IMS Centric Communication Supporting WebRTC Endpoints
Bruno Simões Cruz, João Paulo Barraca
Institute of Telecommunications, Aveiro, Portugal
University of Aveiro, Aveiro, Portugal
{bruno.cruz, jpbarraca@ua.pt}

Abstract - With the evolution of technologies such as WebRTC and telecommunication (Telco) architectures aligned with the latest 3GPP standards, the search for interoperability between traditional and web-oriented endpoints is vital, with new use-cases and multimedia services to be explored. Currently, convergence goes beyond the standards and, as such, its search and the technological obstacles motivate this work. We make a proposal for the implementation of a platform uniting state-of-the-art Telco networks with WebRTC technologies. To validate the solution, an IMS integrated prototype was built, being evaluated in terms of call throughput and system induced mouth-to-ear delay. Results show that call delay is on par with current mobile network calls, making the current solution a viable alternative for communication bridging in a Telco network.

Keywords – WebRTC, IMS, API, telephony evolution, OneAPI

I. INTRODUCTION

With the increase of available services exposing web-oriented telecommunication capabilities, the basis for creating applications that come to live on the web is also increased, giving light and enabling the research of brand new use-cases, previously un-touched. Now, more than ever, creating applications that enable, for instance, sending textual messages via a simple application programming interface (API) request is possible, with commercial services such as Twilio and Plivo providing these capabilities. With a myriad of features exposed, these services allow one to place calls between numbers, exchange either textual or multimedia messages, enter conference calls with multiple users, record the calls themselves, among many other characteristics. However, these are over-the-top (OTT) web services, not integrated with traditional telecom networks and, in consequence, creating a more limited reach in terms of possible use-cases. Thus, the OneAPI initiative from Groupe Speciale Mobile Association (GSMA) has emerged as a possible solution for this obstacle, aiming to standardize an operator-focused API with influences from major operators worldwide. The OneAPI specification defines features such as third party call control, messaging, billing, location, and capability discovery, creating the grounds for a fully standardized, web enabled, telecom API. With a service based on this model, an operator is able to give willing developers the same features enabled by the likes of Twilio, but with a solution that is completely integrated with their existing networks, and completed aligned with 3GPP evolution plans.

Alongside the evolution in these web services comes WebRTC [1], an API definition released by Google in Q2 2011 for browser-to-browser communication, that as seen great market acceptance and spawned to date two JavaScript implementations – one from Google (maintaining the same name) and another from Ericsson (called OpenWebRTC) [2]. With communication advances involving WebRTC, one is able to create full multimedia experiences in one’s browser, with this technology becoming increasingly popular in this realm, even though it is still an open standard and only recently defined the mandatory video codecs to use. Most importantly, browsers can execute web applications that mimic native behavior, effectively enabling generic WebRTC enabled communication solutions. The ability to enable first party call controls (to place and answer audiovisual calls, send multimedia messages and exchange data between users) is as simple as creating corresponding JavaScript commands, leading to believe that the stage is set for an evolution in what were traditionally Flash-based RTC platforms, which are still used today although with decreasing market share.

With the 3rd Generation Partnership Project’s (3GPP) IP Multimedia Subsystem (IMS) architecture, the bond between the traditional network and the Internet is strengthened, bridging the worlds to provide network exposed services to clients, and an architecture motivated by this purpose, aiming to provide the means for operators to retain and regain clients by offering experiences similar to those found on the Internet. In this article, a solution for the integration between IMS and WebRTC technologies is proposed, discussing design and implementation of a prototype, and its evaluation in relation to the performance of traditional GSM and WCDMA voice calls.

Having the proper context been made, section II covers related work over these topics and section III details the features that are planned to be exposed by the prototype. In section IV, an architectural design and implementation is proposed, with the solution evaluated and captured results discussed in V. In section VI, a summary on the results and future work is given.

