webrtc2sip - Smart SIP and Media Gateway for WebRTC endpoints

Technical Guide

by

Mamadou DIOP
diopmamadou {AT} doubango[DOT]org

License

Table of Contents

Table of Figures

Table of Samples

1 Foreword

RTCWeb (a.k.a *WebRTC*) stands for **R**eal-**T**ime **C**ommunication and is a new technology being drafted by the **W**orld **W**ide **W**eb **C**onsortium (W3C) and **IETF** groups. This technology has the ambition to bring native real-time features (audio, video and arbitrary data) to the web browsers without requiring additional plugins.

SIP stands for **S**ession **I**nitiation **P**rotocol and is a signaling protocol defined by the IEFT in *RFC 3261*. *SIP* is widely used today to manage VoIP (**V**oice **o**ver **IP**) communication sessions and has been chosen as signaling protocol for **N**ext **G**enerations **N**etworks such as *IMS* (**IP** **M**ultimedia **S**ubsystem) or *LTE* (**L**ong **T**erm **E**volution). The protocol has quickly become the de facto standard used to interconnect the IP world (Internet) with the PSTN (circuit-switched telephone networks).

webrtc2sip is a smart and powerful gateway using *RTCWeb* and *SIP* to turn your browser into a phone with audio, video and SMS capabilities. The gateway allows your web browser to make and receive calls from/to any SIP-legacy network or PSTN. As an example, you will be able to make a call from your preferred web browser to a mobile or fixed phone.

2 Scope

This technical guide is a reference document explaining why you need *webrtc2sip* and how to leverage its power.

3 Architecture

The gateway contains three modules: *SIP Proxy*, *RTCWeb Breaker* and *Media Coder*.

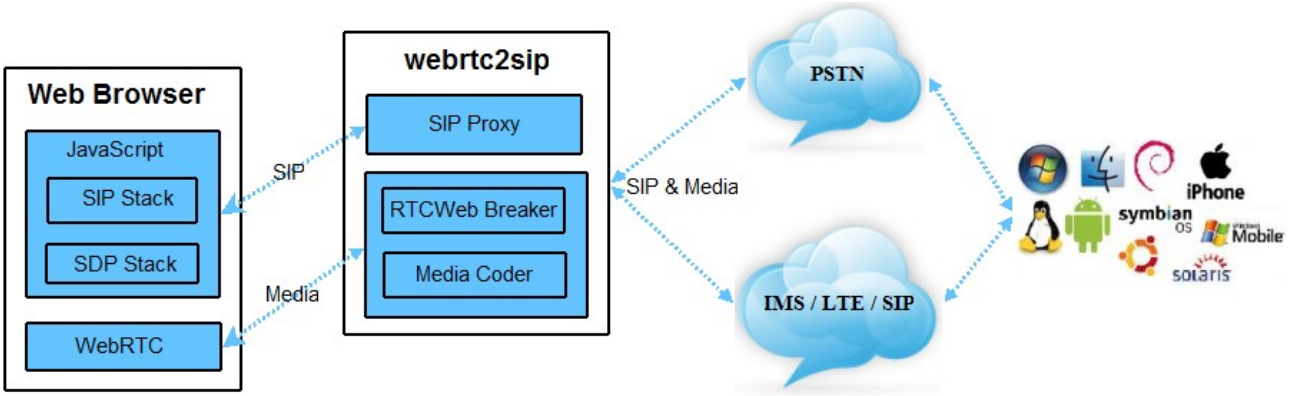


Figure 1: Architecture

The HTML SIP client is any endpoint implementing draft-ibc-sipcore-sip-websocket-06. We highly recommend using sipML5 which is known to work and provide good performances.

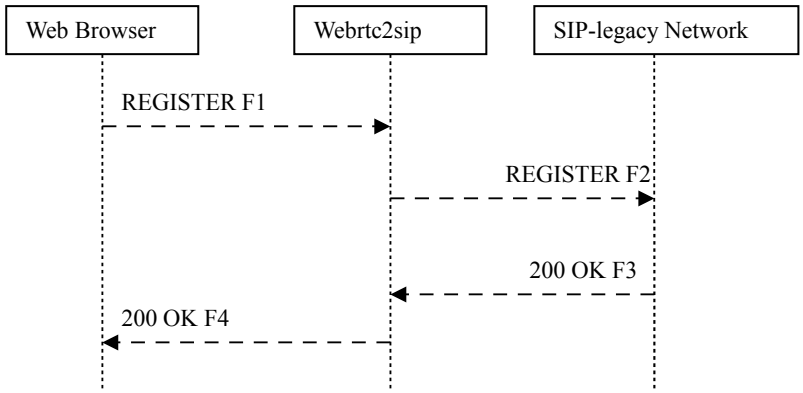
3.1 SIP Proxy module

Figure 2: SIP Proxy architecture

The role of the *SIP Proxy* module is to convert the SIP transport from WebSocket protocol to UDP,

