener.cc peer_connection_listener.h	Forked from peer_connection_client application but now act as a server to request creation/destruction of peer connections
conductor.cc conductor.h	Forked from the peer_connection_client sample application with several modifications
audio_device_module.cc audio_device_module.h	Fork of file_audio_device.cc refactored to become an audio_device_module and include the audio_device_buffer
main.cc	

Live Demo


call
FOSDEM 2018
https://fosdem.org/2018/schedule/track/real_time_communications
Building a WebRTC gateway
Hacking with WebRTC native API
Julien Chavanton
jsSIP
KAMAILIO
flowroute
CAST multimedia
Net
CSS
JS
Security
Logging
Server: kamailio (5.1.0 (x86_64/linux))
Content-Length: 1029
v=0
o=- 8910726235765690133 2 IN IP4 127.0.0.1
s=-
t=0 0
a=msid-semantic: WMS stream_label
m=audio 45581 UDP/TLS/RTP/SAVPF 109 9 0 8
c=IN IP4 147.75.69.51
a=rtcp:9 IN IP4 0.0.0.0
a=candidate:1089371162 1 udp 2122262783 2604:1380:1001:a700:
network-id 2 network-cost 50
a=candidate:1409978951 1 udp 2122194687 147.75.69.51 45581 t
network-cost 50
a=ice-ufrag:eEl7
a=ice-pwd:iCPV5aRRauxq9EM013HWy6y2
a=ice-options:trickle
a=fingerprint:sha-256
12:61:36:4F:22:2B:37:4F:AA:47:B5:53:58:14:9B:0D:69:C0:40:BB:
a=setup:active
a=mid:sdparta_0
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=sendrecv
a=rtcp-mux
a=rtpmap:109 opus/48000/2
a=fmtp:109 minptime=10;useinbandfec=1
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=ssrc:3229499130 cname:EtcSA+yinWGRjcQo
a=ssrc:3229499130 msid:stream_label audio_label
a=ssrc:3229499130 mslabel:stream_label
a=ssrc:3229499130 label:audio_label

Signaling ... JSEP, ICE Trickle and SIP

When using the Native API we still need to handle the SDP exchange and modifications as needed by using Javascript Session Establishment Protocol directly from C++

Interactive Connectivity Establishment (ICE) is automatically triggered and on a server we probably always know in advance which network interface we want to use, we can simply control it and remove the candidate we do not want.

We can also choose to benefit from :
Trickle ICE: Incremental Provisioning of Candidates However this requires more signaling after the Offer/Answer to exchange more candidate as they are discovered. We sometime have to wait after STUN requests

Unfortunately this is not supported in SIP (not defined in any RFC) so I add to spend time to disable it, simply waiting after the candidate of the interface configured is found

```
Conductor::OnIceCandidate
```

Create an Audio Device Module

https://github.com/jchavanton/rtc_gw/blob/master/audio_device_module.cc
https://github.com/jchavanton/rtc_gw/blob/master/audio_device_module.h

Inherit from the abstract base class **AudioDeviceModule** and implement all the methods required.

```
class FileAudioDevice : public webrtc::AudioDeviceModule
```

In this AudoiDeviceModule, the constructor will automatically create an

AudioDeviceBuffer and attach it

```
Audio_device_buffer_ = new webrtc::AudioDeviceBuffer();
AttachAudioBuffer(Audio_device_buffer_);
```

Use an Audio Device Module

https://github.com/jchavanton/rtc_gw/blob/master/conductor.cc

Use the version **CreatePeerConnectionFactory** that let you specify which ADM "Audio device module" you want to use.

```
network_thread_ = new rtc::Thread();
rtcgw::FileAudioDevice *audio_device_ = new
rtcgw::FileAudioDevice("/audio/input_48K_16bits_pcm.raw",
                       "/audio/recording.raw");
network_thread_->Start();

peer_connection_factory_ = webrtc::CreatePeerConnectionFactory(
        network_thread_,          // specify : network, worker and signaling thread
        rtc::Thread::Current(),   // Maybe something to fix here, using the same thread
        rtc::Thread::Current(),   // because FileAudioDevice must run from the thread
        Audio_device_,            //  from which it was instantiated
        webrtc::CreateBuiltinAudioEncoderFactory(), // seems like we could provide our own
        webrtc::CreateBuiltinAudioDecoderFactory(), // factory to have more control over the
        Nullptr,      // Video                                    // Codec, not tested.
        Nullptr       // Video
);
```

Signaling Server using a Kamailio module

https://github.com/jchavanton/kamailio_mod_rtcgw

Example of Kamailio config usage :

```
loadmodule "rtc_gw.so"
modparam("rtc_gw", "server_address", "127.0.1.102");

if(is_method("INVITE") && !has_totag()) {
        rtc_sdp_offer();
        exit;
}
if(is_method("BYE")){
        rtc_bye();
        sl_send_reply("200", "OK");
}
```

