A Case for SIP in JavaScript

Kundan Singh
IP Communications Department
Avaya Labs
Santa Clara, USA
singh173@avaya.com

Venkatesh Krishnaswamy
IP Communications Department
Avaya Labs
Basking Ridge, USA
venky@avaya.com

***Abstract*—This paper presents the challenges and compares the alternatives to interoperate between the Session Initiation Protocol (SIP)-based systems and the emerging standards for the Web Real-Time Communication (WebRTC). We argue for an end-point and web-focused architecture, and present both sides of the SIP in JavaScript approach. Until WebRTC has ubiquitous cross-browser availability, we suggest a fall back strategy for web developers — detect and use HTML5 if available, otherwise fall back to a browser plugin.**

Keywords- SIP; HTML5; WebRTC; Flash Player; web-based communication

I. INTRODUCTION

In the past, web developers have resorted to browser plugins such as Flash Player to do audio and video calls in the browser and to interoperate with the legacy voice-over-IP (VoIP) systems from the web. To avoid the third-party plugin dependency, new standards for web real-time communication (WebRTC) are emerging to support the devices, codecs and communication primitives natively in the browser. While the global voice communications use the Session Initiation Protocol (SIP) [1] for signaling, the HTML5 standard as part of the WebRTC effort [2][3][4] keeps the signaling part outside the scope of the browser.

Traditional voice systems are built around a business model that requires universal reach hence interoperability among multiple VoIP vendors and services is crucial. By contrast, web based services are experimenting with business models capturing as many users within a single domain as possible. For instance, a user on a social networking web site is likely to communicate with others on that web site, but not on another web site. Thus, every web site can choose its own rendezvous (or signaling) protocol without worrying about interoperating with other web sites. We only need interoperability among a few browser vendors for the media path.

In practice, interoperability with legacy VoIP and telephone systems is crucial. This is done in either the end-point browser or a network gateway attached to the web server. In the former end-point approach, the SIP stack runs in JavaScript in the browser while using WebRTC for media path to connect with another SIP device. In the latter approach, if the client-server signaling is standardized, the same web application or the gateway can be reused with mix-and-match interoperability on several web sites. For the endpoint approach, the signaling is using SIP, whereas for the gateway approach, a custom protocol maps to SIP in the backend. We argue for the endpoint approach because it keeps the web developers build applications independent of the specific SIP extensions supported in the vendor's gateway.

We observe that decoupled development across *services*, *tools* and *applications* promotes interoperability. Today, the web developers can build applications independent of a particular service (Internet service provider) or vendor tool (browser, web server). Unfortunately, many existing SIP systems are closely integrated and have overlap among two or more of these categories, e.g., access network provider (service) wants to control voice calling (application), or three party calling (application) depends on your telephony provider (service). The main motivation of the endpoint approach (also known as SIP in JavaScript) is to separate the applications (various SIP extensions and telephony features) from the specific tools (browsers or proxies) or VoIP service providers. In practice, this is not guaranteed because the web site may restrict the hosted SIP-in-JavaScript application to only connect to its own proxy server. Moreover, several challenges make this approach nearly impossible to work in practice.

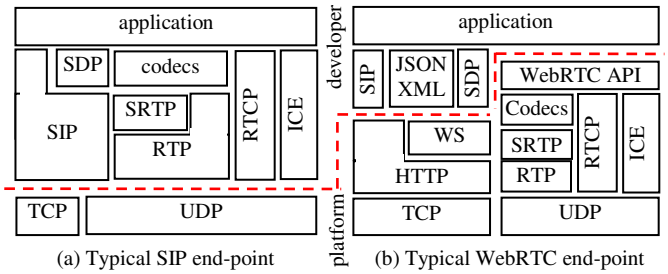
The paper is organized as follows. Section II gives a background on related technologies. Section III compares the two alternatives to interoperate between SIP and WebRTC. We present an implementation in Section IV. Section V describes the fall back strategy of using the Flash Player plugin by the web developers while WebRTC is being incrementally adopted by the browser vendors. Finally, we present our conclusions in Section VI.

