Hangout-like Video Conferences with Jitsi Videobridge and XMPP

Emil Ivov[1]
jitsi.org

Summary

About a year ago the Jitsi project developers started work on support for video conference calls. We had had audio conferencing for a while at that point and we were using it regularly in our dev meetings. Video was then our next big challenge so we rolled our sleeves and got to work.

The first choice that we needed to make was how to handle video distribution. The approach that we had been using for audio was for one of the participating Jitsi clients to mix all streams and then send them back to the other participants. This was relatively easy to do for audio and any recent desktop or even laptop machine can easily mix a call with six or more participants.

Video however was a different story. Mixing video into composite images is an extremely expensive affair and one could never achieve this real-time with today's desktop or laptop computers. We had to choose between an approach where the conference organizer would simply switch to the active speaker or a solution where a central node relays all streams to all participants, while every participant keeps sending a single stream. We went for the latter.

This is how Jitsi Videobridge was born: an XMPP server component that focus agents can control via XMPP.

Today Jitsi Videobridge is the only Free/Libre application today that allows for high quality multi-party video conferences today (and we do mean it when we say high quality).

Keywords

VoIP, video conferencing, multi-party conferences, hangouts, XMPP

1 Introduction

Jitsi is an open source communicator that allows secure audio/video calls and conferences, desktop streaming and sharing, instant messaging, file transfer and many others. We like claiming that we are the most advanced and feature rich communicator, and we don't say this lightly.

Some readers may remember the project from previous editions of JRES such as our 2007 [1] and 2011 [2] appearances. Those of you that are interested in a feature update are welcome to have a look at our Jitsi Features page [3] . This paper is about something else: our adventures in multi-party communication and how they have taken us from client-side to the server and cloud development.

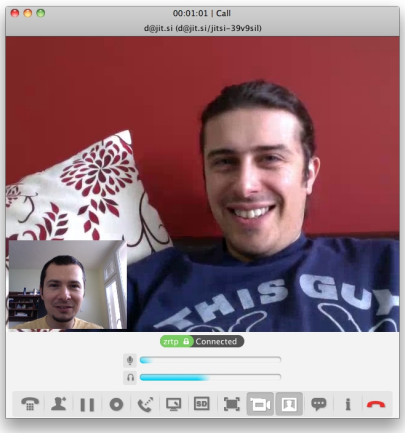

Figure 1: Video call with Jitsi

1. *Comments and questions are welcome at: Emil Ivov <emcho@jitsi.org>*

1.1 Ad hoc audio conferences

As usual, it all started with an itch that we needed scratched. We had to have conferences where small teams of Jitsi developers could get together and have discussions. We didn't want infrastructure requirements, because we all used and tested different SIP [10] and XMPP [8] servers, and the whole process had to come for a minimal cost … preferably none.

With this in mind, it was quite obvious that we needed Jitsi to be capable of mixing audio and hosting conference calls. That's how our conferencing features were first implemented.

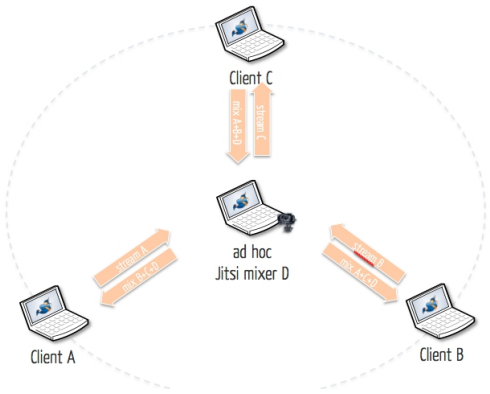

Figure 2: Ad hoc audio conferencing with Jitsi

From a protocol perspective, conferencing is relatively straightforward. Most of the time such calls start as a regular one: Alice is simply calling Bob. Bob could then turn out to be a conference server or another user with a smart client that is capable of making additional calls and then bridging and mixing the audio between them just as in Figure 2.

We implemented this and more. We made sure that every participant would be able to get a rich user interface so that the full list of participants as well as their audio activity would be visible to them (have a look at Figure 3). This wasn't possible with existing protocols at the time so we needed to extend RTP and XMPP a little bit, but we made sure that our changes were then standardized [5] , [6] , [7] .

