On the Seamless Interaction between WebRTC Browsers and SIP-Based Conferencing Systems

Alessandro Amirante, Meetecho S.r.l.

Tobia Castaldi, Lorenzo Miniero, and Simon Pietro Romano, University of Napoli Federico II

Abstract

The growing interest in integrating interactive multimedia features into web applications has recently led to the creation of the W3C WebRTC and the IETF RTCWEB working groups. Such groups are jointly defining both the application programming interfaces and the underlying communication protocols for the setup and management of a reliable communication path between any pair of next-generation web browsers. While the ongoing work is focusing on peer-to-peer communication between browsers, engineers are also facing a new issue, associated with the coexistence of legacy SIP-based systems with the upcoming browser-enabled architectures. We herein discuss how we tackled such an issue, by first identifying interoperability requirements and then presenting a real-world interoperability example dealing with the integration of RTCWEB clients into an existing standards-based collaboration platform.

Background, Rationale, and Motivation

Real-time voice and video communication within browsers is among the topics currently gaining momentum in the two main Internet standardization bodies, the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).

The vision of these two bodies is to standardize an open framework enabling seamless browser-to-browser multimedia applications [1]. This approach represents a real revolution in the world of telecommunications and actually forces us to consider as "legacy" a number of applications and systems that have come to light in the recent past and have been based on telecommunication standards not envisaging the browser among the set of supported end devices. Among such "legacy" frameworks, real-time Session Initiation Protocol (SIP)-based architectures definitely play a major role, having represented the most widespread solution for real-time communication over the Internet. While the ongoing work on peer-to-peer browser-enabled communications is far from complete, there is also a clear need for solutions aimed at the seamless interaction between the huge basis of incumbent SIP solutions and the upcoming products and technologies embracing the newly defined standards.

The focus of this work is on the interoperability issues that have to be faced when trying to let the legacy SIP world and the envisaged browser-to-browser scenarios seamlessly interoperate. We identify the major issues that arise when the end-to-end principle is broken due to the presence of an intermediary, as in the case of a conferencing system relying on a central server in charge of orchestrating the communication among all involved clients. We then discuss potential approaches to the solution of the identified challenges and present the results of our engineering activities aimed at the implementation of an integrated architecture capable of supporting both legacy client devices and browsers equipped with the newly defined real-time capabilities.

設想

The remainder of the article is organized as follows. We briefly describe the main standardization activities in the field of web-enabled real-time communications. A section is devoted to the identification of the issues that must be faced when communication involves some sort of intermediaries. Another section contains the core of our contribution, since it focuses on Meetecho RTCWebLite, our proposal for a solution aimed at the coexistence of legacy SIP-based architectures with real-time web-enabled clients. Related works are presented and analyzed briefly. Finally, we conclude the article by introducing final remarks as well as providing some pointers to open issues and related directions of future investigation.

Real-Time Communications in the Web: A Standards Perspective

The IETF and W3C are interested in different yet complementary facets of the general issue of real-time communication in the web. On one hand, the IETF community is looking into issues like the identification and definition of network-related aspects, including control protocols, connection establishment, and management, and selection of the most suitable encoders and decoders. On the other hand, the W3C is mainly concerned with the definition of proper JavaScript mechanisms aimed at allowing and securing access to input devices, as well as the network protocols chosen for communication.

The two mentioned standardization activities intersect at the boundary between the application-level responsibilities residing in a single node and the intercommunication activities between remote nodes. The current debate around such topics focuses on the definition of application programming interfaces (APIs) capable of clearly separating roles and responsibilities. The idea is to have client-side web applications (typically written in a mix of HTML and JavaScript) interact with web browsers through an ad hoc defined API allowing them to properly exploit and control browser functions, as well as interact with the browsers themselves in both a proactive (e.g., to query browser capabilities) and reactive (e.g., to receive browser-generated notifications) way. The mentioned application-browser API should hence provide a wide set of functions, like connection management (in a peer-to-peer fashion), encoding/decoding capabilities, negotiation, selection and control, media control, and firewall and NAT traversal.

The design of the application-browser API definitely represents a challenging issue, but it does not solve the overall problem at hand. The complete picture, in fact, envisages that a continuous real-time flow of data is streamed across the network in order to put into direct communication two (or even more) browsers at a time, with no further intermediaries along the path. We are hence talking about browser-to-browser communication, which is a revolutionary approach to web-based communication, since it allows peer-to-peer data communication to enter the web application arena for the very first time.