TCP or TLS which are supported by all SIP-legacy networks. If your provider or hosted server supports SIP over WebSocket (e.g. Asterisk or Kamailio) then, you can bypass the module and connect the client directly to the endpoint. Bypassing the *SIP Proxy* is not recommended if you're planning to use the *RTCWeb Breaker* or *Media Coder* modules as this will requires maintaining two different connections.

There are no special requirements for the end server to be able to talk to the *Proxy module*.




```
F1 REGISTER Web Browser -> webrtc2sip (transport WS)
REGISTER sip:proxy.example.com SIP/2.0
Via: SIP/2.0/WS df7jal23ls0d.invalid;branch=z9hG4b5
From: sip:browser@example.com;tag=abc
To: sip:browser@example.com
Call-ID: abcdefghijklmnopqrstuvwxyz
CSeq: 1 REGISTER
Max-Forwards: 70
Contact: <sip:browser@df7jal23ls0d.invalid;transport=ws>
```

This request contains an invalid IP address in the `Contact` (**df7jal23ls0d.invalid**) and `Via` headers because there is no way for the browser to retrieve its local binding *IP:Port address*. The transport type is WebSocket (**ws**). A SIP-legacy server cannot handle this request as the transport is probably not supported and the IP address and port are not valid (not reachable), this is why we need the *SIP Proxy* module to patch the request before forwarding.

```
F2 REGISTER webrtc2sip -> SIP-legacy Network (transport UDP)
REGISTER sip:proxy.example.com SIP/2.0
Via: SIP/2.0/UDP 66.66.66.66:5060;branch=z9hG4b5;rport
Via: SIP/2.0/TCP 192.168.0.9:55210;rport;branch=z9hG4b6;ws-hacked=WS
From: sip:browser@example.com;tag=abc
To: sip:browser@example.com
Call-ID: abcdefghijklmnopqrstuvwxyz
CSeq: 1 REGISTER
Max-Forwards: 70
Contact: <sip:browser@66.66.66.66:5060;transport=udp>
```

The `Via` header is patched to use a well-known protocol (*TCP*) and to use the IP address and port (**192.168.0.9:55210**) from which the request has been received (WebSocket connection).

The *SIP Proxy* adds it's own `Via` header (**66.66.66.66:5060**) where it's willing to receive the response. The same address is used in the Contact header for incoming requests (e.g. `INVITE`).

Before forwarding the request the SIP Proxy determines the destination address using the following algorithm:

```
char* dst_host = get_host(request_uri); // dst_host = "proxy.example.com"
int dst_port = 5060;
if(hast_route(request)){ // there is no route
      dst_host = get_host(first_route);
      dst_port = get_port(first_route);
}
if((dns_result = dns_srv(dns_naptr(dst_host)))){
      dst_host = get_host(dns_result);
      dst_port = get_port(dns_result);
}
```

