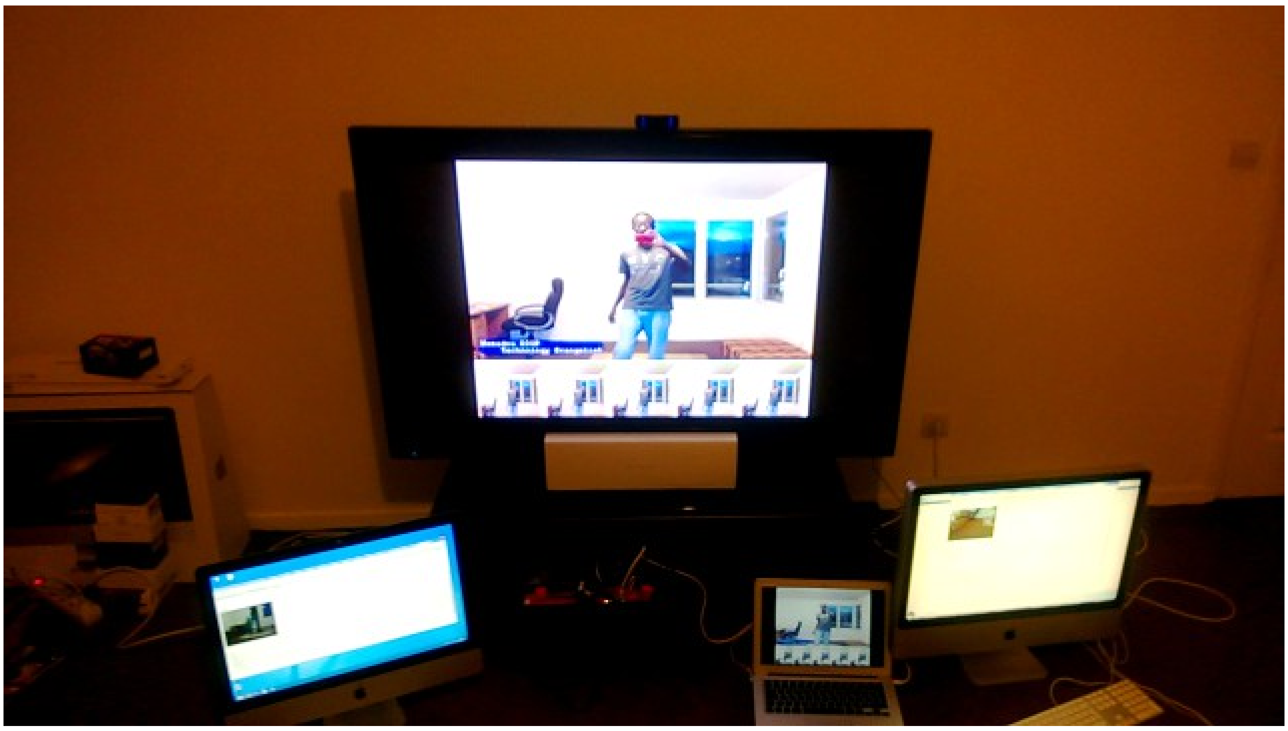
telepresence – the open source SIP TelePresence System

Technical Guide for beta testers

by

Mamadou DIOP
diopmamadou {AT} doubango[DOT]org



telepresence – the open source SIP TelePresence System

License

GPLv3

Commercial

The commercial version is an alternative to GPLv3 if opening your code is a problem. The commercial version also comes with high priority support. For more information, please contact us.

Versioning

Date	Version	SVN revision	Authors	Comments
June 11, 2013	2.0.0	alpha - closed source	Mamadou DIOP	Initial version
June 13, 2013	2.0.1	alpha – closed source	Mamadou DIOP	Update OpenAL build script to use release version instead of git
August 19, 2013	2.1.0	beta – open source	Mamadou DIOP	Fix all issues reported on the public group. Add new configuration parameters: *video-listener-par* *video-speaker-par* *presentation-sharing-enabled* *presentation-sharing-process-local-port* *presentation-sharing-base-folder* *presentation-sharing-app* Add *http* and *https* transports. Add support for presentation sharing (technical description, configuration entries and instructions to install OpenOffice/LibreOffice). Add support for mute/unmute (part of call session management, to be completed with more features). Requires Doubango r989 or later

Table of Contents

Table of Figures

Table of sample configuration entries

1 Foreword

telepresence refers to a set of technologies which allow a person to feel as if they were present, to give the appearance of being present, or to have an effect, via telerobotics, at a place other than their true location (source: Wikipedia).

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).

Our SIP TelePresence product is a free, open source, smart and powerful system to be used on any SIP network without any change for complete integration. This is a good alternative to the well-known commercial products that cost several thousand dollars. The system could be used to connect any SIP endpoint.

This is not a yet another video conference system for the simple reason that we target Full and Ultra HD real-time video @120fps and provide special rooms (XXL screens/monitors, devices, cameras, chairs…) built for amazing experience.

2 Scope

This technical guide is a reference document for beta testers and explains why you need our *telepresence* system and how to leverage its power.

For sure, this technical guide will not be enough to give you a complete help and this is why you can ask on our developer group for any issue. Please note that during the beta phase the group will be locked and you'll need to be invited to be able to post messages. Another way to report issues is to open a ticket on our tracker or ask on the Doubango public group.

3 Main features

This is a short but not exhaustive list of supported features on this beta version:

- Powerful MCU (Multipoint Control Unit) for audio and video mixing
- Stereoscopic (spatial) 3D and stereophonic audio
- Full (1080p) and Ultra (2160p) HD video up to 120fps
- Conference recording to a file (containers: *.mp4, *.avi, *.mkv or *.webm)
- Revolutionary way to share presentations: documents are "streamed" in the video channel to allow any SIP client running on any device to participate.
- Smart adaptive audio and video bandwidth management
- Congestion control mechanism
- SIP registrar
- 4 SIP transports (WebSocket, TCP, TLS and UDP)
- SA (direct connection to SIP clients) and AS (behind a server, such as Asterisk, reSIProcate, openSIPS, Kamailio…) modes
- Support for any WebRTC-capable browser (WebRTC demo client at http://conf-call.org/)
- Mixing different audio and video codecs on a single bridge (h264, vp8, h263, mp4v-es, theora, opus, g711, speex, g722, gsm, g729, amr, ilbc)
- Protecting a bridge with PIN code
- Unlimited number of bridges and participants
- Connecting any SIP endpoint
- Easy interconnection with PSTN
- NAT traversal (Symmetric RTP, RTCP-MUX, ICE, STUN and TURN)
- RTCP Feedbacks (NACK, PLI, FIR, TMMBN, REMB…) for better video experience
- Secure signalling (WSS, TLS) and media (SDES-SRTP and DTLS-SRTP)
- Continuous presence
- Smart algorithm to detect speakers and listeners
- Different video patterns/layouts
- Multiple operating systems (Linux, OS X, Windows…)
- 100% open source and free (no locked features)
- Full documentation
- …and many others

This short list is a good starting point to help you to understand what you could expect from our Telepresence system.

4 Building and installing the product

This is the most important part as it's where you'll decide which features to support/enable. The Telepresence system use Doubango VoIP Framework (requires SVN **r989 or later**) and it's highly recommended to rebuild it if you already have an old version installed because of some new and required features. This section explains how to build the product on CentOS64 but could be easily adapted for any Linux, Windows (MinGW) or OS X (MacPorts).

For any issue, please ask on the telepresence developer group.