This is a major breakthrough in the telecommunications world and hence requires that a number of issues be taken into account, among which trust and security play a fundamental role. With respect to this last point, new security threats come to the fore as soon as we allow direct browser-to-browser communication. The basic web security policy currently in place is in fact based on the principle of isolation, allowing users to protect their computers from both malicious scripts and cross-site content references. As soon as we widen the scope to browser-to-browser communications, new facets of the general security issue are unveiled. First and foremost, communications security has to be taken into account in much the same way as it already is with other network protocols (e.g., SIP), allowing for direct communication between any two endpoints, unless we envisage the presence of relays acting as transparent intermediaries. Second, and also related to the first point, in case of direct interaction between any two browsers, proper mechanisms have to be put in place in order to verify consent before the actual data exchange phase starts. In the depicted scenario, consent verification must be enforced by the browser aiming to initiate communication with a potential peer. We also remark that real-time web-based communication requires that the browser has a strict interaction with the node on which it is installed (e.g., to access local audio and video devices before making a multimedia call with a target peer). This entails the definition of access policies involving some form of end user's consent.

The highly dynamic context we just sketched represents the current state of the art in the international research community. The issues identified, and even more the potential solutions mentioned, actually represent the current outcome of research and engineering work that is in full swing at the time of this writing, and hence susceptible to changes and rethinking in the near future as we gain more experience on the subject.

Real-time voice and video communication within browsers is among the topics currently gaining momentum in the two main Internet standardization bodies, namely the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).

RTCWEB Issues in the Presence of Middle Boxes

As explained in the previous section, the W3C/IETF joint work on the WebRTC/RTCWEB standardization efforts is targeting the ability for two or more users to interact with each other by just using their browsers, that is, with no need for any plugin or third party application of any sort. Both suites are currently conceived in order to be optimized for peer-to-peer scenarios, where application servers are responsible for signaling, security, and privacy enforcement, but not for the delivery of the negotiated streams themselves. Data transmission is assumed to happen directly between users whenever possible (i.e., if no relaying by a TURN server or any other similar intermediary is deemed necessary).

This does not prevent the possibility, though, that one or more of the involved parties is not an end user exploiting an actual browser. There are several scenarios, also documented in a Use Cases draft [2] in the RTCWEB Working Group, where a user may actually be interacting with an application rather than another user. This includes scenarios like connecting to an interactive voice response (IVR) system, a public switched telephone network (PSTN) gateway, or a conferencing server.

While this is not considered a problem per se (new applications may be conceived at the outset with embedded support for the RTCWEB framework/protocol suite), it may become an issue if any of those scenarios involves a legacy system, such as a system that is based on a different voice over IP (VoIP) protocol like SIP. In fact, although the RTCWEB suite is mostly based on pre-existing standard technologies, there are gaps that may need to be filled in on either side in order for an RTCWEB-compliant implementation to be able to properly interact with a legacy system. Specifically, an application server targeting such integration, and hence talking to both worlds, may need to take care of:
• The differences in signaling and/or media negotiation

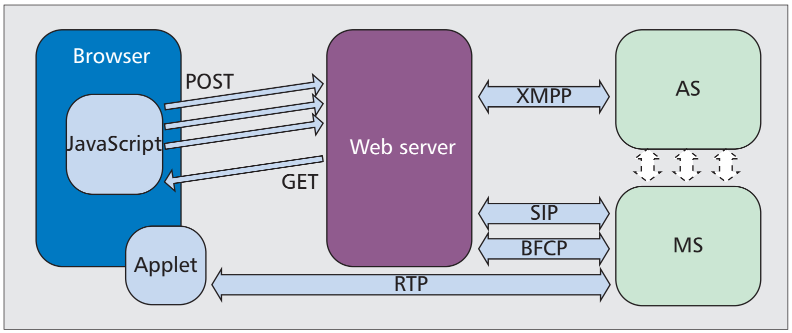

Figure 1. *The Meetecho WebLite "legacy" architecture.*

- Support for Secure Real-time Transport Protocol (SRTP) and interactive connectivity establishment (ICE), if missing
- Transcoding the media if there are no codecs in common