II. BACKGROUND AND RELATED WORK

Before describing the interoperability between SIP and WebRTC, we give a brief background on these systems and their differences from the interoperability point of view.

A. What is SIP?

SIP [1] is the IETF standard for establishing, managing and terminating Internet sessions including voice and video calls and conferences. As shown in Fig.1 (a), a SIP system uses other standards such as SDP (Session Description Protocol) for offer/answer of session negotiation, RTP (Real-time Transport Protocol) for media path transport, RTCP (Real-time Transport Control Protocol) for feedback and control of media path, optionally SRTP (secure RTP) with keys negotiated in SDP for media path security, and optionally ICE (Interactive Connectivity Establishment) for traversal through intermediate NATs and firewalls. The dotted red-line separates what is programmed by the application developer and what is provided by the platform.



(a) Typical SIP end-point (b) Typical WebRTC end-point

Figure 1. Comparision of typical SIP vs WebRTC application stack

B. *What is WebRTC?*

WebRTC represents the family of emerging standards within the WebRTC working group in W3C [3] and the IETF RTCWEB working group [2] to enable end-to-end browser communication for real-time media. Please refer to [4] for an overview of WebRTC. It reuses existing standards such as mandatory SRTP (Secure RTP) for media transport, and mandatory ICE (Interactive Connectivity Establishment) for traversal through NATs and firewalls. The signaling messages are browser independent and are left to the application developer who would typically use HTTP (Hyper-Text Transfer Protocol) and WebSocket (WS) [5] for exchanging call control and session description information among the participants. Fig.1 compares the typical SIP and WebRTC application stack.

The WebRTC API proposal [3] uses SDP, which enables an application developer to use it as is, or transform it to web-friendly JSON (JavaScript Object Notation) or XML (eXtensible Markup Language). Optionally, the application can include a SIP implementation in JavaScript and reuse all the features provides by SIP.

C. *What is WebSocket?*

WebSocket (WS) is an IETF protocol and an HTML5 API that allows creating a bi-directional client-server connection from the JavaScript code in the browser to the web server. To traverse web proxies, it uses HTTP to initiate the first request and subsequently upgrades to a persistent connection via additional handshakes. Once the connection is established, it allows sending any data with packet boundary in either direction over TCP.

D. *Related Work in Interoperability*

In the early days of WebRTC, the IETF rejected having SIP in the browser and left the signaling to the application in JavaScript. Subsequent attempts to interwork between SIP systems and WebRTC enabled browsers fall in two categories: translation at the gateway or implementing SIP in JavaScript. SIP already supports several underlying transports such as TCP and UDP, and can be extended to support WebSocket as yet another transport, if needed [6][7]. This is now available in popular SIP proxy servers such as Kamailio and OfficeSIP. It lets developers implement SIP in JavaScript while promoting end-to-end media path if possible [8][9][10]. Translation at the gateway requires that both the signaling and media path go through the gateway [11][12]. This approach is being adopted by service and application providers to enable yet another way to connect to the service infrastructure. We compare these two approaches in the next section.

TABLE I. INTEROPERABILITY DIFFERENCES IN SIP VS WEBRTC. (O) MEANS OPTIONAL

Property	***SIP***	***WebRTC***
Media transport	RTP, SRTP (o)	SRTP, new RTP profiles
Session negotiation	SDP, offer/answer	SDP, trickle
NAT traversal	STUN (o), TURN (o), ICE (o)	ICE (includes STUN, TURN)
Media transport path/connection	Separate: audio/video, RTP vs RTCP	Same path with all media and control
Security model	User trusts device and service provider	User trusts browser but not web site
Audio codecs	Typically G.711, G.729, G.722, Speex	Mandatory Opus and G.711, optionally others
Video codecs	Typically H.261, H.263, H.264	Undefined yet but likely VP8 and/or H.264

Even though the media path uses the same set of protocols, achieving true media path interoperability between a SIP user agent and a WebRTC capable browser is a challenge. Table I summarizes the main differences between SIP and WebRTC for interoperability based on the discussions in the IETF. WebRTC requires new RTP profiles to send multiple RTP media streams as well as RTCP control packets multiplexed over the same transport path (or logical connection) so that the expensive step of negotiating an end-to-end path across firewalls is done only once. SIP end-points use a variety of optional techniques for NAT traversal, whereas WebRTC makes ICE mandatory. Instead of using the lock-step of SIP offer-answer, i.e., once an *offer* is made the end-point must wait for an *answer* before issuing another *offer*, WebRTC allows trickle or incremental change in the session description to promote incremental address gathering in ICE.