II. RELATED WORK

There are still many obstacles to the widespread usage of WebRTC in browser enabled communications, let alone integration with telecom networks, in both enterprise and consumer-grade products.
How to guarantee security over call eavesdropping or identity impersonation, how to do identity management across different domains, or how to maintain general quality of service and billing procedures for calls are all topics currently in discussion. These non-standardized security [3-4] and identity management [5] requirements, technical considerations and architectural designs [6] have been widely discussed by the academic community, giving an insight into the state of affairs of the technology, how it has been embraced and what is to expected of the standard itself.

In this work, an architecture proposal is given on how to integrate WebRTC-enabled endpoints with IMS networks, allowing for calls between users with this browser technology and legacy VoIP clients. Similar proposals have already been discussed in the community, with a relevant approximation described by Adham Zeidan et al [7], in a work defining a VoIP conference server for Session Initiation Protocol (SIP) based systems. Essentially, the vision described by the authors was the creation of a conference service with multiple User Experience (UX) driven use-cases (such as video and document sharing, or virtual whiteboard), all backed by SIP signalling and WebRTC as the technology providing RTC capabilities. Using a Kamailio server [8] as a registrar for endpoints, which takes the responsibility for locating users across the domain, and implementing its own WebRTC client, conference manager and audiovisual supporting servers, this work provides a clear view on how different systems are able to integrate and form a powerful solution, defining a rich visual experience on these multiparty calls. Nonetheless, the work does not provide a view on how media was affected by the different handlers, and/or if call quality remained on par with user expectations, which we address in our work.

With the WebRTC API, and technology itself, still a work in progress, there is already work involved on perfecting the standard towards easing development effort. ASPINT [9] is a proposal for a wrapper over the WebRTC API, focusing on abstracting technical details to create a more approachable API for developers. With this wrapper on the client-side, but with added communication capabilities using SIP as the signalling protocol (which is not defined on the WebRTC standard), creation of IMS-integrating services is eased. Functionally able to create scenarios such as Skype, using what is essentially a client-side provided SDK, the solution proposed in this work goes beyond what ASPINT aimed for. Here, both SDK and gateway to where clients connect are delivered, being completely integrated with 3GPP standards, and offering a proposal for an end-to-end architecture and relevant testing that solidifies the proposal as a whole. Additionally, ASPINT serves the clients and, thus, is responsible for first-party call control, but is unable to give developers the means to expand use-cases towards third-party control capabilities, which is also a focus on this work, especially with the usage of GSMA’s OneAPI specifications.

3GPP studies are also being made, proposing solutions for integrating these worlds. TR 23.701 [10] gives several solutions towards this objective, exposing architectural recommendations, components needed and their location and responsibilities. The solution we propose follows a similar approach to Solution 3 of TR 23.701, with the advantage that no other components are necessary on the network, with the gateway treating the media, handling transcoding and other protocols to ensure communication between the different endpoints.

III. SOLUTION REQUIREMENTS

The context given and its investigation allow the shaping of our solution and its roadmap. In order to implement an interesting prototype, one must choose which features should be exposed and use-cases to tackle. One may consider scenarios such as CRM plug-ins for browser calling on a client information check-up by a vendor. Such enterprise-aimed application integration would result in an option to trigger a call to a given client from a vendor’s marketing tool. This is a simple webpage button that could introduce a layer of context to the interaction and enable a whole new set of use-cases.

To enable the referred use-case, the solution should implement GSMA’s OneAPI – namely, the third party call control [11] and messaging [12-13] standards, integrating the solution with an existing IMS infrastructure. Third-party call control will allow developers to trigger calls between parties and end them, in addition to holding and resuming, adding participants to a call and retrieving information regarding a call session and its participants. Additionally, textual and multimedia messaging will be exposed by the API and its reception enabled in the WebRTC client, as well as non-standardized authentication mechanisms and the retrieval of notifications for call states. These are necessary to establish proper control, context and security. On the other hand, the WebRTC client will be responsible for all the first party call controls, enabling the reception of IMS signalling, portrayed by SIP requests. This will enable the client to answer and reject incoming calls, in addition to providing the messaging exchanges referred and all processing needs. Moreover, since WebRTC clients may not support the media codecs negotiated with traditional Voice over IP (VoIP) clients, transcoding mechanisms may need to be implemented in order to convert the outgoing multimedia content. This will increase processing needs and may affect call quality, but is a necessity to enable communication between both parties.