4.1 Preparing the system

```
sudo yum update
sudo yum install make libtool autoconf subversion git wget cmake gcc gcc-c++ pkgconfig
```

4.2 Building thirdparties libraries

4.2.1 Building libsrtp

libsrtp is optional unless you want to use WebRTC SIP clients. It's highly recommended. The WebRTC Telepresence demo client requires a system with SRTP enabled.

```
git clone https://github.com/cisco/libsrtp/
cd libsrtp
CFLAGS="-fPIC" ./configure --enable-pic && make && make install
```

You should not use any libsrtp package because the latest dev version is required and building the source **by yourself is highly recommended.**

4.2.2 Building OpenSSL

OpenSSL is required if you want to use TLS, WSS (Secure WebSocket) or DTLS-SRTP (also requires libsrtp). **OpenSSL version 1.0.1 is required if you want support for DTLS-SRTP** which is mandatory for WebRTC implementation from Mozilla (Firefox Nightly or Aurora).

This section is only required if you don't have OpenSSL installed on your system or using version prior to 1.0.1 and want to enable DTLS-SRTP.

A quick way to have OpenSSL may be installing `openssl-devel` package but this version will most likely be outdated (prior to 1.0.1). Anyway, you can check the version like this: `openssl version`.

```
wget http://www.openssl.org/source/openssl-1.0.1c.tar.gz
tar -xvzf openssl-1.0.1c.tar.gz
cd openssl-1.0.1c
./config shared --prefix=/usr/local --openssldir=/usr/local/openssl && make && make
install
```

If you have any error with `"_SSL_CTX_set_tlsext_use_srtp"` when you try to run the telepresence system then, this means you have more than one openssl versions installed. This happens because the configure script detected support for DTLS-SRTP (because of the new openssl) but at runtime your linker try to load the old openssl libraries. The easiest way to fix the issue is to remove the old openssl version (look for libssl).

4.2.3 Building libogg, libvorbis and libtheora

These libraries are optional unless you want to use `*.webm` or `*.mkv` containers.

You can install the devel packages (**recommended**):

```
sudo yum install libogg-devel libvorbis-devel libtheora-devel
```

Or build the source code by yourself:

--This section intentionally left blank--

4.2.4 Building libspeex and libspeexdsp

libspeex (audio codec) is optional but libspeexdsp (audio resampler, jitter buffer…) is **required**.

You can install the devel packages:

```
sudo yum install speex-devel
```

Or build the source by yourself:

```
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 --without-libogg && make && make install
```

4.2.5 Building YASM

YASM is only required if you want to enable and build VPX (VP8 video codec) or x264 (H.264 codec). It's **highly recommended**.

```
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
```

4.2.6 Building libvpx

libvpx adds support for VP8 and is optional but **highly recommended** if you want support for video when using Google Chrome or Mozilla Firefox. libvpx is required if you want to use `*.webm` container or our WebRTC SIP Telepresence client.

You can install the devel packages:

```
sudo yum install libvpx-devel
```

Or build the source by yourself:

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

4.2.7 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
```

4.2.8 Building libopus

libopus is optional but **highly recommended** as it's an MTI codec for WebRTC. Adds support for Opus audio codec.

```
wget http://downloads.xiph.org/releases/opus/opus-1.0.2.tar.gz
tar -xvzf opus-1.0.2.tar.gz
cd opus-1.0.2
./configure --with-pic --enable-float-approx && make && make install
```

4.2.9 Building libgsm

libgsm is optional. Adds support for GSM audio codec.

You can install the devel packages (**recommended**):

```
sudo yum install gsm-devel
```

Or build the source by yourself:

```
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
```

4.2.10 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
```

4.2.11 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
```

4.2.12 Building x264

x264 is optional but **highly recommended** and adds support for H.264 video codec (requires FFmpeg). x264 is required if you want to use `*.mp4` container.

```
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-shared --enable-pic && make && make install
```

4.2.13 Building libfreetype

libfreetype is **required** and used for video overlays.

You can install the devel packages (**recommended**):

```
sudo yum install freetype-devel
```

Or build the source by yourself:

```
wget http://download.savannah.gnu.org/releases/freetype/freetype-2.4.12.tar.bz2
tar -xvjf freetype-2.4.12.tar.bz2
cd freetype-2.4.12
./configure && make && make install
```

4.2.14 Building libfaac

libfaac is optional unless you want support for AAC audio codec or **.mp4* container for recording.

```
wget http://downloads.sourceforge.net/faac/faac-1.28.tar.bz2
tar -xvjf faac-1.28.tar.bz2
cd faac-1.28 && ./configure && make && make install
```

Note: building the tests could fails but you can safely ignore it.

4.2.15 Building FFmpeg

FFmpeg is **required** even if you don't want support for video.

```
# [1] checkout source code
git clone git://source.ffmpeg.org/ffmpeg.git ffmpeg
cd ffmpeg

# [2] grap a release branch
git checkout n1.2

# [3] configure source
./configure \
--extra-cflags="-fPIC" \
--extra-ldflags="-lpthread" \
\
--enable-pic --enable-memalign-hack --enable-pthreads \
--enable-shared --disable-static \
--disable-network --enable-pthreads \
--disable-ffmpeg --disable-ffplay --disable-ffserver --disable-ffprobe \
\
--enable-gpl \
\
--disable-debug \
\
--enable-libfreetype \
\
--enable-libfaac \
\
--enable-nonfree
# [4] build and install
make && make install
```

Remove "`--enable-nonfree`" and "`--enable-libfaac`" if you don't want support for AAC audio codec.

Add "`--enable-libx264`" to force building with support for H.264.

4.2.16 Building OpenAL Soft

OpenAL Soft is optional. Adds support for Stereoscopic (spatial) 3D audio.

```
wget http://kcat.strangesoft.net/openal-releases/openal-soft-1.15.1.tar.bz2
tar -xvjf openal-soft-1.15.1.tar.bz2
cd openal-soft-1.15.1/build
cmake ..
make && make install
```

4.2.17 Building OpenOffice/LibreOffice

OpenOffice (or LibreOffice) are optional and add support for presentation sharing. For information about this feature, check section 5.14. **Version 4.0 or later is required**. Both the application and SDK are required.

This section explain how to install (building would take hours) OpenOffice. LibreOffice could also be used but not recommended (not fully tested).

IMPORTANT: These instructions are for **Linux x86-64** and you must change the paths if you're using a 32-bit system. Run `uname -m` to get your CPU type. All `rpms` could be found at http://www.openoffice.org/download/other.html.

Install OpenOffice application and SDK:

```
## Application (x64) ##
wget http://sourceforge.net/projects/openofficeorg.mirror/files/4.0.0/binaries/en-
US/Apache_OpenOffice_4.0.0_Linux_x86-64_install-rpm_en-US.tar.gz
mkdir -p OpenOfficeApplication && tar -zxvf Apache_OpenOffice_4.0.0_Linux_x86-
64_install-rpm_en-US.tar.gz -C OpenOfficeApplication
rpm -Uvih OpenOfficeApplication/en-US/RPMS/*rpm


## SDK (x64) ##
wget
http://sourceforge.net/projects/openofficeorg.mirror/files/4.0.0/binaries/SDK/Apache_Op
enOffice-SDK_4.0.0_Linux_x86-64_install-rpm_en-US.tar.gz
mkdir -p OpenOfficeSDK && tar -zxvf Apache_OpenOffice-SDK_4.0.0_Linux_x86-64_install-
```

```
rpm_en-US.tar.gz -C OpenOfficeSDK
rpm -Uvih OpenOfficeSDK/en-US/RPMS/*rpm
```

Both OpenOffice application and SDK should be installed into `/opt/openoffice4`. If not true, you'll need to edit the script used to prepare the SDK headers.

Prepare the SDK headers:

```
LD_LIBRARY_PATH=/opt/openoffice4/program:/opt/openoffice4/sdk/lib
/opt/openoffice4/sdk/bin/cppumaker -BUCR -O /opt/openoffice4/sdk/includecpp
/opt/openoffice4/program/types.rdb
```

Please note that the destination folder must be named `includecpp`.

Install java runtime (**required**):

```
yum install java-1.7.0-openjdk
```

Note: Installing OpenOffice application will not add the binary (**soffice**) in your `$PATH` environment variable. The TelePresence system will try to start the program in the background using a relative path unless you have changed `presentation-sharing-app` configuration entry. You can change your `$PATH` environment variable to avoid editing `presentation-sharing-app` but this is not recommend if you're testing different OpenOffice versions. It's also highly recommended to append the folder containing the binary **AFTER** `$PATH`. Appending the folder before `$PATH` will force using shared libraries (e.g. libssl, libcurl…) installed with OpenOffice instead of yours.

Correct: `export PATH=$PATH:/opt/openoffice4/program`

NOT correct: `export PATH=/opt/openoffice4/program: $PATH`

Known Issues:

"`/usr/bin/ld: skipping incompatible /opt/openoffice4/sdk/lib/libuno_sal.so when searching for -luno_sal`": CPU type mismatch (e.g. installed 64-bit libraries on 32-bit OS).

4.2.18 Building Doubango

Doubango VoIP framework 2.0 SVN **r989 or later is required**.