3.2 RTCWeb Breaker

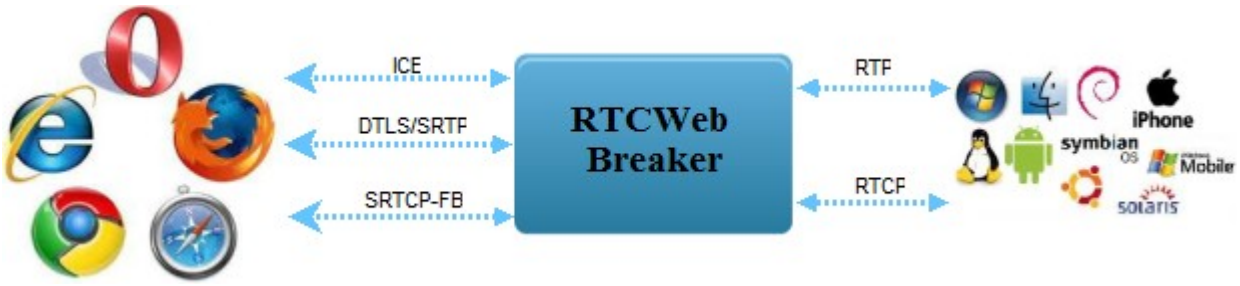


Figure 3: RTCWeb Breaker architecture

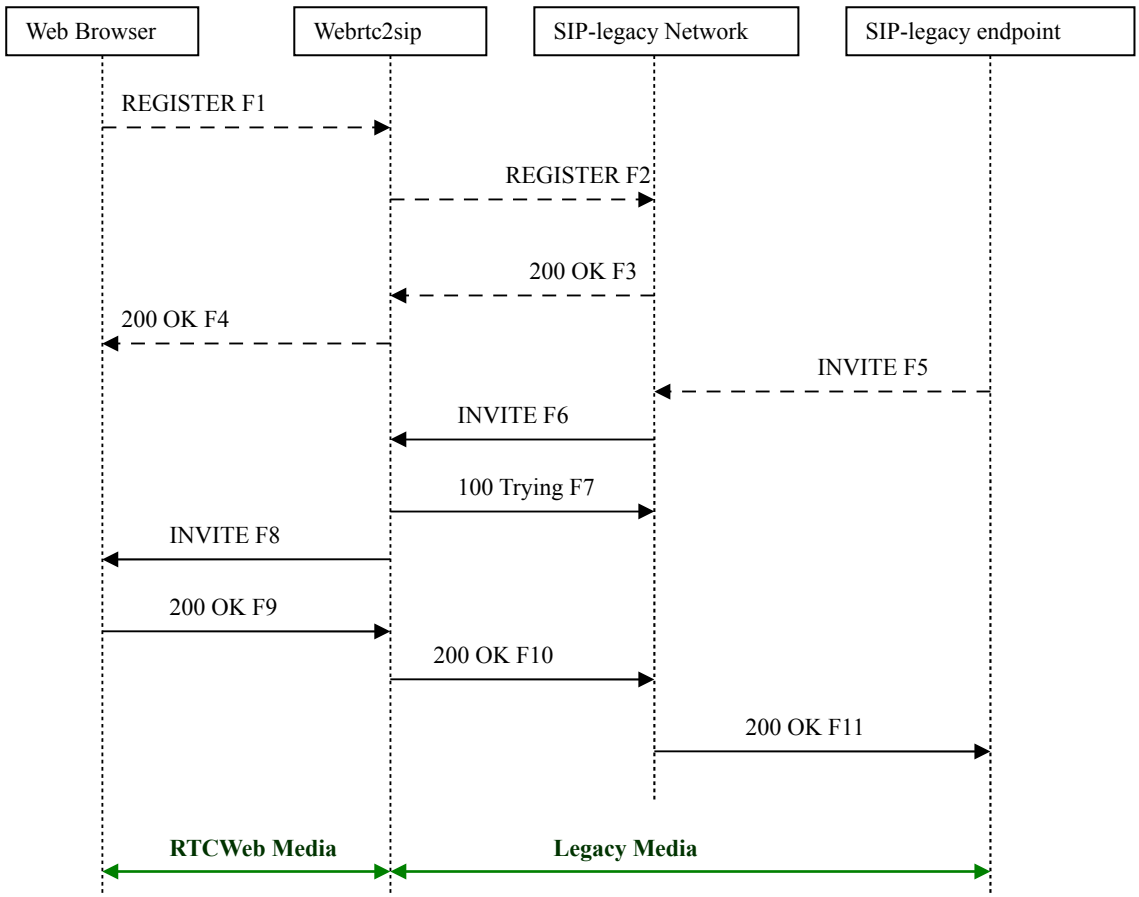
The RTCWeb specifications make support for **ICE** and **DTLS/SRTP** mandatory. The problem is that many SIP-legacy endpoints (e.g. PSTN network) do not support these features. It's up to the *RTCWeb Breaker* to negotiate and convert the media stream to allow these two worlds to interop.

For example, FreeSWITCH do not support ICE which means it requires the *RTCWeb Breaker* in order to be able to connect the browser to a SIP-legacy endpoint.