While there is some overlap in the mandatory audio codecs of WebRTC and the commonly used codecs in SIP systems, the working group is still undecided on the choice of mandatory video codecs. Moreover, the generic data channel proposed in WebRTC has no clear equivalent in existing SIP systems.

For the media path security, WebRTC requires SRTP. SIP systems using SRTP, while not ubiquitous, are nonetheless increasing. The origin-based trust model suggests that the two browsers must be visiting (or share some content on) the same web site to establish a WebRTC session. Finally, the end-user can trust her browser, but not the web site she is visiting.

Several open issues exist and are being debated in the IETF, e.g., the choice of mandatory codecs and the granularity of the API whether to give control of the ICE handshake to the application or not? How the issues are resolved will determine the future interoperability attempts with legacy systems.

III. INTEROPERATING BETWEEN SIP AND WEBRTC

This section first describes the interoperability deployment scenarios and then classifies the interoperability approaches to SIP in gateway versus endpoint.

A. *Deployment Scenarios*

The gateway for interoperability is commonly hosted by the VoIP service provider or the web application provider irrespective of whether SIP is implemented in JavaScript or not. As shown in Fig.2 (a) and (b), although the gateway technology is the same, who runs the gateway and what the core network is differentiates the two scenarios. The first approach applies to the existing web application and social network providers who want to interconnect their web users with phone users. The second approach applies to the existing VoIP service providers who would like to include web browsers and mobile devices as additional clients to their "managed" services network.

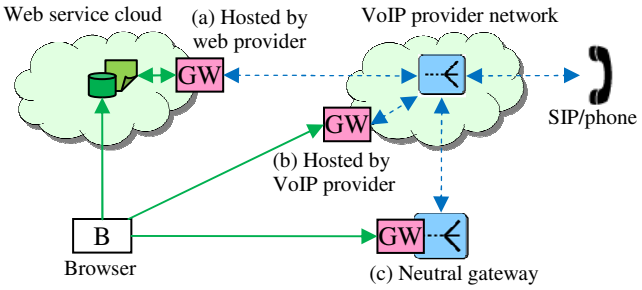


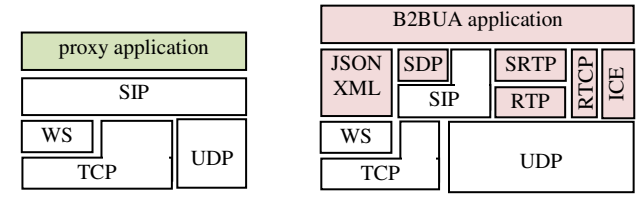
Figure 2. SIP-WebRTC deployment alternatives of the gateway

A third, less popular scenario, shown in Fig.2 (c), separates the gateway from the web site as well as the telephony provider and allows any web application to use any provider using an independent third-party gateway. This is an application obtained from one vendor (website) but connecting to the services of another vendor (telephony). The signaling interoperability in (a) and (b) is limited to a single vendor implementation where the application is tied to the gateway or service, whereas the third approach has a clear separation of the application (web pages) from the tool (browser, gateway) and the service (web site or VoIP provider).

B. *SIP in Gateway or Endpoint*

The gateway (GW) in Fig.2 translates between web-based signaling and SIP. In this *gateway approach*, the web application relies on the web-signaling API of the gateway and, unless a standard is defined, prevents true mix-and-match replacement of applications and tools. Alternatively, SIP itself is used as the signaling protocol, implemented in JavaScript and running in the web browser. We call this the *endpoint approach*. Instead of a gateway, it uses a SIP proxy server in the network that supports WebSocket [6]. The two approaches are further compared below.