and more

Outstanding Issues

The first issue that needs to be faced is, of course, signaling. Considering that the WebRTC endpoint and the legacy system may be talking different languages, a signaling gateway may need to be put in place. This means that an RTCWEB Janus (a component able to seamlessly interact with both worlds while acting as a bridge between them) needs to be implemented. Fortunately, considering the above-mentioned reuse of standard technologies, this is only partially an issue. WebRTC and RTCWEB are currently based on JSEP [3] to handle signaling between web applications and application servers. It is important to notice, though, that JSEP is not a signaling protocol, but rather provides a way to handle the signaling/negotiation state in web applications, thus allowing actual signaling protocols (SIP, Jingle, etc.) to be deployed on top of it. Negotiation itself is instead based on the Session Description Protocol (SDP) offer/answer model. This means that if the legacy system makes use of SDP to negotiate media streams (which is the case in, e.g., SIP-based systems), taking care of bridging the two worlds is significantly easier. This also entails, however, that two different signaling state machines may be involved. Assuming the legacy system is based on SIP, a first offer from a web client would need to be translated into an INVITE toward a uniform resource identifier (URI), further updates from the web client would need to be mapped onto re-INVITEs (and vice versa), closing the session on either side would need to be taken care of accordingly, and so on.

This becomes more complicated whenever either side makes use of specific functionality at the media level that the other end does not support. For instance, as specified in the RTP Usage draft [4] in the RTCWEB WG, WebRTC endpoints are designed to support innovative features like SRTP, the Audio-Visual Profile Feedback (AVPF) profile for RTCP, RTP/RTCP multiplexing, RTP session multiplexing, and so on. If the legacy system does not implement all those features, but only, for instance, plain RTP, a proper RTP gateway may need to be deployed as well. In fact, while some of the negotiated features may just be declined (e.g., RTP/RTCP multiplexing), some others must necessarily be implemented. This is the case, for instance, of SRTP: consensus was reached in the IETF to explicitly forbid, for security reasons, the use of plain RTP in the WebRTC suite. This means that any entity willing to interact with WebRTC endpoints must necessarily support SRTP as well. Systems supporting just plain RTP would hence need to rely on a media gateway in order to fulfill this requirement. The deployment of the mentioned media gateway would be under the responsibility of the RTCWEB Janus introduced before, considering the media negotiation could need to be manipulated in order to place the media gateway along the RTP path between the WebRTC endpoint and the legacy system.

Another feature WebRTC endpoints require as mandatory is ICE support. ICE is the standard suite of technologies adopted by most existing VoIP deployments to take care of Network Address Translator (NAT) traversal. This suite includes ways for two peers to gather candidate addresses where they may be reached (either directly or through relays), negotiate such addresses, and check their reachability by means of the Session Traversal Utilities for NAT (STUN) protocol. While ICE is now fairly widely deployed, there are still several legacy systems that do not implement it. Besides, systems implementing it may be supporting different versions or flavors of ICE (e.g., in terms of how STUN connectivity checks are constructed), meaning that even in this case there may be interoperability concerns. Considering ICE is strictly related to the media channels being negotiated and used, the above mentioned media gateway may need to fill that gap as well, in case the legacy system is not 100 percent compliant with what the WebRTC endpoint is expecting.

Once signaling, negotiation, and RTP/ICE issues are dealt with, success in a WebRTC-legacy endpoint interaction still cannot be guaranteed. In fact, it is quite obvious that a WebRTC endpoint and a legacy one need to share support for at least a codec in common if they want to be able to talk to each other. A further common codec needs to be supported if the two endpoints also want to add video to the communication. Failure to find a codec supported by both endpoints would force the RTCWEB gateway to also provide transcoding functionality. Transcoding is usually a choice gateway implementors try to avoid; in fact, it is a CPU-intensive feature, which may affect the number of interactions the gateway may provide at the same time. Besides, it adds latency to the interaction, considering an intermediate component bridging the media and adapting them to both parties is deployed. In order to minimize the chances that transcoding is needed, the RTCWEB WG has so far reached consensus on G.711 as a mandatory-to-implement (MTI) codec for all WebRTC endpoints. G.711 is the most widely deployed codec in VoIP endpoints and systems, and as such it can safely be considered the codec to fall back on if one wants to avoid transcoding. The RTCWEB Working Group (WG) has also reached consensus on the adoption of Opus as an MTI higher-quality audio codec. Consensus on a video codec, instead, has not been reached as of yet, and a heated debate is still taking place.