In addition to these functional requirements, other general constraints must be ensured. In terms of performance, call setup times must remain minimal: from the time a caller starts the call to the time the callers hears the ringing on the line, no more than one second should pass, processing-wise. Moreover, concurrent calls must be supported, aiming for a baseline of 100 calls in parallel.
In summary, a minimal solution must at least provide the follow features:

- Third-party call control and message sending functionalities enabled by an OneAPI compliant REST interface. This API must be exposed by an application server (AS), able to be integrated in an IMS network core, translating HTTP requests to SIP signalling and handling all the necessary information forwarding and processing;
- WebRTC client-supporting gateway that can be integrated in an IMS network, translating JavaScript actions to SIP signalling requests and responsible for all first-party call control needs;
- A solution provided Software Development Kit (SDK) to allow external developers to create their own WebRTC clients and integrate them with the solution’s gateway;
- Server and gateway implementation, as well as its design, should strive to enable parallel call processing able to maintain approximately 100 on-going calls, on a single server, while keeping call setup time to a minimum.

IV. PROPOSED ARCHITECTURE

Taking to account the requirements discussed, we propose an architecture, defining the platform in terms of component separation and interaction, showing how the different components will be connected in the IMS network – as represented in Figure 1.

Figure 1 - Proposed architecture showing WebRTC and API components

As depicted, the solution is composed of three different components - the AS exposing a REST API, an operator-side WebRTC gateway and WebRTC client. The API is responsible for translating incoming HTTP requests to SIP requests, used for interacting with the IMS core. For instance, when an external client contacts the endpoint responsible for triggering call sessions, the AS will parse its JSON content for the contact information and send the expected SIP INVITE request to the network, using SIP Servlets (a Java specification for creating SIP applications). This SIP request will be forwarded to each parties’ equipment, with the server acting as a proxy for the remaining signalling generated. On the other hand, the gateway is responsible for receiving and processing requests coming from an available proprietary SDK, used in third-party applications. When these applications connect to the gateway, this component will take care of network registration, session management and forwarding of communications, media and notifications. This facilitates development effort, by creating an abstraction layer over the WebRTC API and providing additional support for integration with the gateway. Both AS and WebRTC gateway were connected to the network’s CSCF’s (Call Session Control Function), the main signalling forwarding and handling nodes.

The P-CSCF’s (Proxy-CSCF) Gm interface was used for the gateway, routing client generated actions into SIP requests to the network, effectively mimicking a normal VoIP client and enabling users to register as these devices. Essentially, what happens at this stage is all in all similar to what would happen in a VoIP client: when trying to register, the user’s client would send a SIP REGISTER request to the network, which would be forwarded by the internal CSCFs up to a point where the credentials would be checked in a central database (Home Subscriber Server or HSS), returning a query to the client, that will be responded to complete the authentication. When these authentication steps are complete, the client is effectively registered in the network, receiving notifications of incoming requests, among many others. With the WebRTC client and gateway, this scenario is mapped but with the difference that browser actions are translated into JavaScript interactions with the gateway, with it translating the necessary actions to SIP requests. The gateway then handles which requests, coming from the network, are routed to each WebRTC user, making notifications available to the user’s browser and, effectively making it another mobile device on the network.

The S-CSCF’s (Serving-CSCF) ISC interface was used to notify the AS’s of incoming SIP messages and to route SIP requests into the network. Alike the Gm interface, it supports the SIP protocol and is designed to provide communication between the IMS core and deployed application servers. In order to implement the conference call features, the AS itself is designed to provide media relay, creating media bridges to which clients get connected upon placing the calls, where media is exchanged between call parties. Although this increases overall complexity, it provides greater control over call management and notification forwarding. A third party application, one that uses the solution’s API, can make use of two components, one which translates user actions to API requests (implemented by developers using the solution) and one which uses the given client-side WebRTC SDK to handle all the necessary processing from the interface and establishing a bridge between them and the gateway.