The *RTCWeb Breaker* is disabled by default and it's up to the client to enable it before registering to the server.

To activate the *RTCWeb Breaker*, the client must include "*rtcweb-breaker=yes*" as Uri parameter of its AoR (**A**ddress **o**f **R**ecord). When the module is enabled it acts as a *b2bua* (**b**ack **2** **b**ack **u**ser **a**gent) by answering to the *INVITE* and making a new one.

Figure 4: Enabling RTCWeb Breaker on sipml5



F1 REGISTER web browser -> webrtc2sip (transport WSS)

-- TODO--

F2 REGISTER webrtc2sip -> SIP-legacy network (transport UDP)

-- TODO --

F3 200 OK SIP-legacy network -> webrtc2sip (transport UDP)

--TODO--

F4 200 OK webrtc2sip -> web browser(transport WSS)

--TODO--

F4 200 OK webrtc2sip -> web browser(transport WSS)

--TODO--

F5 INVITE SIP-legacy endpoint -> SIP-legacy network (transport UDP)

--TODO--

F6 INVITE SIP-legacy network -> webrtc2sip (transport UDP)

--TODO--

F7 100Trying webrtc2sip -> SIP-legacy network (transport UDP)

--TODO--

F8 INVITE webrtc2sip -> web browser (transport WSS)

--TODO--

```
F9 200 OK web browser -> webrtc2sip (transport WSS)
```

--TODO--

```
F10 200 OK webrtc2sip -> SIP-legacy network (transport UDP)
```

--TODO--

```
F11 200 OK SIP-legacy network -> SIP-legacy endpoint (transport UDP)
```

--TODO--

3.3 Media Coder

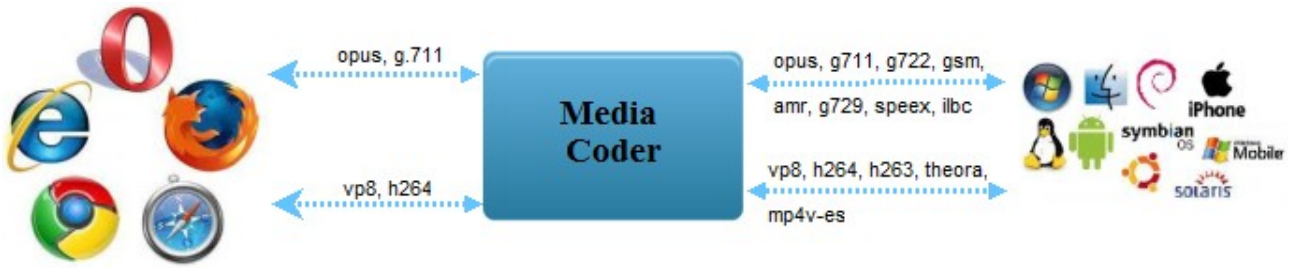


Figure 5: Media Coder architecture

The RTCWeb standard defined two MTI (**M**andatory **T**o **I**mplement) audio codecs: *opus* and *g.711*.

For now there are intense discussions about the MTI video codecs. The choice is between *VP8* and *H.264*. VP8 is royalty-free but not widely deployed while H.264 *AVC* is not free but widely deployed. Google has decided to use *VP8* in Chrome while Ericsson uses *H.264 AVC* in Bowser. Mozilla and Opera Software will probably use *VP8* and Microsoft *H.264 AVC*. As an example, the *Media Coder* will allow to make video calls between Chrome and Bowser. Another example is calling a Telepresence system (e.g. Cisco) which most likely uses *H.264 SVC* from Chrome.

The *Media Coder* is enabled using the xml configuration file and requires *RTCWeb breaker* module to be enabled.

4 Configuration

The gateway is configured using an xml file named *config.xml* and stored in the same folder where the gateway is running.