The entire set of above described requirements has proven useful when trying to design a hybrid architecture for seamless interaction between WebRTC endpoints and our legacy standards-based conferencing platform, as explained in the next section.

Bringing Research to the Field: Meetecho RTCWebLite

The analysis made in the previous section motivated us into designing a WebRTC point of entry to our conferencing platform called Meetecho [5]. Meetecho is a standards-based conferencing architecture that makes use of standard protocols (e.g.,Extensible Messaging and Presence Protocol [XMPP], SIP, and Binary Floor Control Protocol [BFCP], among others) to provide collaboration features. As such, especially for its use of SIP/SDP and RTP to provide audio and video, it can be seen as the "legacy" system with which a WebRTC endpoint might want to interact. For what concerns Meetecho and browsers, we already devised a fully web-based interface to the functionality provided, called WebLite. Meetecho WebLite allows participants to access a conference room from their browser. This means that access to all of the above mentioned standard protocols and functionality is provided by means of a gateway allowing users to interact with native clients in a transparent fashion. While this was relatively simple to accomplish with just HTML and JavaScript for most of the envisaged features, for what concerns audio and video we so far had to rely on a different approach: a Java Applet that, guided by the application logic in JavaScript, takes care of the user devices (microphone, webcam), as well as of encoding and decoding functionality and management of the RTP streams exchanged with the conferencing server, as illustrated in Fig. 1.

This was a necessary step since, as explained in the first sections, browsers currently have no means to provide a native way to access audio/video capturing functionality and bidirectional streaming if not by means of plugins like Java Applets, Flash/Silverlight applications, or specifically designed Netscape Plugin API (NPAPI) plugins. This is, of course, what the RTCWEB/WebRTC joint work is trying to provide a solution for, and as such this offered us an excellent opportunity to try to make available a completely web-based interface to our "legacy" conferencing system, as depicted in Fig. 2.

In order to validate our interoperability efforts, we exploited Chrome Canary as the WebRTC endpoint, being this the only browser that at the time provided a ready-to-use, almost complete WebRTC stack.

Following the guidelines presented in the previous section, the first step was, of course, to take care of signaling. As anticipated, Meetecho makes use of SIP as its signaling protocol to negotiate media streams. SIP is also used to set