```
svn checkout http://doubango.googlecode.com/svn/branches/2.0/doubango doubango
cd doubango && ./autogen.sh && ./configure --with-speexdsp --with-ffmpeg
make && make install
```

Only few options are used to configure the source code and force enabling mandatory libraries. Any optional library is automatically detected. For example, use "`--with-opus`" to force using Opus audio codec or "`--without-opus`" to avoid automatic detection. You can also specify a path where to search for a library (e.g. `--with-opus=/usr/local`).

Use `configure --help` for more information on supported options.

4.3 Building the Telepresence system

4.3.1 Building the source code

```
svn checkout https://telepresence.googlecode.com/svn/trunk/ telepresence
```

```
cd telepresence
./autogen.sh && ./configure
make && make install
```

If no prefix is defined then, the binaries will be installed into **/usr/local/sbin**.

4.3.2 Installing the configuration and fonts files

This is only required for first-time installations and will **override** any existing configuration file.

```
make samples
```

We highly recommend using our WebRTC SIP telepresence client to test the system.

5 Technical details

5.1 SA versus AS modes

The server supports two modes: **SA** (stand-alone) and **AS** (application server). These modes are not exclusive and no special configuration is needed.

The SA mode allows any SIP client to directly connect to the system without any intermediate node. If the client requires to be registered then, you can enable this option as explained in section 6.4 because the SIP registrar mode is OFF by default.

The AS mode is useful if you already have your own SIP network/server and want to integrate the Telepresence system as an application server. This mode has been tested against Asterisk, reSIProcate, openSIPS and Kamailio. This is as easy as forwarding any INVITE (based on some criterias, e.g. [*domain name equal to **@conf-call.org***]) received by the SIP registrar to the Telepresence system.

5.2 Client Requirements

```
--This section intentionally left blank--
```

5.3 Bandwidth management and congestion control

In this current beta version we only focus on video bandwidth.

The upload and download video bandwidth settings have to be defined using the configuration file as explained at 6.12.1. The maximum download bandwidth is signaled to the remote endpoints using the SDP (b=AS:X attribute as per RFC 3556) and RTCP-REMB packets (as per draft-alvestrand-rmcat-remb-02). In this beta version, RTCP-TMMBN (RFC 5104) packets are deserialized but not processed by the system.

The congestion control manager could be enabled using the configuration file as per section 6.12.1. In this beta version, draft-alvestrand-rtcweb-congestion-03 is not fully implemented yet and we're using our own algorithms to compute the bandwidth usage. The computed maximum bandwidth (periodically) when the congestion control is enabled will never be higher than the maximum allowed values defined in your configuration file (this is kind of safe guard).

5.4 Stereoscopic (spatial) 3D audio

Our 3D audio mixer is based on OpenAL Soft and supports up to 256 sources. Your OpenAL version must be at least 1.15.1 and implements ALC_SOFT_loopback extension.

To have a 3D audio, each SIP client should signal its position in the virtual room. The signaling is done using two SIP headers: TP-AudioPosition and TP-AudioVelocity. These two SIP headers must contain an array of three floating numbers.

```
TP-AudioPosition: [0.0f, 0.0f, 0.0f]