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
  <debug-level>INFO</debug-level>

  <transport>udp;*;10060</transport>
  <transport>ws;*;10060</transport>
  <transport>wss;*;10062</transport>

  <enable-100rel>no</enable-100rel>
  <enable-media-coder>no</enable-media-coder>
  <enable-videojb>yes</enable-videojb>
  <rtp-buffsize>65535</rtp-buffsize>
```

```
  <avpf-tail-length>100;400</avpf-tail-length>
  <srtp-mode>optional</srtp-mode>

  <codecs>pcma;pcmu;gsm;vp8;h264-bp;h264-mp;h263;h263+</codecs>

  <nameserver>66.66.66.66</nameserver>
  <nameserver>77.77.77.77</nameserver>

  <ssl-certificates>/tmp/priv.pem;/tmp/pub.pem;cacert.pem</ssl-certificates>

</config>
```

Sample 1: config.xml

<debug-level />

Define the minimum debug-level to display.

Format: debug-level-value

Debug-level-value = INFO | WARN | ERROR | FATAL

<transport />

Each entry defines a protocol, local IP address and port to bind to.

Format: proto-value;local-ip-value;local-port-value

proto-value: udp | tcp | tls | ws | wss

"ws" protocol defines WebSocket and "wss" the secure version. At least one WebSocket transport must be added to allow the web browser to connect to the server. The other protocols (tcp, tls and udp) are used to forward the request from the web browser to the SIP-legacy network.

local-ip-value: Any valid IP address. Use star (*) to let the server choose the best local IP address to bind to. Examples: udp;*;5060 or ws;*;5061 or wss;192.168.0.10;5062

local-port-value: Any local free port to bind to. Use star (*) to let the server choose the best free port to bind to. Examples: udp;*;*, ws;*;*, wss;*;5062

<enable-100rel>

Indicates whether to enable SIP 100rel extension.

Format: enable-100rel-value

enable-100rel-value: yes|no

<enable-media-coder />

Format: enable-media-coder-value

enable-media-coder-value: yes|no

Indicates whether to enable the Media Coder module or not. This option requires the RTCWeb Breaker to be enabled at the web browser level. When the Media Coder is enabled the gateway acts as a b2bua and both audio and video streams are transcoded if the remote peers don't share same codecs.

<enable-videojb />

Format: enable-videojb-value

enable-videojb-value : yes | no

This option is only useful if the RTCWeb Breaker module is enabled at the web browser side. Enabling video jitter buffer gives better quality and improve smoothness. No RTCP-NACK messages will be sent to request dropped RTP packets if this option is disabled.

<rtp-buffsize />

Format: rtp-buffsize-value

rtp-buffsize-value: Any positive 32 bits integer value. Recommended: 65535.

Code usage:

```
setsockopt(SOL_SOCKET, SO_RCVBUF, rtp-buffsize-value);
setsockopt(SOL_SOCKET, SO_SNDBUF, rtp-buffsize-value);
```

Defines the internal buffer size to use for RTP sockets. The higher this value is, the lower will be the RTP packet loss. Please note that the maximum value depends on your system (e.g. 65535 on Windows). A very high value could introduce delay on video stream and it's highly recommended to also enable videojb option.

<avpf-tail-length />

Format: avpf-tail-length-min;avpf-tail-length-max

avpf-tail-length-min: Any positive 32 bits integer

avpf-tail-length-max: Any positive 32 bits integer

Defines the minimum and maximum tail length used to honor RTCP-NACK requests. This option require the Media Breaker module to be enabled on the web browser size. The higher this value is, the better will be the video quality. The default length will be equal to the minimum value and it's up to the server to increase this value depending on the number of unrecoverable packet loss. The final value will be at most equal to the maximum defined in the xml file. Unrecoverable packet loss occures when the b2bua receive an RTCP-NACK for a sequence number already removed (very common when network RTT is very high or bandwidth very low).

<srtp-mode />

Format: srtp-mode-value

srtp-mode-value: none | optional | mandatory

Defines the SRTP mode to use for negotiation when the RTCWeb Breaker is enabled. Please note that only optional and mandatory modes will work when the call is to a WebRTC endpoint.

Based on the mode, the SDP for the outgoing INVITEs will be formed like this:

- *none: pofile = RTP/AVP; no crypto lines*
- *optional: profile = RTP/AVP; two crypto lines*
- *mandatory: profile = RTP/SAVP; two crypto lines*

<codecs />

*Format: codec-name (; codec-name)**

codec-name: pcma|pcmu|amr-nb-be|amr-nb-oa|speex-nb|speex-wb|speex-uwb|g729|gsm|g722|ilbc|h264-bp|h264-mp|vp8|h263|h263+|theora|mp4v-es

Defines the list of all supported codecs. Only G.711 is natively supported and all other codecs have to be enabled when building the Doubango IMS Framework source code.

<nameserver />

```
Format: nameserver-value

nameserver-value: Any IPv4 or IPv6 address.