There are alternatives to building such a solution, such as relying on the AS to interact with WebRTC clients or using a network component called Session Boarder Controller (SBC) to replace the job of the gateway through natively supported WebRTC components. However, the suggested approach minimizes software complexity and session management issues (which would happen if using the AS as the WebRTC signalling and media relay), while decreasing network dependencies that would come from using an external SBC.
V. EVALUATION AND RESULTS

Since media is relayed by both gateway and AS, these components interfere in both media and signalling paths, making performance testing more important to measure how these design decisions come to affect call quality.

We evaluate our solution through two types of performance tests: load-tests to analyze how many concurrent calls the system is able to handle, and mouth-to-ear delay tests to see how much time it takes for voice to go from one party to the other. For both types of tests, the codec in use by all parties was G.711. Several scenarios were devised, each with increasing complexity in terms of server-side components used and resulting interactions between solution components.

At first, only peer-to-peer calls were established, capturing a baseline value for concurrent calls supported between traditional VoIP clients, registered in the IMS core (without enabling media relaying). Secondly, calls were made with the media relayed by a media proxy. This component mediates the exchange of media between parties, an approach most similar to what would be the AS’s behavior in the network – media rooms are created in this server, to where parties are connected and media is exchanged. Thirdly, only calls created via the AS (connected to the IMS environment) were established, revealing how the AS behaves under load and how it is affected by the decision of managing media sharing. Fourthly, calls were established between WebRTC clients, using only the gateway connected to the IMS core. This mostly tests the gateway itself, since IMS will not relay media (which is kept in the gateway and forwarded to the parties involved), with the gateway only interacting with IMS for signalling exchanges. Finally, an end-to-end scenario was assembled and calls were established between WebRTC and VoIP clients, with media being relayed by the AS and an API client continuously creating call sessions.

With these considerations, one is able to compare the performance at different times and setups, promptly revealing solution bottlenecks and possible design obstacles.

A. Load-testing

A performance tool called Multi-protocol Test Suite (MTS) was used to create the test-sets, focusing on number of concurrent calls supported by the solution. Tests followed very similar approaches, with an MTS script responsible for creating multiple calls and another for receiving and answering these calls. Each MTS script simulates an endpoint on the network, capable of communicating in a manner similar to any other VoIP client. A “caller” script sends a SIP INVITE to the network, inviting a “callee” script to an audio call. In turn, the callee acknowledges and processes the invite, accepting it after 3 seconds (mimicking user delay on call answering), and sending randomly generated media (a G.711 encoded stream) for 30 seconds.

On the VoIP-to-VoIP scenario with the AS deployed, a script would generate multiple HTTP requests per second to the API, creating calls between MTS-simulated users, which would process the calls automatically. Similarly, focusing on an end-to-end test where both WebRTC/VoIP clients and gateway/AS were engaged, the same script (“MTS API Loader” on Figure 2) would be used to create calls between MTS users (“MTS Responder” on Figure 2) and custom WebRTC clients. This client was designed to answer incoming calls automatically and transmit audio captured from the microphone along the duration of the call. In short, media would flow between MTS endpoints and WebRTC clients, going through the AS and gateway, testing all the components.

Across all test cases, the solution performed above the testing tool’s capacity, with MTS remaining a bottleneck over time, being unable to handle all the signalling and media exchange necessary to establish calls above a certain level. As such, when it comes to the end-to-end scenario, call throughput level is considered the same as AS and RTP proxy calls, values that can be seen in Table 1, expressed in total number of concurrent calls. In this table, WebRTC-to-WebRTC calls are not contemplated, with the reasoning behind the decision being that media is relayed by the gateway. This means that more calls would only increase gateway load, which, according to previous and external testing, is able to handle a substantially greater number of concurrent calls than those tested here.