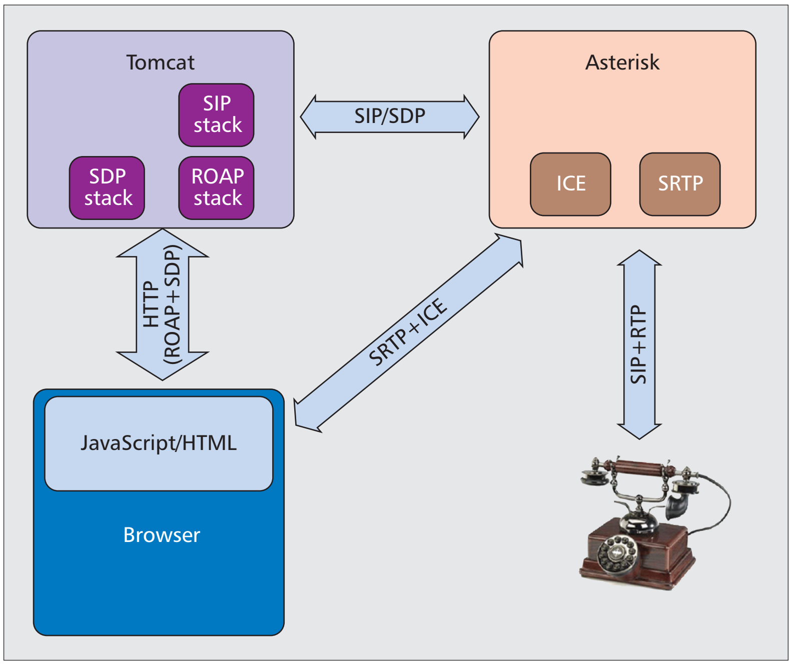

Figure 2. *The Meetecho RTCWebLite archite.*

up a BFCP channel with participants, in order to provide floor control functionality. We chose to make use of RTCWeb Offer/Answer Protocol (ROAP) [6], a JSON-based signaling protocol, as a simple and quick solution to provide signaling in the web interface. ROAP was a first proposal that was made in the RTCWEB WG to perform signaling in WebRTC, but was then put aside in favor of JSEP, which, as explained in the previous section, allows for more generic management of the signaling state rather than signaling functionality per se. As such, JSEP can be used as a standing ground to implement any kind of signaling, including ROAP. ROAP-over-JSEP looked like a simple and effective solution, and for the purpose we made use of a simple JavaScript library, developed by Google as an example of how JSEP could be used to manage signaling on the client side. We then provided a simple user interface (UI) to let users decide whether to negotiate both audio and video or just audio, in order to trigger the signaling accordingly. For the server side, we implemented the signaling gateway as a simple web application taking care of signaling messages originated by the client, and in turn generating SIP INVITEs to our conferencing system. The signaling state is then handled accordingly, for example, by providing ROAP answers when a 200 OK to the INVITE is received or hanging up either side of the call when the other party closes the session. The task accomplished by the gateway is not limited to only signaling, of course; as explained in the previous section, there may be several features the WebRTC endpoint may support and the legacy system may not. This was what we found as well in a few aspects. The first feature we were lacking was SRTP support. As anticipated, this is an absolutely mandatory feature in WebRTC. In order to fix this requirement, we moved from Asterisk 1.4, the PBX implementation on which our

It is important to notice that while Asterisk provides audio mixing natively, it does nothing in that sense when it comes to video, which can only be handled with a passthrough approach. To overcome this issue, we have designed and implemented a video mixer that can take care of both transcoding and mixing heterogeneous video streams.

audio/video features were built, to Asterisk 1.8, which provided us with native SRTP support. Another mandatory feature we found ourselves lacking was ICE. This was not an issue we could solve by just upgrading Asterisk (work on this is already ongoing in Asterisk 11, which will also provide native WebRTC support, but is not available at the time of this writing); thus, we implemented support for it ourselves. We did so in two different parts of our system. The negotiation of candidates was taken care of in the above introduced signaling gateway; in fact, considering our conferencing server is always reachable at a public address, the gathering of candidates is definitely simplified and allows the gateway to act as an ICE-Lite peer. The connectivity checks were instead implemented on the Asterisk side, by modifying the pre-existing STUN protocol implementation in order to take care of all the required additional attributes like USE-CANDIDATE, XOR-MAPPED-ADDRESS, MESSAGE-INTEGRITY, and so on, and be able to both respond to connectivity checks and generate checks on its own. This choice was made in order to keep things as simple as possible for signaling, and focus on the media challenges instead, while also keeping signaling- and media-related issues well separated and independently addressable.

At this point, we started looking at possible issues in the negotiation process itself. We first of all noticed that Asterisk would reject the "RTP/SAVPF" profile as negotiated by the WebRTC endpoint: this is caused by the fact that, while Asterisk 1.8 supports RTP/SAVP, it does not currently support the extensions for RTCP feedback. Hence, we looked, after rewriting the SDP descriptions, in both directions, in order to make sure that Asterisk finds RTP/SAVP in the SDP received from the gateway, and WebRTC gets back the RTP/SAVPF it negotiated in the first place instead. When done with that, we looked after taking care of SDP attributes that might confuse either side. As anticipated in the previous section, WebRTC endpoints try to multiplex RTP and RTCP streams, which is something Asterisk does not support at the moment. Specifically, in accordance with RFC 4566, Asterisk assumes that the port used for RTCP is equal to the RTP port, incremented by one. In order to make sure the WebRTC endpoint was made aware of this, we let the gateway properly modify the conferencing server's SDP by introducing a new media-level attribute of type "a=rtcp:..." for each involved medium, with explicit indication of the port to be used for RTCP, before sending it to the WebRTC endpoint. Moving further, as anticipated we chose to delegate ICE-related negotiation to the gateway rather than to the private branch exchange (PBX) itself. To this purpose, we made sure that before providing the WebRTC endpoint with any reply or update from the conferencing server, the SDP would be extended with the ICE additional attributes: a session-level attribute to report an ICE-Lite implementation (a=ice-lite), and media level attributes to convey authentication information (a=ice-ufrag and ice-pwd) and candidates (a=candidate). Of course, this obliged us to provide, for each negotiated medium, two different candidates instead of just one. In fact, as explained before, we negotiate RTCP explicitly on a different port, which is to be accordingly reported as a candidate, since connectivity checks have to be performed for both RTP and RTCP.