TP-AudioVelocity: [0.0f, 0.0f, 0.0f]
```

Using our WebRTC TelePresence client, the 3D settings could be defined at http://conf-call.org/settings.htm.

From OpenAL documentation, *"AL_POSITION specifies the current location of the object in the world coordinate system. Any 3-tuple of valid float values is allowed. Implementation behavior on encountering NaN and infinity is not defined. The object position is always defined in the world* *coordinate system."*

From OpenAL documentation, *"AL_VELOCITY specifies the current velocity (speed and direction) of the object, in the world coordinate system. Any 3-tuple of valid float/double values is allowed. The object AL_VELOCITY does not affect the source's position. OpenAL does not calculate the velocity from subsequent position updates, nor does it adjust the position over time based on the specified velocity. Any such calculation is left to the application. For the purposes of sound processing, position and velocity are independent parameters affecting different aspects of the sounds."*

5.5 Selecting the speaker and listeners

--This section intentionally left blank--

5.6 Audio mixer design

The audio mixer is part of the MCU engine.

The audio mixer supports mixing several streams with different settings (rate, channels, bits per sample or ptime). For example, a bridge can host a conference with two endpoints, one using g711 (8khz, mono, 20ms) and the other using opus (48khz, stereo, 30ms). As you may expect, it's not technically possible to mix two streams with different settings without resampling.

In the audio mixer there is a notion of "pivot settings". "pivot settings" is the audio parameters to which any stream is resampled to, before mixing. The pivot settings are defined using the configuration file as per section 6.11.

The Doubango framework use libspeexdsp for the resampling while the MCU uses libswresample (from FFmpeg). Both libraries are required.

It's very important to understand the notion of "pivot settings" because using wrong values could lead to poor audio quality and high CPU usage.

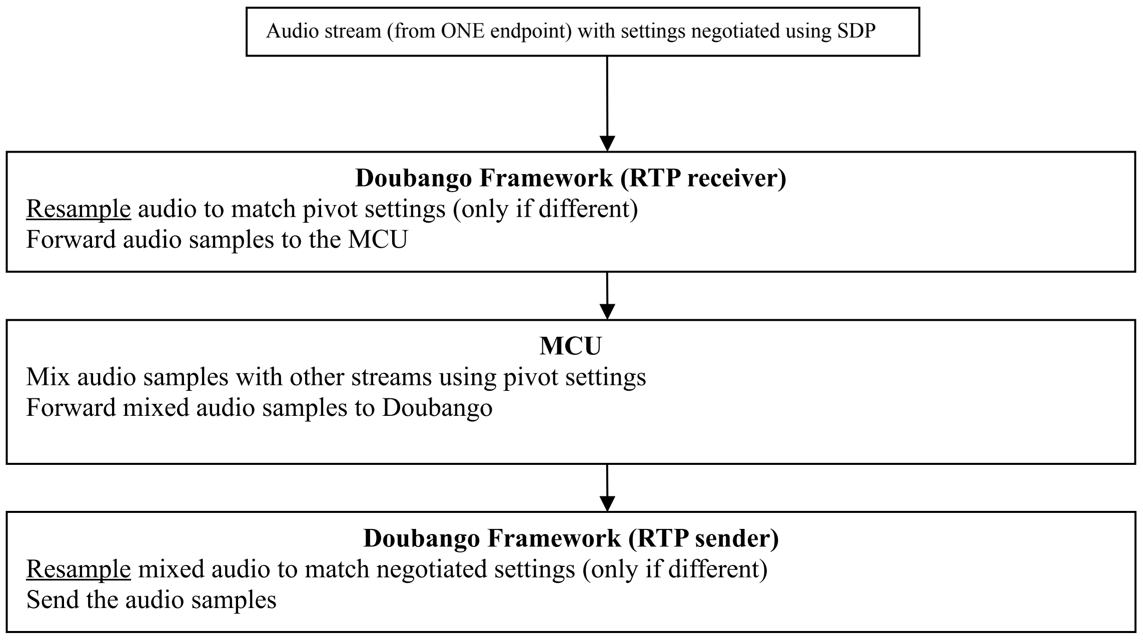

Audio stream (from ONE endpoint) with settings negotiated using SDP
Doubango Framework (RTP receiver)
Resample audio to match pivot settings (only if different)
Forward audio samples to the MCU
MCU
Mix audio samples with other streams using pivot settings
Forward mixed audio samples to Doubango
Doubango Framework (RTP sender)
Resample mixed audio to match negotiated settings (only if different)
Send the audio samples

Figure 1: Audio resampling

From the above figure, you can easily see that the incoming audio samples from an endpoint to the MCU could be resampled up to two times if your pivot and negotiated codec settings mismatch. To minimize the number of audio resampling processes your codec settings have to be as close as possible to those used as pivot. If the settings (pivot, codecs) match, then no resampling will be done.

In this beta version, we support 2d and 3d mixing types. The type of mixing is defined using the configuration file as explained at section 6.11.

The 2d mixing is linear (monophonic or stereophonic) and very basic. No additional thirdparties library is required for this.

The 3d mixing is stereoscopic (spatial) and requires OpenAL Soft.

5.7 Video mixer design

The video mixing and scaling is completely managed using FFmpeg. In this beta version, only 2D mixing is supported. We'll consider support for libyuv in the next versions.

5.7.1 Encoders and decoders

To minimize CPU utilization, all endpoints with the same video codec use the same encoder but different decoders. For example, if you have 7 endpoints using VP8 codec then, the MCU will have 1 encoder and 7 decoders. Using a single encoder gives better CPU performances but more bandwidth than what's needed because if one endpoint requests an IDR then, the prediction chain is restarted for all other peers.

5.7.2 Overlays

The overlays are configured as per section 6.12.6.
The mixed video contains many overlays displayed using FFmpeg filters. There are two kinds of overlays: texts and images. The text overlays use drawtext filter which requires libfreetype to be enabled when building FFmpeg. The image overlays (watermarks) use movie filter and accept any PNG or JPEG file as input. JPEG images are natively supported by FFmpeg (thanks to MPEG4) while PNG requires zlib to be enabled.
The configuration file could be used to define custom font types for the text overlays. The font types must be TrueType-compatible. The default fonts come from ftp://ftp.gnu.org/pub/gnu/freefont.

5.8 Audio quality

--This section intentionally left blank--

5.9 Video quality

This section explains how to make sure to have a good video quality when using the system.

5.9.1 Packet loss recovery

This feature requires the jitter buffer to be enabled.
When a RTP packet is lost then, we request the remote party to send it again using RTCP-NACKs as per RFC 5104. Support for this feature is indicated using the SDP (attribute a=rtcp-fb: * nack). If the remote peer cannot honor the request in a reasonable delay then, the packet is considered as definitely lost. If we're very lucky then, losing a packet would just introduce artifacts. Otherwise (not lucky), this will break the prediction chain and any attempt to decode the video would fail until the next IDR frame. Enabling *zero-artifacts* feature would fix artifact and prediction issues.
Increasing the internal RTP buffer size as per section 6.6 could also help to lower packet loss.

5.9.2 Jitter buffer

The video jitter buffer could be enabled as per section 6.12.3.
Enabling the video jitter buffer introduce small delay (~100ms) but worth it. Buffering the video packets allows requesting missing packets using RTCP-NACK (RFC 5104) and reordering them based on the RTP sequence numbers. It's also up to the jitter buffer to consume any delay or burst to have smooth video according to the frame rate. The video frame rate is negotiated using the SDP but this value will be updated based on the RTP timestamps.

5.9.3 Zero-artifacts

This feature is enabled or disabled as per section 6.12.4.
A video stream contains artifacts when some RTP packets are lost. The MCU try its best to avoid packet loss (see section 5.9.1) but sometimes it fails and this leads to visual artifacts. As the video streams are mixed then, the artifacts will be propagated to all endpoints. When *'zero-artifact'* is enabled then, the MCU pauses the rendering on the stream and sends RTCP-FIR (RFC 5104) to request new IDR frame to repair the prediction chain. Only the stream with the missing RTP packets is paused until the next IDR frame is received. Support for RTCP-FIR is signaled to the remote endpoints using the SDP (attribute a=rtcp-fb: * fir). The MCU sends IDR frames when it receives RTCP-FIR or RTCP-PLI from one of the endpoints.

5.9.4 AVPF tail length

As already explained, RTCP-NACKs are used to ask a peer to send a packet again. In order to be able to honor these requests we need to save the outgoing RTP packets in a queue. The AVPF tail length option defines the minimum and maximum lengths for the queue. The higher these values are the better will be the video quality. The default queue length will be equal to the minimum value and it's up to the MCU 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 configuration file. Unrecoverable packet loss occurs when the MCU receives an RTCP-NACK for an already removed sequence number (very common when network RTT is very high or bandwidth very low).
Setting the AVPF tail length (min, max) is done as per section 6.7.

5.9.5 FEC (Forward Error Correction)

--This section intentionally left blank--

5.9.6 RED (Redundant video data)

--This section intentionally left blank--

5.10 NAT and firewall traversal

This section explains how to tackle NAT and firewall traversal issues.

5.10.1 Symmetric RTP

This feature is enabled or disabled as per 6.5.
The Telepresence system fully supports RFC 4961.

An RTP/RTCP stream is symmetric if the same port is used to send and receive packets. This helps for NAT and firewall traversal as the outgoing packets open a pinhole for the ongoing ones.

The local/outgoing stream (MCU → endpoints) is always symmetric.

If both parties (remote and local) have successfully negotiated ICE candidates then, none will be forced to use symmetric RTP/RTCP.

Let's imagine your Telepresence instance is on a public network and the SIP client/endpoint on private network:

1. Telepresence: Public IP address is **1.1.1.1**

2. Client: Private IP address is **2.2.2.2** and public IP address is **1.1.1.2**
3. The SDP from the client to the Telepresence system will contain client's private IP address (**2.2.2.2**) which is not reachable
4. The RTP/RTCP packets from the client to the server will be received with source IP address equal to the client's public IP address (**1.1.1.2**)
5. If *rtp-symetric-enabled* option is used then, the Telepresence system will send RTP/RTCP packets to **1.1.1.2** (learnt from the received packets) instead of **2.2.2.2** which is private and unreachable.

5.10.2 ICE

This feature is enabled or disabled as per section 6.5.
The Telepresence system fully supports RFC 5245.

ICE is negotiated only if this feature is enabled and incoming SDP (SIP endpoint → MCU) contains candidates. **ICE is mandatory for WebRTC endpoints.**

5.10.3 RTCP-MUX

This feature is enabled or disabled as per section 6.5.
The Telepresence system fully supports RFC 5761.

RTCP-MUX is used to minimize the number of ports and help for NAT traversal and administration.

5.11 Security

This section explains how to secure both signaling and media plans.

5.11.1 Signaling

Two secure signaling protocols are supported: **TLS** and **WSS**. "WSS" is WebSocket secured using TLS. For more information on how to enable these transport protocols using the configuration file, please refer to section 6.2. Both transports require OpenSSL which have to be enabled when building the Doubango framework only. More information on how to configure the SSL certificates at section 6.3.

5.11.2 Media

Both SRTP-SDES (RFC 4568) and SRTP-DTLS (RFC 5763, RFC 5764) are supported. Check section 6.3 for more information on how these features have to be configured.

--This section intentionally left blank–

5.12 Protecting a bridge with password

This feature is configured as per section 6.13.

There are two ways for a SIP client to authenticate to a protected bridge: DTMF or *TP-BridgePin* SIP header. If authentication fails, a SIP 403 response will be returned with a short description.

The DTMF method doesn't require changing your SIP client but is not supported yet in this beta version (on the roadmap for the release version). The second method (using the SIP header) requires some modifications on you SIP client to include this new header. If you are using our WebRTC SIP telepresence demo client then, no modification is needed.

5.13 Recording conference to a file

This feature is configured as per section 6.9.

We support almost any container depending on how you built FFmpeg and which codecs are enabled. Right now we recommend only `*.avi` and `*.mp4` as they are fully tested. `*.mkv` and `*.webm` will also work but not fully tested yet. The audio and video mixing is done using libavformat from FFmpeg.

`*.avi` (recommended): requires FFmpeg with MPEG4 video codec and AC3 audio codec

`*.mp4`: requires FFmpeg with H.264 (libx264 thirdparty library) video codec and AAC audio codec. There is a built-in experimental AAC codec in FFmpeg but the code is intentionally designed to not accept such codec because of random crashes. For AAC audio codec, you'll need to build FFmpeg with libfaac or any other third-party AAC library. Please note that all AAC libraries are not free.

`*.avi` is recommended instead of `*.mp4` for the simple reason that the first one consume less CPU.

The output file will have the bridge identifier as name and container type as extension, e.g. `+336000000.avi`. The file is locked and invalid until the last user quit the bridge.

We highly recommend using VLC to play the output files.

5.14 Presentation sharing

Presentation sharing allows any SIP client to share PowerPoint documents with any client connected to the same bridge. The only technical requirements are support for HTTP(S) (sharer only) and SIP INFO.

This is a revolutionary way to share presentation as the documents are "streamed" in the video channel which means any client supporting video could see it. The slides are extracted from the document using OpenOffice (or LibreOffice) as JPEG pictures, re-encoded (H.264, VP8 or whatever) and mixed in the current video stream. Technically, OpenOffice is not integrated in the system but forked as new process and this is why both the SDK and application are required. The OpenOffice process will be started by the TelePresence system at boot time and added to the same job group as the current process to make sure the child will exit when the parent unexpectedly die. The default port used for the inter-process commutation is **2083** and could be changed using the configuration as per section 6.10.

The TelePresence system supports CORS which means the request could be sent from any domain.

This feature could be tested using our online WebRTC SIP client.

Steps:

1. Publish the PowerPoint document to the bridge using HTTP(S) POST requests.
2. Receive feedbacks from the MCU (SIP INFO messages).
3. Move from slide to slide using SIP INFO messages.
4. Close the presentation session using SIP INFO message.

5.14.1 Publishing the document

To start sharing a PowerPoint document you must have an active video session. You can only share **ONE** presentation at time.

The document is sent to the MCU in **TWO** HTTP(S) requests using the same connection. The first request sends information about the document and the second the content. You must not mix the document information and content.

The TelePresence system must be configured with an **http** or **https** transport (or both) as explained at section 6.2 .

The first request structure:

Element	Value	Availability
Request type	HTTP(S) POST	Mandatory
Request URL	/presentation	Optional
Content-Type	application/json	Mandatory

First request content (JSON):

Field name	Field value	Type	Availability
action	"req_presentation_upload"	String	Mandatory
name	<user defined>	String	Mandatory

type	<user defined>	String	Optional
size	<user defined>	Integer	Optional
bridge_id	<user defined>	String	Optional
bridge_pin	<user defined>	String	Optional
user_id	<user defined>	String	Mandatory