Fig.3 compares the implementation complexity in the block diagrams of (a) a SIP proxy with WebSocket that is needed in the endpoint approach, and (b) a gateway that translates between WebRTC and SIP. The former is required as a network element for a SIP endpoint in JavaScript in the browser, whereas the latter is a network element that relies on a custom signaling protocol over WebSocket from the browser. In particular, assuming that the media path can go end-to-end between the browser and the SIP device, the SIP proxy does not deal with the media and session description components, whereas the gateway initiates and terminates signaling and, sometimes, media.



(a) SIP proxy with websocket (for endpoint approach)

(b) SIP-WebRTC gateway (for gateway approach)

Figure 3. Network elements for (a) endpoint approach and (b) gateway approach. The highlighted parts are different in the two systems.

The gateway approach limits the web developer to the vendor supplied functions such as for call transfer, multiple devices in the same call, and other advanced features of a SIP user agent. On the other hand the endpoint approach allows the web developer to implement any such SIP extension or feature in JavaScript. It also means that SIP is used as is from the browser unlike the gateway that defines custom APIs for existing and new SIP extensions. Thus, in the endpoint approach, the web developer can build applications based on SIP extensions and features independent of the gateway vendor. The portable SIP stack in JavaScript avoids any platform related issues. This results in a very thin server (or SIP proxy) that does not need to maintain the session state, and thus, creates more scalable and robust network elements with distributed clients and client-initiated fail-over strategies.

TABLE II. SIP IN GATEWAY VS ENDPOINT

Property	***Gateway***	***Endpoint***
Interoperates with existing (good)	Yes	No
Dependent on tool vendor (bad)	Yes	Yes
Dependent on service vendor (bad)	Yes	No
App developer adds features (good)	No	Yes
Can hide the source code (good)	Yes	No
Client complexity (neither good nor bad)	Low	High
Server complexity (bad)	High	Low

Table II compares the two approaches. The main advantage of the endpoint approach is that the (web) application developer can add features and extensions independent of the VoIP service provider. In the gateway approach, the vendor dependency locks the web applications to a single service and gateway. Vendor independence is likely to create more applications because there is manifold more number of web developers writing client JavaScript than the vendors working on gateways and servers.

Certain drawbacks of the endpoint approach are: due to the mandatory requirements (see Table I), WebRTC is quite unlikely to interwork out-of-the-box with existing SIP devices on the media path. While one can obfuscate the JavaScript code, it is difficult to hide the source code of the SIP application against theft from snoopy web users.

Although, the endpoint approach allows interoperability, every new feature has a dependency on the standards which could limit the flexibility of adding a new feature, unlike in the gateway which could add new custom API for non-standard features. The vendors and service providers with significant

investment in the form of session border controller (SBC) or back-to-back user agent that terminate SIP in the network and require an intermediate entity in the signaling and media path, have no perceived advantage of moving SIP to the browser. Finally, the battery life of running a full SIP stack instead of a light weight custom protocol on the mobile phones needs to be studied.

IV. IMPLEMENTATION

The SIP-JS [8] project has an implementation of SIP and related standards in JavaScript and a demonstration of a web-based phone using this stack. The portable SIP/SDP stack is about 3.5k source lines of code in JavaScript, and the phone application is about 2.5k lines in JavaScript, HTML and CSS. The application uses the native WebSocket and WebRTC extensions in the Google Chrome browser for the signaling and media path, respectively. It has another mode to use the Flash Player plugin and a host application for network transport and device access for those browsers that do not have native WebSocket and WebRTC capabilities. The SIP-in-JavaScript stack is reused in both the modes. The web-based phone implements basic registration and video call using a SIP proxy that supports WebSocket. Others have also built the SIP stack in JavaScript, e.g., [9][10].

V. INTEROPERABILITY STRATEGY

Although an implementation of a SIP-stack-in-JavaScript is independent of HTML5, a SIP endpoint running in the browser has two basic requirements - a way to transport the SIP signaling messages to and from the SIP proxy server and a way to access end user devices to establish the media path for voice and video communication. These requirements are ideally filled by WebSocket and WebRTC extensions available in HTML5.