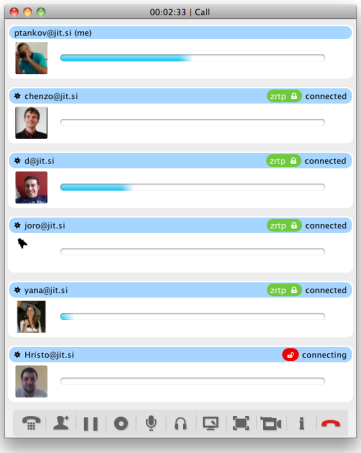

Figure 3: Conference call with Jitsi

We found all this as useful as expected and audio conferencing became one of Jitsi's most popular features. One day however, it just wasn't enough any more and we needed video. Not only would this allow us to see more of each other during these calls, but we would also be able to share slides, code or our screens, through Jitsi's desktop sharing features because they too were using video streams.

1.2 The early days of video conferencing

Here is where things were starting to get complicated. Or so we thought. It is important to understand that the reason Jitsi could easily become a conferencing mixer, was that the process of mixing audio is extremely simple. It is really just a matter of adding numbers together. One does need to pay special attention to keeping these numbers within a special range, or to not sending one's audio back to themselves (which often means producing individual mixes for every user) but even with that in mind, audio mixing remains a process that is sufficiently lightweight so as to be handled by commodity hardware.

The situation is quite different for video … even though it has taken the industry some time to arrive at that conclusion. If we were to handle video conferencing the way we generally handle audio, then we would have to mix video content. The concept of video mixing is that of creating composite images. In other words, if users A, B, C, and D were to participate in a mixed video conferencing call, then they would each start a regular one-to-one session with the mixer and send their video streams to it as usual. In return, they would receive a single video stream that would happen to contain everyone else's content even if a little scaled down (Figure 4).

The simplicity of it all is actually quite appealing for a client: conferences are just as any other call and no special effort is required to support them. Unfortunately things aren't quite so simple at the server side. The reason is that video content mixing requires a huge amount of processing resources.

When performing video content mixing, one needs to decode all incoming frames (one per participant), scale down each one of them, create composite images and then re-encode them once again. A regular, non-hd image stream with approximately 25 frames per second (which is what we generally see with applications such as Skype, Hangouts and Jitsi) and four participants would hence require 100 encodings per second, 100 scale downs, 25 compositions and 25 encodings.

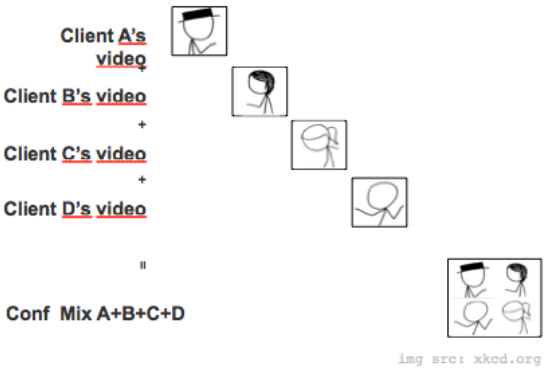

Figure 4: Video mixing

Note: the above numbers assume that all users would be seeing the exact same video stream, which is likely to include their own reflections. While this is less disturbing than audio echo, the effect of seeing themselves could still be relatively distracting to many users. Yet, producing separate mixes for every participant, exponentially increases the processing load on the mixer which makes this a very rare feature in content mixed conferences.

In addition to the cost of processing resources, video content mixing also implies substantial compromises in terms of quality and usability. Let us not forget that every single frame received by a participant has undergone lossy encoding twice, rather than once. Images are scaled down. The video layout is fixed. As mentioned above, it is inconceivable to even remove once own stream from the resulting mix, let alone allow users to turn participants on and off. Finally, central content mixing is bound to add at least 200ms of latency. Note that these problems the last two problems are unaffected by the amount of processing capabilities that are available and are always present.

Yet, for a very long time, this was how video conferencing worked. Obviously, providing that much resources is quite an expensive affair and video conferencing was hence available to a very limited number of users. After all, as expensive as they were, CPU resources costed less than bandwidth.

1.3 Modern video conferencing

Time went by, the Internet evolved, bandwidth prices dropped and the game changed. With broadband becoming a commodity, downloading a stream of three to five megabits per second was no longer a problem and an alternative video conferencing architecture quickly became obvious:

What if, rather than mixing, we were to simply relay it all?

In other words, video conferencing clients keep sending the same streams: the video from their desktop or web cam, only this time, rather than getting one stream in return, they would directly receive everyone else's packets the way they were sent.

The advantages to this approach are numerous. By receiving separate flows, user agents can render them any way they or their users choose. Quality is better as video streams have only undergone encoding once. Latency is not increased by the additional encodings, scalings and decodings.