<table>
<thead>
<tr>
<th>Test</th>
<th>Concurrent Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-peer calls without media</td>
<td>330</td>
</tr>
<tr>
<td>RTPproxy calls</td>
<td>99</td>
</tr>
<tr>
<td>AS calls</td>
<td>99</td>
</tr>
<tr>
<td>WebRTC/AS calls</td>
<td>99</td>
</tr>
</tbody>
</table>

The hardware used during this testing phase was an Intel Pentium dual-core (clocked at 1.46 Ghz), with 3 GB of memory, running Ubuntu 12.04 LTS (32 bit) and with a 1Gb Ethernet connection. As such, it is considered that performance levels can be higher when using higher end...
equipment, since server memory and processing were stable across the tests. This is, in fact, an indication that the requirements for the implementation of this solution are low, validating the adoption of WebRTC based communication solutions.

B. Mouth-to-ear delay

More than just concurrent call performance, since the AS and gateway remain in the media exchange path, one may envision added delay in audio transmission, making mouth-to-ear testing all the more important. The tests analyzed in this topic aim to measure how much media transmission latency is affected by the solution, with an approach structured as represented in Figure 3. The methodology follows previous work made by Agastya et al [14], focusing on measuring the real world latency of several VoIP communication suites.

![Figure 3 - Mouth-to-ear delay test configuration](image)

We rely on Audacity, running on the server, as the tool used as sound generator and recorder in these tests, with the approach set as follows: the server’s sound-out port is connected to the caller’s microphone and the callee’s sound-out to the server’s microphone. As such, when Audacity plays any sound, it will be forwarded to the caller’s microphone and through the ongoing call, eventually reaching the callee, which forwards the sound back to the server’s microphone. To this end, after the test calls were established, Audacity plays a pre-configured track composed of several ping sounds evenly spaced over the course of 30 seconds (a Click Track), recording the resulting audio stream coming from the callee. This test configuration would allow the easy capture and comparison between the time the ping sounds were sent through the connection and the times they were received, with their average being noted and analyzed. The results also exclude the acquisition delay from the hardware and software, which as measured to be approximately 25 ms. We consider the time difference between the two signals (originally sent and received by client), and measure the reproduction delay through a loop test.

Similarly to the load-testing scenarios done previously, multiple test-cases were devised. Several singular peer-to-peer calls were first established and captured, having their recordings been analyzed. This reveals the default voice delay in communications when VoIP clients establish a call with each other, with only the IMS assuring signalling and no media interference from network components. Next, singular VoIP calls were made with media going through the RTPProxy and, afterwards, between WebRTC clients. Finally, with calls being triggered by the AS between a VoIP and a WebRTC client, the end-to-end use-case was tested.

The results gathered for this last set of tests, with the complete solution in analysis, show that total delay is set at 265 ms, with the reference peer-to-peer VoIP client tests showing a delay of 225 ms by itself, as can be seen in Table 2. This delay is quite significant and can be justified with network conditions, codec delays or browser handling of JavaScript, but in no way related to the implemented solution since media was not handled by the AS or gateway on the first scenario. Entities such as ITU have created recommendations for these scenarios, guidelines for levels of acceptable call quality, that should not be overlooked when designing such a system: according to ITU’s recommendations on one-way audio delay [15], above a delay of 250 ms, communication is placed at a “Some users dissatisfied” level, predicting problems in the calls that will ultimately worsen the experience for the parties involved.