When done with the above mentioned steps, we started looking into interoperability at the encoding level. For audio, this was quite easy. In fact, we found out that WebRTC endpoints and our conferencing server had a codec in common, specifically G.711 m-law. Even though G.711 is definitely not the best choice when it comes to audio, it greatly simplified our task of creating a bridge between WebRTC and our legacy system. This said, we recently started working on the integration of an Opus codec in Asterisk. Opus is a further MTI codec in WebRTC and provides much better audio quality, with affordable network bandwidth. As explained later, this task is not complete at the time of this writing and will be the subject of future work.

Concerning video, the work to be done was quite different. In fact, it is important to notice that while Asterisk provides audio mixing natively, it does nothing in that sense when it comes to video, which can only be handled with a passthrough approach. To overcome this issue, we have designed and implemented a video mixer that can take care of both transcoding and mixing heterogeneous video streams. The mentioned video mixer, though, was able to support "legacy" encodings like H.261, H.263 (with related extensions), and H.264, but not VP8, which, waiting for a clear consensus on the MTI video codec in WebRTC, is the only video codec our reference WebRTC endpoint supported. As a consequence, we had to accomplish two different tasks:

- Implement a simple pass-through mechanism for VP8 frames in Asterisk
- Implement full VP8 transcoding features in our video mixer, in order to allow VP8-compliant endpoints to take advantage of the desired video composition capabilities

This process was almost successful, with just a minor drawback. In fact, our video mixer currently only supports QCIF and CIF resolutions with respect to video, while the reference WebRTC endpoint always sends 640 × 480 video streams. While this is not a problem on the client side (the browser showed no issue when presented with a lower-resolution incoming video stream), it can be problematic for the video mixer, as it has to continuously downsize incoming frames from WebRTC endpoints in order to mix them. This issue will be solved as soon as WebRTC endpoints implement the "constraints" mechanism recently introduced in the WebRTC specification. To complete the integration, we also implemented support for the RTCP FIR feedback message in Asterisk; this was needed in order to allow the video mixer to receive a full frame for a participant whenever the participant was to be included in the mix (e.g., when the participant was granted the video floor by means of BFCP), in order to avoid annoying artifact or ghosting effects.

Related Work

WebRTC is gaining more and more interest, and a number of relevant projects are currently under development. A large community of developers exchanges information, comments, suggestions, and announcements on the discuss-webrtc group on Google Groups.

Doubango Telecom has recently started the so-called sipml5 project, which shares some similarities with our work. Sipml5 is based on an open source HTML5 SIP client, entirely written in JavaScript and relying on WebRTC. It was conceived at the outset with the aim of fostering interoperability with SIP networks. The reference architecture envisages the presence of an Asterisk server playing the same role as our Janus component. The main difference between the two concerns the chosen signaling protocol: we decided to rely on the lightweight ROAP over JSEP approach, while sipml5 makes use of the SIP protocol. It is worth noting that sipml5 developers had to face the same ICE-related issues as those we found for Asterisk.

A number of solutions leveraging the use of a multipoint control unit (MCU) have been developed and advertised on the aforementioned discuss-webrtc group. Among them, we mention *Lynckia*, an MCU-based approach to WebRTC videoconferencing currently under development at the Universidad Politecnica de Madrid. Lynckia leverages a single PeerConnection in the broadcaster, connected to an external agent (MCU) that publishes the media. A simple web application connects the subscribers to the MCU, which forwards the stream. Other similar projects exist, all leveraging the use of an MCU. The approach fostered by Lynckia and other MCU-based solutions is significantly different from ours. First, it is not aimed at interaction between WebRTC browsers and the "legacy" SIP network. Furthermore, it just provides audio/video capabilities, while we focus on providing a whole set of collaboration tools, with audio and video being just two of them. This also applies to sipml5.