```
POST /presentation HTTP/1.1

Host: 192.168.0.37:20065

Connection: keep-alive

Content-Length: 174

Origin: http://conf-call.org

User-Agent:  Mozilla/5.0  (Windows  NT  6.0)  AppleWebKit/537.36  (KHTML,  like  Gecko)
Chrome/28.0.1500.95 Safari/537.36

Content-type: application/json

Accept: */*

Referer: http://conf-call.org

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8


{"action":"req_presentation_upload","name":"db_pres_01.ppt","type":"application/vnd.ms
-
powerpoint","size":752128,"bridge_id":"100600","bridge_pin":"1234","user_id":"johndoe"
}
```

The second request structure:

Element	Value	Availability
Request type	HTTP(S) POST	Mandatory
Request URL	/presentation	Optional
Content-Type	application/file	Mandatory
Content	<Binary>	Mandatory

5.14.2 Receiving feedbacks

Once the presentation is published the TelePresence system will send SIP INFO messages to give feedbacks about the session state. The SIP INFO messages always contain JSON content.

JSON content:

Field name	Field value	Type	Availability
action	“req_presentation_upload”	String	Mandatory
name	<server defined>	String	Mandatory
state	“opened” “exported” “closed”	String	Mandatory

	“error”		
id	<server defined>	Integer	Optional
page_index	<server defined>	Integer	Optional (0 if missing)
page_count	<server defined>	Interger	Mandatory

5.14.3 Fetching the presentation

Once the presentation is opened (see previous section), you can navigate through it using SIP INFO messages. For security reasons the presentation is tied to your SIP connection to be sure no one else could control it.

JSON content:

Field name	**Field value**	**Type**	**Availability**
action	“req_presentation_goto”	String	Mandatory
page_index	<user defined>	Integer	Mandatory
id	<user defined>	Integer	Optional

5.14.4 Closing the presentation

At any time you can end the presentation session using SIP INFO request.

JSON content:

Field name	**Field value**	**Type**	**Availability**
action	“req_presentation_close”	String	Mandatory
id	<user defined>	Integer	Optional

5.15 Desktop or screen sharing

--This section intentionally left blank—

5.16 Call session management

A call session is managed using SIP INFO messages with JSON content. For now only muting/unmuting the session is supported. Next versions will add support for ejecting a participant, getting the list of participants, getting the call state (packet loss, RTT, audio/video quality…)…

5.16.1 Muting/unmuting

The MCU could detect that a session is muted based on the RTP packets but it's highly recommended to also send a SIP INFO message for confirmation. For audio-only sessions, muting a session without sending a SIP INFO could be interpreted as a crash or network issue which automatically disconnects the call.

When the "hangout" video pattern is selected the MCU renders the speaker's video with the highest quality and size. Detecting a speaker could be problematic when the participants are in a noisy environment. Manually muting/unmuting your session is a way to avoid such issues.

JSON content:

Field name	Field value	Type	Availability
action	"req_call_mute"	String	Mandatory
enabled	<user defined>	Boolean	Mandatory

6 Configuration

The Telepresence system is configured using "*cfg*" files. The main *cfg* file is named "`telepresence.cfg`" and should be on the same directory where the binary is installed unless you run the app with "`--config=PATH`" argument. The source code contains a sample configuration file to use to get started. The sample configuration is installed after successfully building the system and running "`make samples`".

The configuration files are parsed using code generated with Ragel tool.

A configuration file contains comments, sections and entries:

Comments

A comment starts with **#**

```
# I'm a comment
Age = 25 # I'm another comment
```

Sections

A section name must be enclosed by square brackets. The section name is case insensitive.

```
# this is a bridge section
[bridge]
```

Entries

An entry is a key-value-pair and must be tied to a section. Both the key and the value are case insensitive. **The key must not start with a SPACE**.

```
[product] # I'm the section
version = 1.2 # I'm an entry with floating number value
name = telepresence # I'm an entry with a string value
```

6.1 Debugging the system

As a developer, the first action is to edit your configuration file to change the `debug-level` from `ERROR` to `INFO`.

When you connect to the MCU you always have video stream back (you see yourself) but your audio stream is never sent back. For debugging purposes, it could be useful to ask your audio stream back. Hearing your own sound helps testing that everything work. Without loopback audio, you must connect at least two endpoints to test audio (encoding, decoding, streaming, resampling…).