Defines additional entries for DNS servers to use for SRV and NAPTR queries. Please
note that this option is optional and should be used carefully.

On Windows and OS X the server will automatically load these values using APIs provided
by the OS. On linux, the values come from /etc/resolv.conf. The port must not be
defined and the gateway will always use 53.
```

```
<ssl-certificates />

Format: private-key-value;public-key-value;cacert-key-value.

private-key-value: A valid path to a PEM file.

public-key-value: A valid path to a PEM file.

cacert-key-value: A valid path to a certificate autority file. Should be equal to *.

Code usage:

SSL_CTX_use_PrivateKey_file(ssl_ctx, private-key-value, SSL_FILETYPE_PEM);

SSL_CTX_use_certificate_file(ssl_ctx, public-key-value, SSL_FILETYPE_PEM);

SSL_CTX_load_verify_locations(ssl_ctx, cacert-key-value, CaPath);
```

5 Building source code

This section explains how to build the project using CentOS 64 but could be easily adapted for Linux, Windows or OS X.

webrtc2sip gateway depends on Doubango IMS Framework v2.0.

1. Preparing the system

```
sudo yum update

sudo yum install make libtool autoconf subversion git wget
```

5.1 Building Doubango IMS Framework

Doubango is an IMS framework and contains all signaling protocols (SIP, SDP, WebSocket…) and media engine (RTP stack, audio/video codecs…) required by webrtc2sip gateway.

The first step is to checkout Doubango 2.0 source code:

```
svn checkout http://doubango.googlecode.com/svn/branches/2.0/doubango doubango
```

1. Building **libsrtp**

libsrtp is required.

```
wget http://srtp.sourceforge.net/srtp-1.4.2.tgz

tar -xvzf srtp-1.4.2.tgz

cd srtp

./configure --enable-pic && make && make install
```

2. Installing **openssl**

openssl is required if you want to use the *RTCWeb Breaker* module or Secure WebSocket transport (WSS).

```
yum install openssl-devel
```

3. Building **libspeex** and **libspeexdsp**

libspeex (audio codec) an libspeexdsp (audio processing and jitter buffer) are optional. It's highly recommended to enable libspeexdsp.

```
wget http://downloads.xiph.org/releases/speex/speex-1.2beta3.tar.gz
tar -xvzf speex-1.2beta3.tar.gz
cd speex-1.2beta3
./configure --disable-oggtest && make && make install
```

4. Building **YASM**

YASM is only required if you want to enable VPX (VP8 video codec) or x264 (H.264 codec).

```
wget http://www.tortall.net/projects/yasm/releases/yasm-1.2.0.tar.gz
tar -xvzf yasm-1.2.0.tar.gz
cd yasm-1.2.0
./configure && make && make install
```

5. Building **libvpx**

Date: December 1, 2012.

libvpx adds support for VP8 and is optional but highly recommended if you want support for video when using Google Chrome or Mozilla Firefox.

```
git clone http://git.chromium.org/webm/libvpx.git
cd libvpx
./configure --enable-realtime-only --enable-error-concealment --disable-examples --en-
able-vp8 --enable-pic --enable-shared --as=yasm
```

6. Building **libyuv**

libyuv is optional. Adds support for video scaling and chroma conversion.

```
mkdir libyuv && cd libyuv
svn co http://src.chromium.org/svn/trunk/tools/depot_tools .
./gclient config http://libyuv.googlecode.com/svn/trunk
./gclient sync && cd trunk
make -j6 V=1 -r libyuv BUILDTYPE=Release
make -j6 V=1 -r libjpeg BUILDTYPE=Release
cp out/Release/obj.target/libyuv.a /usr/local/lib
cp out/Release/obj.target/third_party/libjpeg_turbo/libjpeg_turbo.a /usr/local/lib
mkdir --parents /usr/local/include/libyuv/libyuv
cp -rf include/libyuv.h /usr/local/include/libyuv
cp -rf include/libyuv/*.h /usr/local/include/libyuv/libyuv
```

7. Building **opencore-amr**

opencore-amr is optional. Adds support for AMR audio codec.