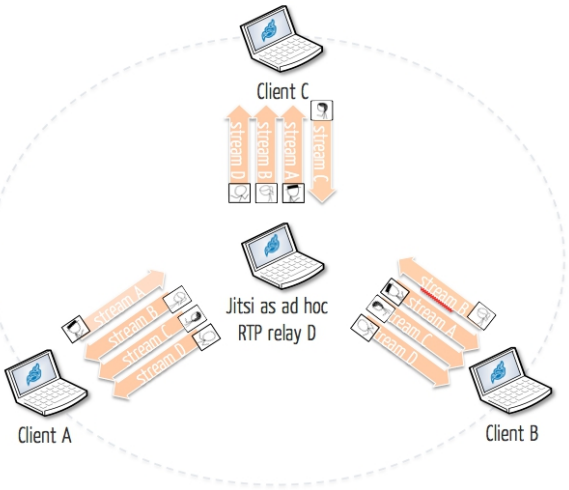

Figure 5: Relayed video conferences

First and foremost of course, video relaying requires hundreds of times less resources than mixing. If done right the operation could even be implemented in routers and more importantly …

2 Ad hoc video conferences with Jitsi

… commodity hardware. In other words, as long as you have the bandwidth, implementing video conferencing with relaying rather than mixing, makes it possible for Jitsi to execute this task on your home laptop or desktop computer.

This is how we first implemented video conferencing. Most of the required code was already in place from our support for audio conferencing. The one new feature that we needed, was the ability to switch packets within our media core and resend them to everyone else. So we added it.

It is important to note that from the start this was intended to be an intermediary step only. While processing-wise, hosting a video conference on your home computer is entirely possible, doing the same on a home internet connection is a lot less so.

A 640x480 video stream with Jitsi takes an average of 200 Kbps for a regular video conference. Movement can easily cause bursts of up to 500 Kbps. This means that a video conference of four participants would require agents to handle a downstream debit of 800 Kbps with potential bursts of up to 2 Mbps. Obviously this is well within the capabilities of most broadband deployments today.

In this same scenario however, a focus agent would need to be prepared to support an average upstream of 1800 Kbps with potential bursts of up to 4500 Kbps. Unfortunately there are very few home Internet subscriptions that would allow clients to sustain upstream bit rates in that range.

For this reason we always knew that an entirely client hosted video solution would not be as plausible as an audio one. We therefore moved to our next step:

3 Jitsi Videobridge and the Colibri Protocol

Simply put, Jitsi Videobridge is an application that takes *libjitsi* [4] , the media and conferencing core of Jitsi and puts it on a server. It is important to understand that nothing else changes. Jitsi Videobridge is NOT a SIP or an XMPP server that agents call and join conferences. It is a remotely controlled server component.

3.1 XMPP as an API

Making *libjitsi* available on a server meant that we had to define an API in order to access it. We chose to do this over XMPP making the Videobridge an XMPP server component. While not a definitive choice, we may still add a second REST API to make the bridge more WebRTC friendly, XMPP does offer several very important advantages.

To begin with, using XMPP meant that we didn't need to worry about authentication. Once an XMPP component establishes a trusted connection with the corresponding XMPP server, it is guaranteed to receive XMPP stanza with correct “from” addresses. It is therefore enough for the videobridge to make sure that these addresses match the domain it is configured for. There is no need for implemented an authentication policy or maintaining a user base. This makes the bridge very easy to integrate in a varying deployments.

The second XMPP feature that made it a compelling choice was the protocol's discovery capabilities. An XMPP client can simply send a query to its server and obtain a list of all available components and their supported features. This meant it would be very easy for Jitsi to only active its videobridge features in environments where they are actually usable.

3.2 Interacting with the videobridge

This is probably the right place for a quick reminder on how VoIP works. Most of you have likely already seen the VoIP trapezoid on Figure 6. Every user is connected to a signalling server authoritative for their domain.

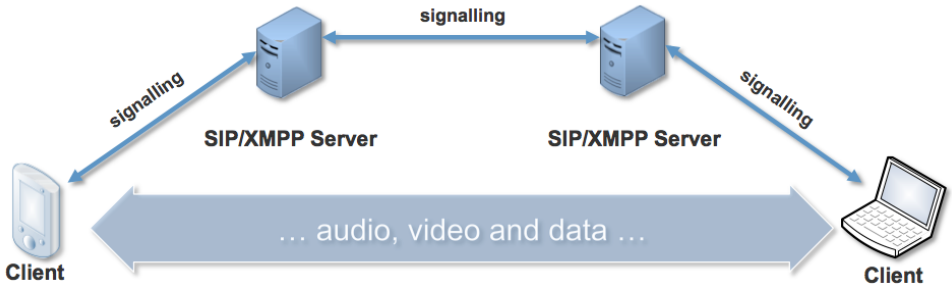

Figure 6: VoIP Trapezoid

Signalling messages such as the initiation, modification or termination of a call, are sent through these servers. Most of the traffic, however, including audio, video and end-to-end data, is expected to be directly exchanged between the participants. The purpose of the model is to allow for a very light infrastructure.