```
debug-level = INFO
debug-audio-loopback = yes
```

Configuration 1: Useful debug settings

`debug-level` - defines the minimum debug level to display. Supported values: `INFO`, `WARN`, `ERROR` and `FATAL`.

`debug-level-loopack` - whether to enable audio loopback for testing (see above for more information).

We require having your `debug-level` equal to `INFO` when reporting/sharing issues.

6.2 Network transports

The system support many network transports for the SIP signaling, presentation uploading, call control... The network connections are declared using "`transport`" configuration entries.

```
transport = udp;*;20060;*
transport = ws;*;20060;4
transport = wss;*;20062;*
transport = tcp;*;20063
transport = tls;*;20064;*
transport = https;*;20065
transport = https;*;20066
```

Configuration 2: Network transports

Format: *protocol-value*;*ip-address-value*;*port-value*;*ip-version-value*.
protocol-value must be `udp` , `tcp` , `tls` , `http`, `https`, `ws` or `wss`
"`ws`" protocol defines WebSocket and "`wss`" the secure version (requires OpenSSL). At least one WebSocket transport must be added to allow a web browser (WebRTC SIP client) to connect to the system. The other protocols (`tcp`, `tls` and `udp`) are used for SIP-legacy devices or PSTN.
local-ip-value is any valid IPv4/IPv6 address or FQDN. Use star (`*`) to let the system 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 is any unused local port to bind to. Use star (`*`) to let the system choose the best unused port to bind to. Examples: `udp;*;*` or `ws;*;*` or `wss;*;5062`
ip-version-value defines the IP version to use. Must be `4`, `6` or `*`. Star (`*`) is used to let the system choose the best one. Using star (`*`) only make sense if *local-ip-value* is a FQDN instead of IP address.
A transport configuration entry must have at least a protocol, IP address (or star) and port (or star). The IP version is optional.
`udp` , `tcp` , `tls` , `ws` and `wss` transports are used to transport SIP messages while `http` and `https` are used to upload presentations.

6.3 Security

The configuration file allows setting the SSL certificate files to be used for TLS and WSS signaling protocols. The certificates are also used for DTLS-SRTP. The Doubango framework must be built with OpenSSL enabled as explained in section 5.11.

```
ssl-private-key = /tmp/ssl.pem
ssl-public-key = /tmp/ssl.pem
ssl-ca = /tmp/ssl.pem
ssl-mutual-auth = no
```

Configuration 3: Setting SSL certificates

`ssl-private-key` - the full path to the PEM file.

`ssl-public-key` - the full path to the PEM file.

`ssl-public-key` - the full path to the PEM file.

`ssl-mutual-auth` - whether the incoming connection requests must fail if the remote peer certificates are missing or do not match the local ones. This only applies to TLS or WSS and is useless for DTLS-SRTP as certificates are always required.

The configuration file also allows setting the SRTP type.

```
srtp-mode = optional
srtp-type = sdes;dtls
```

Configuration: 4 SRTP settings

`srtp-mode` – defines the SRTP mode to use for negotiation. Supported values are `none`, `optional` and `mandatory`. Only `optional` and `mandatory` modes will work if the SIP client is a WebRTC browser as SRTP is required.

Based on the mode, the SDP on the outgoing **INVITE**s will be formed like this:

none:

- profile will be equal to **RTP/AVP**
- no crypto lines or certificate fingerprints will be added

optional:

- profile will be equal to **RTP/AVP**
- two crypto lines will be added if `srtp-type` includes `'sdes'`, plus certificate fingerprints if `srtp-type` also includes `'dts'`.

mandatory:

- profile will be equal to **RTP/SAVP** if `srtp-type` is equal to 'SDES' or **UDP/TLS/RTP/SAVP** if `srtp-type` is equal to `'dtls'`
- two crypto lines will be added if `srtp-type` is equal to `'sdes'` or certificate fingerprints if `srtp-type` is equal to `'dtls'`

`srtp-type` - defines the list of all supported SRTP types. Defining multiple values only make sense if the `srtp-mode` value is equal to `optional` which means we want to negotiate the best one. Supported values are `'sdes'` and `'dtls'`.

DTLS-SRTP requires valid SSL certificates and Doubango source code must be compiled with OpenSSL version 1.0.1 or later.

6.4 SIP registration

Many SIP clients require to be registered (logged in) before being able to make calls. By default, any REGISTER request to the gateway will be rejected.

`accept-sip-reg` entry defines whether to accept incoming SIP REGISTER requests or not (acting as SIP registrar).

```
accept-sip-reg = yes # no to disable
```

Configuration 5: Enabling/disabling SIP registration

When the Telepresence system is behind a SIP registrar (e.g. Asterisk) then, this configuration entry is useless as the REGISTER requests will not be forwarded to the MCU.

6.5 NAT / Firewall traversal

This section shows how to enable or disable symmetric RTP (section 5.10), ICE (section 5.10.2) and RTCP-MUX (section 5.10.3).

```
rtp-symmetric-enabled = yes # no to disable
```

```
ice-enabled = yes # no to disable

icestun-enabled = yes

stun-server = stun.l.google.com;19302;stun-user;stun-password

rtcp-mux-enabled = yes # no to disable
```

Configuration 6: Enabling/disabling NAT traversal features

`rtp-symmetric-enabled` - whether to enable symmetric RTP (RFC 4961) for NAT and firewall traversal

`ice-enabled` - whether to enable ICE (RFC 5245) for NAT and firewall traversal.

`icestun-enabled` - whether to use STUN to gather reflexive addresses or not. This option is useful when the server is on a public network or all peers are on the same local network. In these cases, disabling STUN for ICE will speed up the call setup. Disabling `icestun` is also useful when the system is installed on a PC without access to internet.

`stun-server` - defines the STUN/TURN server to use to gather reflexive addresses for the ICE candidates. If no server is defined then, a default one will be used. The default STUN/TURN server is **numb.viagenie.ca:3478**.

Format: *server-fqdn-value*; *server-port-value*; *user-name-value*; *user-password-value*

server-fqdn-value: A valid IPv4/v6 address or host name.

server-port: A valid port number.

user-name-value: The login to use for TURN authentication. Use star (*) to ignore.

user-password-value: The password to use for TURN authentication. Use star (*) to ignore.

`rtcp-mux-enabled` - whether to enable RTC-MUX (RFC 5761).

6.6 RTP buffer size

`rtp-buffersize` configuration entry is used to define 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 video jitter buffer option.

Code usage:

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

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

Configuration:

```
rtp-buffersize = 65535
```

Configuration 7: Setting RTP buffer size

6.7 AVPF tail length

`avpf-tail-length` configuration entry defines the maximum and minimum queue length used to store the outgoing RTP packets. The stored packets are used to honor incoming RTCP-NACK requests. See section 5.9.4 for more information.

```
avpf-tail-length = 200;500 # min;max
```

Configuration 8: Setting AVPF tail length

6.8 Audio/Video codecs

`codecs` configuration entry defines the list of all supported codecs. Only G.711 and G.722 are natively supported and all other codecs have to be enabled when building the Doubango VoIP framework source code.
Each codec priority is equal to its position in the list. First codecs have highest priority.
Supported values are: *opus*, *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* and *mp4v-es*.

```
codecs = pcma;pcmu;vp8;h264-bp;h264-mp
```

Configuration 9: Setting audio/video codecs

6.9 Recording conference to a file

The configuration file is used to specify whether to record the sessions, which container to use and where to store the output file.