WebRTC represents a promising next generation technology. However, this requires significant changes in the browsers today. In the past, minor incompatibilities among HTML browsers such as margin or padding have been a nightmare for developers. Fortunately, HTML5 adoption is growing rapidly among browser vendors. However, extending HTML5 with a complex concept such as WebRTC is expected to cause more interoperability problems. Thus, WebRTC will take some time before it is consistently available in all the popular browsers, or it may never happen. In that case, we need a strategy so that web developers can continue to innovate with HTML5 if the support is detected in the browser and fall back to the legacy plugin otherwise.

There are other scenarios such as when WebSocket and WebRTC are supported by the end user's browser but do not work in the network due to the enterprise firewalls blocking the media path or old HTTP proxies not correctly handling the WebSocket handshake. Such scenarios are for further study. Here in Fig.4 we only list the possible interoperability scenarios from the most preferred to the least based on whether WebRTC and/or WebSocket is available in the browser or whether the remote SIP endpoint implements WebRTC related profiles.

A web application that intends to use WebRTC would need to fall back to an alternative if WebRTC extensions are not available in the end user' browser. For a SIP end-point in the browser, the fall back decision happens at multiple steps. For example, the signaling over WebSocket and media over WebRTC is the most optimal configuration. If these HTML5 features are not available, then we could use an efficient separate application to do the transport and device implementation that keeps the standard signaling and media path in the end point without relying on a server to do the translation. If that fails, we detect and use an in-network gateway capable of translation. If that fails, we fall back to using the Flash Player plugin that uses RTMP (Real-Time Messaging Protocol) for both signaling and media.

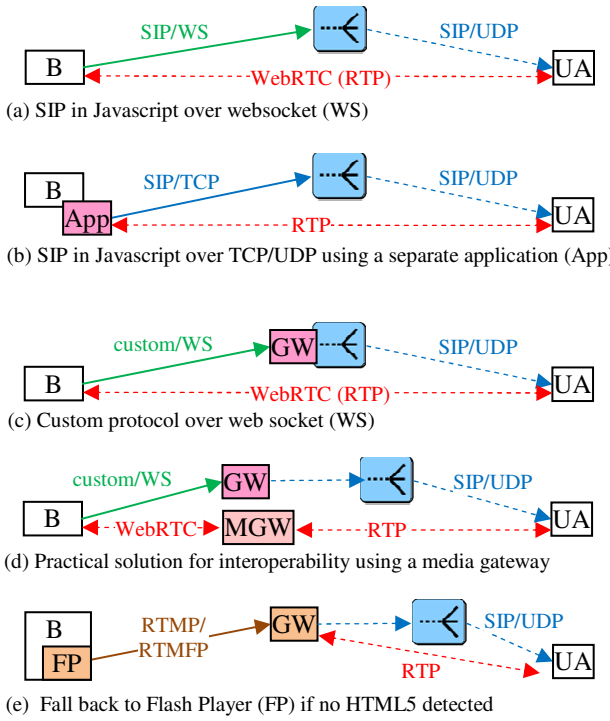


Figure 4. Architectures for SIP-WebRTC interworking highlighting the most preferred (a) to the least preferred (e) configuration.

While browsers have competed against each other for the market share, the Adobe Flash Player plugin has played a dominant role in its own domain covering majority of the Internet connected users. Fortunately, Flash Player can fill both the end point requirements – provide a transport for SIP signaling and access devices for media path. Thus, Flash Player forms a natural fallback strategy for a missing WebRTC or WebSocket support in the browser. Secondly, certain UDP-blocking corporate firewalls also block WebRTC traffic. In such cases, failing over to a traditional Flash Player plugin based approach that can readily use HTTP tunneling for media could provide an intermediate solution until appropriate standards are developed to tunnel WebRTC over HTTP, for example. The similarity in the WebRTC's JavaScript API and the Flash Player's communication related ActionScript API allows us to create a wrapper layer which combines various elements in to a widget and use it for various communication use cases such as two-party call, multiparty conferencing and video messaging.