Both SIP and XMPP achieve this model by requiring user agents (be it software or hardware ones) to include their IP addresses and port numbers in the signalling messages and then use them for media exchange. Up until the recent past, the model has caused a variety of problems because of the ever growing popularity of Network Address Translation devices. Most of these problems have since been resolved with protocols such as STUN [11] , TURN [12] and ICE [13] , so we can once again concentrate on one of the main advantages of the trapezoidal model: the possibility for media and signalling to flow through completely different paths.

As explained earlier the point of the Colibri protocol is to allow video conference organisers to communicate with the videobridge in exactly the same way as if they were interacting with it locally. The main difference is that rather than allocating ports locally, clients need to allocate them on the remote component. Once they do this however, session negotiation continues with no additional modifications.

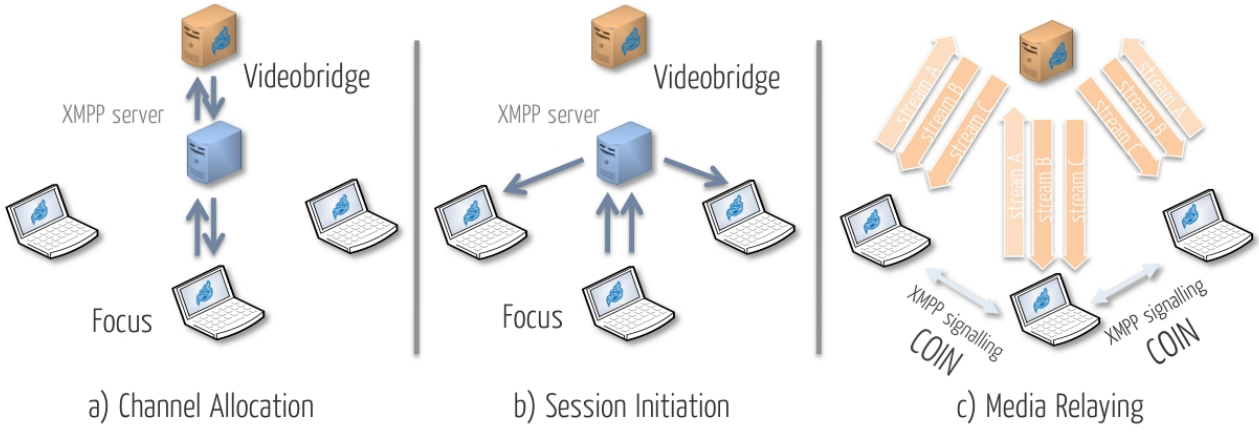

Figure 7: Jitsi Videobridge peration: Channel allocation, session Initiation and media relaying

In other words, before establishing a call, the Focus agent (i.e. the organiser of the conference) will allocate port numbers at the video bridge (Figure 7.a). It would then establish regular sessions with the other participants only, rather than indicating its own IP address, it will use the address:port couple it received from the bridge (Figure 7.b). The remaining conference participants would hence start streaming media to and receiving it from the videobridge (Figure 7.c). When that happens the Focus may choose to send additional information on the participant list using protocols such as COIN [7] .

It is important to note that the Focus is the only entity that has a direct signalling interaction with the videobridge. There is hence no constraint on any of the other participants to support that interaction or use a specific signalling protocol. In the case of Jitsi for example, the videobridge is currently being used in the context of XMPP Jingle [9] calls such as the following:

Figure 8: A video conference with Jitsi and Jitsi Videobridge

4 Jitsi Videobridge and WebRTC

In 2012 Google started a project that allowed the Chrome browser to participate in end-to-end real-time communication sessions. The name of the project was WebRTC [14] and the perceived potential was so strong that the entire browser vendor industry adopted the idea and the effort was moved to official standards bodies such as the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).

The architecture of WebRTC is very similar to that of previously existing protocols such as SIP and XMPP in that it separates signalling and media in exactly the same way. It also uses the same protocols for media transport (SRTP) and NAT traversal (ICE).