```
git clone git://opencore-amr.git.sourceforge.net/gitroot/opencore-amr/opencore-amr
autoreconf --install && ./configure && make && make install
```

8. Building **libgsm**

libgsm is optional. Adds support for GSM audio codec.

```
wget http://www.quut.com/gsm/gsm-1.0.13.tar.gz
tar -xvzf gsm-1.0.13.tar.gz
cd gsm-1.0-pl13 && make && make install
#cp -rf ./inc/* /usr/local/include
#cp -rf ./lib/* /usr/local/lib
```

9. Building **g729**

G729 is optional. Adds support for G.729 audio codec.

```
svn co http://g729.googlecode.com/svn/trunk/ g729b
cd g729b
./autogen.sh && ./configure --enable-static --disable-shared && make && make install
```

10. Building **iLBC**

iLBC is optional. Adds support for iLBC audio codec.

```
svn co
http://doubango.googlecode.com/svn/branches/2.0/doubango/thirdparties/scripts/ilbc
cd ilbc
wget http://www.ietf.org/rfc/rfc3951.txt
awk -f extract.awk rfc3951.txt
./autogen.sh && ./configure
make && make install
```

11. Building **x264**

Date: December 2, 2012

x264 is optional and adds support for H.264 video codec (requires FFmpeg).

```
wget ftp://ftp.videolan.org/pub/x264/snapshots/last_x264.tar.bz2
tar -xvjf last_x264.tar.bz2
# the output directory may be difference depending on the version and date
cd x264-snapshot-20121201-2245
./configure --enable-static --enable-pic && make && make install
```

12. Building **FFmpeg**

Date: December 2, 2012

FFmpeg is optional and adds support for H.263, H.264 (requires x264) and MP4V-ES video codecs.

```
git clone git://source.ffmpeg.org/ffmpeg.git ffmpeg

cd ffmpeg

./configure \

--extra-cflags="-fPIC" \

--extra-ldflags="-lpthread" \

\

--enable-pic --enable-memalign-hack \

--enable-shared --disable-static \

--disable-network --disable-protocols --disable-pthreads \

--disable-devices --disable-filters --disable-bsfs --disable-muxers --disable-demuxers
--disable-parsers --disable-hwaccels \

--disable-ffmpeg --disable-ffplay --disable-ffserver \

--disable-encoders --disable-decoders \

--disable-zlib \

\

--enable-gpl \

\

--disable-debug \

\

--enable-encoder=h263 --enable-encoder=h263p --enable-decoder=h263 \

\

--enable-encoder=mpeg4 --enable-decoder=mpeg4 \

\

--enable-libx264 --enable-encoder=libx264 --enable-decoder=h264
```

13. Building **Doubango**

Minimal build

```
cd doubango && ./autogen.sh && ./configure --with-ssl --with-srtp
```

Recommended build

```
cd doubango && ./autogen.sh && ./configure --with-ssl --with-srtp --with-speexdsp --en-
able-speexjb --with-ffmpeg --with-h264
```

Full build

```
cd doubango && ./autogen.sh && ./configure --with-ssl --with-srtp --with-vpx --with-yuv
--with-amr --with-speex --with-speexdsp --enable-speexresampler --enable-speexjb --en-
able-speexdenoiser --with-gsm --with-ilbc --with-g729 --with-ffmpeg --with-h264
```

5.2 Building webrtc2sip

webrtc2sip depends on Doubango IMS Framework v2.0 and libxml2.

The first step is to checkout the source code:

```
svn co http://webrtc2sip.googlecode.com/svn/trunk/ webrtc2sip
```

1. Installing libxml2