The media codecs inventory available in Flash Player is different from the proposed WebRTC codecs capabilities. This poses additional complexity in interoperating when some participants use WebRTC but others fall back to Flash Player. In particular, Flash Player is capable of capturing and encoding media using Nellymoser, Speex and G.711 for audio, and Sorenson and H.264 for video, whereas WebRTC is likely to define G.711 and Opus as mandatory audio codecs and VP8 *and/or* H.264 as mandatory video codec (see Table I). In addition to the required codecs, the browsers are free to implement additional codecs. Thus, the web application should do session negotiation before arriving at the common set of codecs among the participants.

The RTP profile for WebRTC requires multiplexing multiple media streams as well as media control messages on a single port. However, existing SIP devices do not support these extensions. Hence, either these devices will need to be upgraded or intermediate media gateways will facilitate conversion.

VI. CONCLUSIONS

There is a need to interoperate between traditional SIP systems and emerging WebRTC standards, and we have presented alternatives to where the interworking is done and where it gets deployed. Fig.4 shows the message architecture for some of the alternatives. In particular, option (a) of using SIP in Javascript with native WebSocket transport for signaling and interoperable end-to-end media path over WebRTC is ideal because it allows us to keep the tools, services and applications separate from each other while potentially improving the overall scalability and robustness. Option (b) of using a separate host-resident application to facilitate transport and device access functions to the browser has similar motivation but requires an additional download. The custom signaling in option (c) and (d) fits with the technical and business requirements of the existing VoIP and web services, but reduces mix-and-match interoperability among various applications and tools. If end-to-end media path cannot be achieved, then an intermediate media gateway is used for re-packetization and/or transcoding as in option (d). Finally, if everything else fails then the traditional approach of using a browser plugin is used as in option (e). We believe that the success of WebRTC will depend on its adoption by browser vendors as well as the innovative applications built on it. A clear separation among the tools, services and applications promotes decoupled development of client applications from the services. However, a lot depends on how things evolve in the near future regarding the business needs and technology adoption among the competing vendors.

There are several things that that could go wrong in the true motivation of SIP in JavaScript. It has a dependency on a consistent and simple WebRTC standard – what if some browser vendors do not implement WebRTC? Or implement it differently? What if there are browser backdoors that prevent creating cross-browser applications? The choice of audio and video codecs by different browser vendors could force use of an in-network transcoder or media gateway. Finally, if the vendors cannot figure out how to make profit – especially with the risk of opening up the JavaScript source code, and the lack of control over the endpoint application – they may not adopt this.

REFERENCES

[1] J. Rosenberg et al., “SIP: Session Initiation Protocol”, IETF, RFC 3261, June 2002.

[2] Real-Time Communication in WEB-browsers (RTCWEB) IETF working group, http://tools.ietf.org/wg/rtcweb

[3] WebRTC 1.0: Real-Time Communication Between Browsers, W3C Working Draft, Aug 2012, http://www.w3.org/TR/webrtc/

[4] S.Loreto and S.P.Romano, "Real-Time Communications in the Web: Issues, Achievements and Ongoing Standardization Efforts", IEEE Internet Computing, pp.68-73, Volume 16, Issue 5, Sept-Oct 2012.

[5] The WebSocket API, W3C candidate recommendation, Sep 2012, http://www.w3.org/TR/websockets/

[6] I.Castillo et al., “The WebSocket Protocol as a Transport for SIP”, IETF, work in progress, Oct 2012, draft-ietf-sipcore-sip-websocket

[7] SIP on the web, project website, http://sip-on-the-web.aliax.net/

[8] SIP-JS: SIP in JavaScript project site, http://code.google.com/p/sip-js

[9] SIPML5: HTML5 SIP client, project website, http://sipml5.org

[10] jsSIP: The JavaScript SIP library, project website, http://jssip.net

[11] SIP audio interworking demo, http://kapejod.org/webrtc/index-sip.html

[12] WebRTC interworking with traditional telephony services, Ericsson Labs Blog, Feb 2012, https://labs.ericsson.com/blog