```
record = yes

record-file-ext = mp4
```

Configuration 10: Recording conference to a file

record - whether to record the sessions.
record-file-ext - defines the container to use. Almost any container (*avi*, *mp4*, *webm*, *mkv*…) could be used but this depends on how you built FFmpeg. For more information, please check section 5.12.
We highly recommend using VLC to play the output file.

6.10 Presentation sharing

A presentation is any PowerPoint document and it could be shared from any SIP client running on any device. The presentation is uploaded to the TelePresence system using HTTP(S) POST request which means a http (or https) transport must be configured as explained in section 6.2. More technical details could be found in section 5.14.

```
presentation-sharing-enabled = yes

presentation-sharing-process-local-port = 2083

presentation-sharing-base-folder = ./presentations

presentation-sharing-app = soffice
```

Configuration 11: Presentation sharing

presentation-sharing-enabled - whether to enable presentation sharing. Default is *yes*. The application must be built with OpenOffice (recommended) or LibreOffice SDK to support this feature. This feature will be silently disabled if both SDKs are missing.
presentation-sharing-process-local-port - some implementations requires a third-party application (e.g. OpenOffice or LibreOffice) to export the presentation. The third-party application will be forked to run in the background and the local port ([1024-65535]) is used to communicate with the TelePresence system. For example, if the third-party application is OpenOffice and the local port is equal to **2083** then:

- command string would be: ***soffice*** *-norestore -headless -nofirststartwizard -invisible "-accept=socket,host=localhost,port=2083;urp;StarOffice.ServiceManager"*
- and the connection string would be: *uno:socket,host=localhost,port=2083;urp;StarOffice.ServiceManager*

`soffice` is the OpenOffice application binary. Your `$PATH` environment variable must reference the folder containing the binary or `presentation-sharing-app` must contain a full path (e.g. `/opt/openoffice4/program/soffice`).
`presentation-sharing-base-folder` - base folder where to store uploaded presentations and temporary exported jpeg images. For example, a document named `mypres.ppt` uploaded by bob who is connected to a bridge with number equal to `100600` would have a path equal to `<the base folder>/100600/bob/ mypres.ppt`.
`presentation-sharing-app` - third-party application name. Could be full (e.g. "/opt/openoffice4/program/soffice") or relative ("soffice") path. Relative path requires having the folder containing the application in your `$PATH` environment variable.

6.11 Audio

This section explains how to use settings related to the audio.

6.11.1 Pivot settings

The notion of "pivot settings" is explained in section 5.6.

```
audio-channels = 1

audio-bits-per-sample = 16

audio-sample-rate = 8000

audio-ptime = 20

audio-volume = 1.0f

audio-dim = 2d

audio-max-latency = 200
```

Configuration 12: Audio settings

`audio-channels` - number of audio channels to use. Supported values are `1` and `2`.

`audio-bits-per-sample` - number of bits for each audio sample. Supported values are `8`, `16` and `32`.

`audio-sample-rate`: - audio sample rate. Almost any value is supported: http://en.wikipedia.org/wiki/Sampling_rate. Unit: Hz.

`audio-ptime` - number of milliseconds for each audio frame. The value should be multiple of 10. Supported values: [`1 – 255`]

`audio-volume` - attenuation (or gain) to apply to the mixed audio. Supported values: [`0.0f` - `1.0f`].

`audio-dim` - mixer dimensions. The value must be `2d` (Linear) or `3d` (Spatial). `3d` requires building the system with OpenAL Soft.

`audio-max-latecncy` - maximum audio delay (because of clock drift) before resetting the jitter buffer. The value could be any positive value. Unit: milliseconds.

6.12 Video

This section explains how to use settings related to the video.

6.12.1 Bandwidth and congestion control

There are two kinds of video bandwidths: upload and download.
Upload: Bandwidth (kbps) used by the video stream (RTP + RTCP) from the MCU to a single endpoint.
Download: Bandwidth (kbps) used by the video stream (RTP + RTCP) from the one endpoint to the MCU. This user-defined value will be forwarded to the remote endpoint using the SDP and RTCP-REMB and it's up to this one to respect it or not. For more information, check section 5.3.

The configuration file allows setting the maximum upload and download bandwidths to use. If these values are undefined then, the upload bandwidth is computed following this formula:

```
video-max-upload-bandwidth (kbps) = ((video-width * video-height * video-fps * motion-rank * 0.07) / 1024)
```

For example, 720P video stream @15 frames per second with medium (2) motion rank will consume 1280*720*15*2*0.07 = 1935360 bps = ~1890 kbps unless `video-max-upload-bandwidth` entry is defined.

```
Congestion-ctrl-enabled = yes
video-max-upload-bandwidth = -1 # in kbps, <=0 means undefied
video-max-download-bandwidth = -1 # in kbps, <=0 means undefied
video-motion-rank = 2 # 1(low), 2(medium) or 4(high)
video-fps = 15 # [1 - 120]
```

Configuration 13: Setting video bandwidth and congestion control

`Congestion-ctrl-enabled` – whether to enable draft-alvestrand-rtcweb-congestion-03 and draft-alvestrand-rmcat-remb-01. Check section 5.3 for more information.

`video-max-upload-bandwidth` - defines the maximum bandwidth (kbps) to use for outgoing video stream (per endpoint). If congestion control is enabled then, the bandwidth will be updated based on the network conditions but these new values will never be higher than what you defined in your configuration file

`video-max-download-bandwidth` - defines the maximum bandwidth (kbps) to use for incoming video stream (per endpoint). If congestion control is enabled then, the bandwidth will be updated based on the network conditions but these new values will never be higher than what you defined in your configuration file

`video-motion-rank` - defines the video type. Supported values: *1* (*low*, e.g. home video security systems), *2* (*medium*, e.g conference call) or *3* (*high*, e.g. basketball game).

`video-fps` - defines the video framerate for the mixed stream regardless the input fps. Supported values: [*1 – 120*].

To check available bandwidth: http://www.speedtest.net/
To check bandwidth usage: iftop.

6.12.2 Output size, pixel aspect ratio and letterboxing

The output (MCU → endpoints) mixed video size is independent of the input sizes (from the endpoints). '*video-mixed-size*' configuration entry is used to set the preferred value. Accepted values are: `sqcif(128x98)`, `qcif(176x144)`, `qvga(320x240)`, `cif(352x288)`, `hvga(480x320)`, `vga(640x480)`, `4cif(704x576)`, `svga(800x600)`, `480p(852x480)`, `720p(1280x720)`, `16cif(1408x1152)`, `1080p(1920x1080)`, `2160p(3840x2160)`. If no value is defined then, the mixed video size is assumed to be equal to `vga (640x480)`.
`720p`, `1080p` and `2160p` are commonly named HD, Full HD and Ultra HD.

```
video-mixed-size = vga
```

Configuration 14: Setting output mixed video size

In this beta version, it's not allowed to set arbitrary values because of backward compatibility. The final version will probably allow this.

To draw the speaker and listeners video on the output mixed video we need to resize these frames to feet the destination. The video frames are linearly resized following a specific Pixel Aspect Ratio (*PAR*) before being letterboxed.

```
video-speaker-par = 0:0

video-listener-par = 1:1
```

Configuration 15: Setting Pixel Aspect Ratio

A `PAR` equal to `1:1` means “skip the linear resizing” and a value of `0:0` means “skip both linear resizing and letterboxing”.
Common `PAR` values: `16:9` or `4:3`.

6.12.3 Jitter Buffer

`video-jb-enabled` configuration entry is used to enable or disable video jitter buffer. It's highly recommended to enable video jitter buffer because it's required to have RTCP-FB (NACK, FIR, PLI... as per RFC 5104) fully functional. Enabling video jitter buffer gives better quality and improves smoothness. For example, no RTCP-NACK messages will be sent to request dropped RTP packets if this option is disabled. It’s also up to the jitter buffer to reorder RTP packets.
For more information, check section 5.9.2.