```
yum install libxml2-devel
```

2. Building webrtc2sip

```
export PREFIX=/opt/webrtc2sip

# use --with-doubango=PATH to set path to the doubango installation ('lib' and 'include' folders).

cd webrtc2sip && ./autogen.sh && ./configure --prefix=$PREFIX

make clean && make && make install

cp -f ./config.xml $PREFIX/sbin/config.xml
```

6 Interoperability

This section contains good tips to help you to debug some issues you can find when you're trying to make/receive calls to/from well-known SIP clients or servers using a web browser. Please note that if your preferred web browser is Google Chrome then, **we highly recommend using the STABLE version**.

6.1 Servers

This section explains know issues and how to tackle them.

6.1.1 Asterisk

Date: November 29, 2012

There are some issues (on both Asterisk and Chrome) to get both way audio and video when using Google Chrome stable. There are two solutions.

1. Patching Asterisk: This is only recommended if you're a developer and trying to learn new cool features. Please note that this will not allow video to flow as Asterisk doesn't support *VP8*. For more information on how to patch Asterisk, visit http://code.google.com/p/sipml5/wiki/Asterisk
2. Enabling the RTCWeb Breaker: This is the recommended solution and it allows both audio and video to flow. Video stream will flow even if the web browser and the SIP client/server do not share the same codecs (thanks to the *Media Coder* module).

6.1.2 FreeSWITCH

The problem here is that *FreeSWITCH* do not support *ICE* and some other mandatory *RTCWeb* features. Enabling the *RTCWeb Breaker* module (web browser side) is enough to fix the issue.
There is also an issue on the SRTP side. To fix it, please refer to http://code.google.com/p/webrtc2sip/wiki/FAQ?ts=1355271529&updated=FAQ#I_see_"a=crypto_in_RTP/AVP,_refer_to_RFC_3711"_on_my_c

6.2 Web Browsers

6.2.1 Google Chrome

Date: November 29, 2012

We highly recommend using the STABLE version for your tests. Please note that we don't provide any kind of help or support if you're using the DEV or CANARY versions.

3. Chrome uses **SAVPF** profile. The **S** is for secure (SRTP) and the **F** for feedbacks (RFC 4585). If one of these features is not supported by the remote SIP client/server then you have to enable the *RTCWeb Breaker* module (web browser side).
4. Chrome only includes *VP8* video codec which is not supported by most of SIP clients/servers (e.g. xlite, Asterisk…). If your SIP client/server supports *H.264*, *H.263*,

Theora or *MP4V-ES* then, you have to enable both the *RTCWeb Breaker* (web browser side) and *Media Coder* (server side) modules to have video. Please note that the *Media Coder* module will most likely not be enabled on the sipml5.org hosted servers.

6.2.2 Firefox, Safari, IE and Opera

Date: November 29, 2012

--This section intentionally left blank--

6.2.3 Ericsson Bowser

Date: November 29, 2012

Ericsson Bowser does not support Secure RTP (*SRTP*) and only include *H.264* video codec. Bowser can talk to most of SIP clients but is not compatible with Canary or any RTCWeb client.
Enabling the *RTCWeb Breaker* (browser side) will allow Bowser to talk to Chrome for audio only as G.711 is a common codec but video requires the *Media Coder* to be enabled (server side).

6.3 JavaScript SIP stacks

Date: November 29, 2012

--This section intentionally left blank--

7 Security issues

When the *RTCWeb Breaker* module is enabled on the client side (web browser) then, the server will act as a *b2bua* for all incoming and outgoing *INVITE*s to this web browser. Please note that this only apply to the SIP account tied to this particular web browser. Acting as a *b2bua* means the server will generate a completely new request for each *INVITE*. The new *INVITE* request from the *b2bua* could be challenged (*SIP 401/407 response*) by the remote SIP-legacy network which means the *b2bua* must have the SIP account credentials. Instead of sending the username and password to the *b2bua* we transmit an authentication token (***HA1***). Off course there is no possibility to retrieve the password from the token but it's highly recommended not to allow any intermediate node to intercept it and this is why sipML5 automatically use ***secure websocket (WSS)*** when *RTCWeb Breaker* is enabled.

```
HA1 = MD5(username:realm:password)
```

```
INVITE sip:1061@sip2sip.info SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK1tvqE4UJ9VNwxbRNKODUvXQeoDUPL
w2W;rport
From: <sip:13131313@sip2sip.info>;tag=JA2uxtI28xUAM4ZyForT
To: <sip:1061@sip2sip.info>
Contact: "13131313"<sip:13131313@df7jal23ls0d.invalid;rtcweb-breaker=yes;transpo
rt=wss>;impi=13131313;ha1=050a0170e77b5d345388598f70d2d1bf;+sip.ice
Call-ID: e7c9abfc-67ce-3192-75e6-4429cbdf2626
CSeq: 9517 INVITE
```

The above *INVITE* request is received from the web browser when *RTCWeb Breaker* module is enabled. The *b2bua* will not include the ***HA1*** parameter when making a new INVITE to the SIP-legacy network even if a secure transport (e.g. *DTLS or TLS*) is used to forward it.