Finally, we mention a project currently ongoing at the Real-Time Communication Laboratory of the Illinois Institute of Technology, called Voice and Video on the Web (VVoW), which aims at building a web-based conferencing service. In its first version, developed in 2011, the solution was Flash-based; recently, the system has been redesigned to be WebRTC-compliant and to use HTML5.

https://sites.google.com/site/vvowproject, no longer exist

Conclusions and Directions of Future Work

In this article we propose an engineering approach to the integration between legacy SIP-based systems and WebRTC applications. We discuss the main issues entailed by the mentioned process and present a real-world example associated with allowing access to the Meetecho collaboration framework from WebRTC browsers.

We are continually updating our implementation with respect to changes occurring on a daily basis, such as how ICE is now handled in Chrome, in contrast with the less standard version it relied on before. We are also working on the support of different WebRTC endpoint implementations, like Firefox Nightly, as well as browsers exploiting the webrtc4all (http://code.google.com/p/webrtc4all/) plugin to implement the required functionality before it is natively supported and widespread. Besides, as anticipated earlier, we are also devoting efforts to the integration of the Opus codec in Asterisk in order to have it available in the list of supported codecs when negotiating a media session. While this integration is nearly completed, it still needs extensive testing, and is so far only supported by Firefox Nightly.

We are also devoting efforts to the integration of the Opus codec in Asterisk, in order to have it available in the list of supported codecs when negotiating a media session. While this integration is nearly completed, it still needs extensive testing, and is so far only supported by Firefox Nightly.

References

[1] C. Holmberg, S. Hakansson, and G. Eriksson, "Web Real-Time Communication Use-Cases and Requirements," draft-ietf-rtcweb-use-cases-and-requirements-10, Dec. 2012.

[2] C. Perkins, M. Westerlund, and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP," draft-ietf-rtcweb-rtp-usage-05, Oct. 2012.

[3] C. Jennings *et al.*, "RTCWeb Offer/Answer Protocol (ROAP)," draft-jennings-rtcweb-signaling-01, Oct. 2011.

[4] J. Uberti and C. Jennings, "Javascript Session Establishment Protocol," draft-ietf-rtcweb-jsep-02, Oct. 2012.

[5] A. Amirante *et al.*, "Standard Multimedia Conferencing in the Wild: the Meetecho Architecture," *Multimedia Tools and Applications*, 2011, pp. 1–18.

[6] S. Loreto and S. P. Romano, "Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts," *IEEE Internet Computing*, vol. 16, no. 5, Sept.–Oct. 2012, pp. 68–73.

These 4 are all internet-drafts.

Biographies

Alessandro Amirante (alex@meetecho.com) received both his M.Sc. degree in telecommunications engineering in 2007 and his Ph.D. in computer engineering and systems in 2010 from the University of Napoli "Federico II," Italy. He is currently CTO at Meetecho s.r.l. and a senior researcher at the Computer Science Department of University of Napoli "Federico II." His research interests fall in the field of networking, with special regard to next generation network architectures and multimedia services over the Internet.

Tobia Castaldi (tobia.castaldi@unina.it) received his degree in telecommunications engineering from the University of Napoli "Federico II" in 2006. He is currently CEO at Meetecho s.r.l. and a senior researcher at the Computer Science Department of the University of Napoli "Federico II." The main topic of his research concerns real-time applications for the next-generation Internet with special regard to the IP multimedia subsystem (IMS) architecture and services.

Lorenzo Miniero (lorenzo.miniero@unina.it) received his degree in computer engineering from the University of Napoli "Federico II" in 2006. He is currently COB at Meetecho s.r.l. and a senior researcher at the Computer Science Department of the same university. His research interests mostly focus on next generation networks, network real-time applications, and communication protocols, with special emphasis on the related standardization efforts.

Simon Pietro Romano (spromano@unina.it) is an assistant professor in the Computer Engineering Department at the University of Napoli and cofounder of Meetecho, a start-up (and spin-off of the university) focusing on Internet-based conferencing and collaboration. His research interests primarily fall in the field of networking, with special regard to real-time multimedia applications, network security, and autonomic network management. Romano has a Ph.D. in computer networks from the University of Napoli. He actively participates in IETF standardization activities in the real-time applications and infrastructure areas.