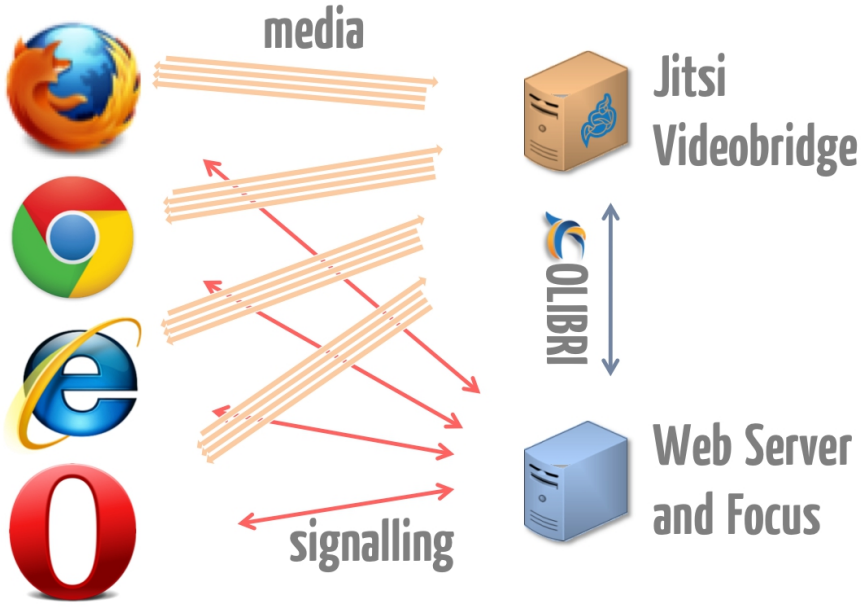

Figure 9: Using Jitsi Videobridge with WebRTC

This means that, with a very reasonable effort, we have been able to make Jitsi Videobridge compatible with browsers. The necessary changes have been about:

- Adding support for the DTLS/SRTP key negotiation mechanism
- Trickle ICE
- SSRC preallocation

Once the above list was implemented Jitsi Videobridge became usable in pure WebRTC deployments or environments where SIP and/or XMPP clients co-exist with browsers.

5 Conclusions and Future Work

Jitsi Videobridge is still a young project and gaining adoption is its first important challenge. Hopefully we would be able to describe videobridge deployments in a future JRES edition.

From a development perspective we hope to be able to address several important aspects such as support for mobile clients, Scalable Video Coding (SVC) and/or simulcasting, retransmission strategies and large scale conferences and teaching sessions.

Making Jitsi Videobridge friendly towards mobile clients would require us to implement switched or selective video relaying so that only some of the streams would be rendered on the mobiles. Similar retransmission strategies would be required for the support of on-line classes and large scale conferences.

SVC and simulcasting would be important for optimizing bandwidth consumption, improving error resilience and usability.

Bibliographie

[1] Emil Ivov et Jean-Marc Muller, SIP Communicator – Un outil open source de communication sur IP adapté à nos laboratoires et à nos universités. *Journées Réseaux*, Novembre 2007.

[2] Emil Ivov, Guillaume Schreiner, Philippe Portelli – Déployer une réelle alternative à Skype dans nos universités en utilisant des outils libres et standardisés. *Journées Réseaux*, Décembre 2011.

[3] Jitsi: A Secure Audio/Video Open Communicator – https://jitsi.org

[4] LibJitsi: An advanced Java media library for secure real-time audio/video communication – https://jitsi.org/libjitsi

[5] J. Lennox, E. Ivov, and E. Marocco, (December 2011) "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication." Internet Engineering Task Force RFC 6464 (Status: PROPOSED STANDARD).

[6] E. Ivov, E. Marocco, and J. Lennox (December 2011) "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication." Internet Engineering Task Force RFC 6465 (Status: PROPOSED STANDARD).

[7] E. Ivov, and E. Marocco (June 2011) "XEP-0298: Delivering Conference Information to Jingle Participants (Coin)." XMPP Standards Foundation, XEP-0298

[8] Peter Saint-Andre, RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core. *Ineternet Engineering Task Force*, March 2011.

[9] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, Joe Hildebrand, XEP-0166: Jingle. XMPP Standards Foundation, Décembre 2009.

[10] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, RFC 3261: Session Initiation Protocol. *Ineternet Engineering Task Force*, June 2002.

[11] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, RFC 5389: Session Traversal Utilities for NAT (STUN). *Internet Engineering Task Force*, Octobre 2008.

[12] Rohan Mahy, Philippe Matthews, et Jonathan Rosenberg, RFC 5766: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). *Internet Engineering Task Force*, Avril 2010.

[13] Jonathan Rosenberg, Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. *Internet Engineering Task Force*, Avril 2010.

[14] WebRTC: a free, open project that enables web browsers with Real-Time Communications (RTC) – http://webrtc.org