Measurements Metrics and statistics

grep kStats ./api/statstypes.cc | wc -l

153 Metrics in this file

{ StatsReport::kStatsValueNameCurrentDelayMs, info.delay_estimate_ms },
{ StatsReport::kStatsValueNameDecodingCNG, info.decoding_cng },
{ StatsReport::kStatsValueNameDecodingCTN, info.decoding_calls_to_neteq },
{ StatsReport::kStatsValueNameDecodingCTSG,info.decoding_calls_to_silence_generator },
{ StatsReport::kStatsValueNameDecodingMutedOutput,info.decoding_muted_output },
{ StatsReport::kStatsValueNameDecodingNormal, info.decoding_normal },
{ StatsReport::kStatsValueNameDecodingPLC, info.decoding_plc },
{ StatsReport::kStatsValueNameDecodingPLCCNG, info.decoding_plc_cng },
{ StatsReport::kStatsValueNameJitterBufferMs, info.jitter_buffer_ms },
{ StatsReport::kStatsValueNameJitterReceived, info.jitter_ms },
{ StatsReport::kStatsValueNamePacketsLost, info.packets_lost },
{ StatsReport::kStatsValueNamePacketsReceived, info.packets_rcvd },
{ StatsReport::kStatsValueNamePreferredJitterBufferMs,info.jitter_buffer_preferred_ms },

What's Next ?

Complete call bridging

Possible option 1 : disable RTCP MUX, DTLS, and control ICE to create another WebRTC call leg compatible with VoIP.

Possible option 2 : Create a **MediaStreamer2** module and use it in conjunction with **oRTP** to create a standard simple media call leg to interface the “legacy” VoIP.

See an example of the Kamailio module doing bridging/transcoding using **MediaStreamer2/oRTP**:

Could be simple and efficient, I did an experimented last year

Kamailio media processing module

Dig more on QoS functionality & testing

Test Various scenarios of Jitter + packet loss and verify

Experiment with BWE for audio and adaptive codec behavior

Using PESQ to test

POLQA licence ...

webrtc/tree/master/modules/audio_processing/test/py_quality_assessment

Bridge call using MediaStreamer2/oRTP ?

http://www.belledonne-communications.com/mediastremer2.html

Example showing how MediaStreamer2 can be convenient and high level

```
int create_call_leg_media(call_leg_media_t *m, str *callid){
    m->ms_factory = rms_create_factory();
    m->callid = callid;
    // create caller RTP session
    m->rtps = ms_create_duplex_rtp_session(m->local_ip, m->local_port, m->local_port+1, ms_factory_get_mtu(m->ms_factory));
    rtp_session_set_remote_addr_full(m->rtps, m->remote_ip, m->remote_port, m->remote_ip, m->remote_port+1);
    rtp_session_set_payload_type(m->rtps, m->pt->type);
    rtp_session_enable_rtcp(m->rtps,FALSE);
    // create caller filters : rtprecv1/rtpsend1/encoder1/decoder1
    m->ms_rtprecv = ms_factory_create_filter(m->ms_factory, MS_RTP_RECV_ID);
    m->ms_rtpsend = ms_factory_create_filter(m->ms_factory, MS_RTP_SEND_ID);
    m->ms_encoder = ms_factory_create_encoder(m->ms_factory, m->pt->mime_type);
    m->ms_decoder = ms_factory_create_decoder(m->ms_factory, m->pt->mime_type);
    /* set filter params */
    ms_filter_call_method(m->ms_rtpsend, MS_RTP_SEND_SET_SESSION, m->rtps);
    ms_filter_call_method(m->ms_rtprecv, MS_RTP_RECV_SET_SESSION, m->rtps);
    return 1;
}

int rms_bridge(call_leg_media_t *m1, call_leg_media_t *m2) {
    MSConnectionHelper h;
    m1->ms_ticker = rms_create_ticker(NULL);
    // direction 1
    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);
    // direction 2
    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;
}
```

Thank you for listening !

Looking forward working with you on Free Software

Searching for a working WebRTC Gateway ? you may want to look at :

Freeswitch, Janus, Kamailio/RTP Engine, Asterisk, SEMS, ...

Thanks to Flowroute for sponsoring my trip to Fosdem and Other Free software events !

Speaking about them and audio quality, why would I recommend using Flowroute as a SIP trunk carrier in the US ?

Because it is the only carrier (as far as I know) that will give you direct carrier media connection, therefore you get the low latency, high quality path for you media !

Thanks to :
Claude Lamblin @ Orange Lannion (Feedback on NetEQ)
William King and Maria Bermudez @ Flowroute (Help and support)