<table>
<thead>
<tr>
<th>Test</th>
<th>Average delay (ms)</th>
<th>Total difference (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-peer calls</td>
<td>225</td>
<td>-</td>
</tr>
<tr>
<td>RTPProxy calls</td>
<td>230</td>
<td>5</td>
</tr>
<tr>
<td>AS calls</td>
<td>235</td>
<td>10</td>
</tr>
<tr>
<td>WebRTC calls</td>
<td>255</td>
<td>30</td>
</tr>
<tr>
<td>WebRTC/AS calls</td>
<td>265</td>
<td>40</td>
</tr>
</tbody>
</table>

A worthy note for comparison can be made between the delay felt in this more controlled scenario, and the one from normal mobile network calls. The previous approach was replicated in a scenario using commonly available cell phones and through commercial GSM or UMTS networks. We followed the same script, with Audacity playing a Click Track through the computers speakers, which was captured by the callee’s microphone and sent to the callee. The callee, on the other hand, would emit the sound received to its loudspeakers. This created an echo in the recording, which was taken into account when conducting the experiment and calculating the results, while discarding acquisition latency from the measuring system. In these tests, the acquisition delay is promptly discarded because Audacity will record both the sound first emitted and the one finally emitted by the callee, considering the difference between both. Although not a completely fair comparison due to the differences in network control, this test will be useful in obtaining a reference value for what are everyday communications.
Results show that GSM calls, with similar devices registered to the same operators, featured a delay of 241 ms. This delay is increased when looking at communication between two neighbor operators, reaching 335 ms. At this delay interval, the predicted call quality falls under a ITU level of “Some users dissatisfied”, envisioning obstacles during the call that will be noticed. Yet, these are the results obtained in commercial networks. Two final tests were made, this time with devices in WCDMA mode (Wideband Code Division Multiple Access). On the same operator, the resulting delay was 281 ms, 16 ms more when compared to the solution developed. On different operators, this value rises to 438 ms. One can see how comparing these results to the delay in a call between two mobile phones is not completely fair, as network conditions for those calls were unknown, whereas in the solution’s tests they were known and controlled, easing testing itself and creating more consistent results. Notwithstanding, as one can see from the results in Table 3, they do provide a good reference to the latency that can be expected in these traditional communications and one can see how this solution’s performance remains on par, since the lowest delays recorded for the calls were 281 ms (in WCDMA mode) and 241 ms (in GSM mode).

Table 3 - Mouth-to-ear test results (GSM tests)

<table>
<thead>
<tr>
<th>Test</th>
<th>Average delay</th>
<th>Total difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution calls</td>
<td>265</td>
<td>-</td>
</tr>
<tr>
<td>GSM calls</td>
<td>241</td>
<td>-24</td>
</tr>
<tr>
<td>GSM calls (different operator)</td>
<td>335</td>
<td>70</td>
</tr>
<tr>
<td>WCDMA calls</td>
<td>281</td>
<td>16</td>
</tr>
<tr>
<td>WCDMA calls (different operator)</td>
<td>438</td>
<td>173</td>
</tr>
</tbody>
</table>

VI. CONCLUSION AND FUTURE WORK

Advances and research are being made to create a higher degree of interoperability between telecommunication networks and the web-centered world, with WebRTC motivating new use-cases around the IMS architecture. This work is such an example, providing novel research on the tools available and describing a solution for a platform integrating both an IMS network and this emerging technology. Succeeding in both features exposed and performance constrains, one can see how the design decisions came to affect the platform and enhance its capabilities. Moreover, the evaluation made and yielded results, although not characterized at length, give insight to the behavior and performance expected from a solution such as this, which allows advanced communication over what are becoming increasingly deployed network architectures, shaping future communications.

Future iterations are considered, aiming to create a richer environment for communication convergence and growing the solution to expose richer services (such as billing features or location discovery, both featured on GSMA’s OneAPI specification). Another layer that could be improved is the media one, with the AS providing support for more codecs (only the G.711 audio codec was supported in this current implementation). This would enable the support for calls ranging in quality and a more flexible negotiation between clients for the codec in use.

ACKNOWLEDGMENT

The authors would like to thank the engineers at WIT Software. This work is funded by National Funds through FCT - Fundação para a Ciência e a Tecnologia under the project PEst-OE/EEI/LA0008/2013.

REFERENCES