```
video-jb-enabled = yes # no to disable
```

Configuration 16: Enabling or disabling video jitter buffer

6.12.4 Zero-artifacts

It’s up to the MCU to decode all video streams from the endpoints, mix them before sending the result. If RTP packets are lost on one stream then, artifacts will be introduced on the mixed frame (result). Enabling *zero-artifact* feature fix this issue. There are some requirements on the endpoints to have this feature fully functional.
For more information, check section 5.9.3.

```
video-zeroartifacts-enabled = yes # no to disable
```

Configuration 17: Enabling or disabling zero-artifact

6.12.5 Mixing type

--This section intentionally left blank–

6.12.6 Overlays

The configuration file allows managing the overlays (font size and type, position, watermark…).

```
overlay-fonts-folder-path = ./fonts/truetype/freefont
overlay-copyright-text = Doubango Telecom
overlay-copyright-fontsize = 12
# full path: ./fonts/truetype/freefont/FreeSerif.ttf
overlay-copyright-fontfile = FreeSerif.ttf
overlay-speaker-name-enabled = yes
overlay-speaker-name-fontsize = 16
# full path: ./fonts/truetype/freefont/FreeMonoBold.ttf
overlay-speaker-name-fontfile = FreeMonoBold.ttf
overlay-speaker-jobtitle-enabled = yes
overlay-watermark-image-path = ./images/logo35x34.jpg
```

Configuration 18: Video overlays

`overlay-fonts-folder-path` - defines the base folder path where to look for the font types. The default fonts come from ftp://ftp.gnu.org/pub/gnu/freefont. For more fonts (not free), we recommend http://www.dafont.com/.
`overlay-copyright-text` - defines the copyright text to display on the mixed video. Comment the line to disable this feature.
`overlay-copyright-fontsize` - defines the font size to use to draw the copyright text.
`overlay-copyright-fontfile` - defines the font file to use to draw the copyright text on the mixed video. The full path to the TruType file will be 'overlay-fonts-folder-path+"/"+overlay-copyright-fontfile'.
`overlay-speaker-name-enabled` - whether to draw the speaker's name on the mixed video.
`overlay-speaker-name-fontsize` - defines the font size to use to draw the speaker's name (and job title) on the mixed video.
`overlay-speaker-name-fontfile` - defines the font file to use to draw the speaker's name on the mixed video. The full path to the TruType file will be 'overlay-fonts-folder-path+"/"+overlay-speaker-name-fontfile'.
`overlay-speaker-jobtitle-enabled` - whether to draw the speaker's job title on the mixed video.
`overlay-watermark-image-path` - defines the full path to the image to use to watermark the mixed video. Comment the line to disable this feature.

For more information, check section 5.7.2.

To test test the overlays we highly recommended using the WebRTC Telepresence client.

6.12.7 Patterns

--This section intentionally left blank–

6.13 Bridge configuration

In this beta version, global configuration entries are ignored when reused with a `[bridge]` section. For example, setting the audio sample rate at global scoop applies to all bridges but redefining it for a `[bridge]` section will be ignored. In the release version, it will be possible to override almost any configuration entry.

It's not required to add a bridge in order to be able to make conference calls.

You can add as many `[bridge]` sections as you want.

Supported entries are: `id` and `pin-code`.

```
[bridge]
id=10060
pin-code=1234

[bridge]
id=10061
pin-code=0000
```

Configuration 19: Bride settings

`id` - defines the bridge identifier (a SIP client would call "**sip:<the id>@domain**" to connect to this bridge).

`pin-code` – A 4-digit code used to protect the bridge.

For information on how a client authenticates to a bridge, check section 5.12.

7 Testing the system

This section explains how to test the Telepresence system.

As already explained on this document, any SIP client can be used to connect to the Telepresence system but we highly recommend using our WebRTC client. The WebRTC client is hosted at http://conf-call.org/.

Few steps to get started (we assume you have successfully built and installed the system):

1. Build, install and run the system as per section 4.
2. Change the debug level to *INFO* as explained at section 6.1 (not required).
3. The default configuration already has a WebSocket transport listening on port 20060 but the IP address is automatically retrieve at the startup (because of the star (*) in the transport configuration entry). To bind to a specific IP address, change your settings as explained at section 6.2.
4. Starts the system and check the console logs to be sure that all is ok. The logs also show which IP addresses and port are used for each protocol (WS, WSS, TCP, TLS and UDP).
5. Go to http://conf-call.org/settings.htm and enter your Private Identity and the WebSocket connection URL. The Private Identity is a SIP authentication name (without special characters or SPACEs). The connection URL must be the same the transport entry defined in the configuration file (the star (*) after the protocol name -scheme- must be replaced with a valid IP address or host name). Save your settings.
6. Go back to the home page, enter a bridge identifier (any number would work) and press “join” button. If a bridge with this identifier doesn’t exist then, it will be created automatically. Any person calling the same identifier will be part of the conference. No pin code is required unless one is defined in the configuration file.

To report issues: https://groups.google.com/group/opentelepresence.

For issues related to bandwidth, please attach results from http://www.speedtest.net/.

8 Known issues

This is a short list with all known issues (to be fixed before the end of the beta phase).

1. The audio quality on the recorded files is not as good as we expect. This looks like an issue on the PTS and DTS.
2. Mixed video looks stretched when the SIP clients is a mobile in portrait (e.g. iDoubs and IMSDroid). No issue when device is in landscape or using our WebRTC demo client.
3. The algorithm to detect the current speaker and listeners is buggy.
4. 3D (spatial) mixed audio quality is not as good as we expect.

9 Tips

9.1 Lowering CPU

1. Avoid audio resampling

To avoid audio resampling, the SIP clients connecting to a bridge have to use a codec with the sample rate, channels and bits per sample as the "pivot settings". For more information, check section 5.6.
☞ In you configuration file, enable codecs with same settings as the pivot.

2. *Only record sessions if needed*

Do not enable recording if it's not important to you or use *.avi container which consume less CPU than *.mp4 (because of AAC encoder from libfaac).

3. *Use common video codec*

All SIP clients with the same video codec will share a single encoder. Try to use common video codec for all clients. For example, if you have two clients, A and B, with A supporting both H.264 and VP8 and B only H.264 then, you should make sure that A will offer H.264 with highest priority. For more information, check section 5.7.1.
☞ In your configuration file, enable a single video codec if you cannot control the SIP clients.

4. *Use 2d audio mixing*

Enable 2D audio mixing instead of 3D.

5. *Lower mixed video size and fps*

If you have a weak CPU then, consider using a reasonable video size (e.g. VGA) and fps ([15 - 30]).

6. *Multi-threading and ASM*

Make sure to enable YASM and pthread when building FFmpeg, x264 and VP8.

9.2 Lowering bandwidth

1. Use "slow" motion rank (see section 6.12.1)
2. Use small mixed video size (see section 6.12.2)
3. Set the maximum upload and download bandwidth (see section 6.12.1)
4. Use small video frame rate (see section 6.12.1)

☞To test your available bandwidth, we recommend http://www.speedtest.net/.
☞To check bandwidth usage, we recommend iftop.

9.3 Improving audio quality

1. Use Opus (or G.722) audio codec if supported by the SIP clients (see section 6.8).
2. Avoid audio upsampling and downsampling (see section 5.6).
3. If the "pivot settings" use a sample rate (SR) equal to S then, try to use codecs with a SR equal to "S << n" or "S >> n".

9.4 Improving video quality

1. Use Google Chrome as SIP client (check our WebRTC demo client at http://conf-call.org/).
2. Enable "Zero-Artifacts" feature (see section 5.9.3 and 6.12.4)
3. Use a client supporting something close to 16/9 video size to avoid stretching issues
4. Avoid video upsampling and downsampling

9.5 Lowering recorded video file size

--This section intentionally left blank—