Signalling-On-the-fly: SigOfly

WebRTC Interoperability tested in contradictive Deployment Scenarios

Paulo Chainho
Coordenação Tecnológica e Inovação Exploratória
PT Inovação e Sistemas
Oeiras, Portugal
paulo-g-chainho@telecom.pt

Kay Haensge, Steffen Druesedow
Telekom Innovation Laboratories
Deutsche Telekom
Berlin, Germany
{kay.haensge|steffen.druesedow}@telekom.de

Michael Maruschke
Hochschule für Telekommunikation Leipzig (HfTL)
University of Applied Sciences
Leipzig, Germany
maruschke@hftl.de

***Abstract*— this article presents the results of WebRTC trials conducted in a partnership between Telekom Innovation Laboratories (T-Labs) and PT Inovação e Sistemas under the project called WONDER. The main achievement of WONDER project was the design and validation of the new "Signalling-On-the-fly" (SigOfly) concept. This concept ensures interoperability between any WebRTC administrative domains without the need to use a standardized signalling protocol such as SIP. This result paves the way for the design of a new more agile and competitive service architecture as an alternative to the current IMS (IP Multimedia Subsystem) architecture.**

Keywords— WebRTC, Signalling, IMS, Service Architecture

I. INTRODUCTION

Many analysts envisioned that Web Real-Time Communications (WebRTC) will introduce further and deeper disruptions in the telecommunications industry predicting more than 2 Billion WebRTC users and more than 6 Billion WebRTC devices by 2019 [1]. WebRTC ubiquity will make it as simple for any Web developer to create and provide a rich, real-time communication experience as it is to create a web page. Furthermore, such WebRTC Applications can easily bypass the traditional network control plane: every HTTP server can act as a control plane between two or more users. However, Telecommunication Operators (Telcos) can also take advantage of WebRTC characteristics as an opportunity to embrace new technical approaches to build a more competitive service infrastructure that leverages recognized merits of Telco operated services in terms of security, identity management, interoperability and Quality of Service. The Eurescom Study P2252 characterized the WebRTC threats and opportunities for Telcos providing a set of recommendations on how to make the best use of WebRTC [2]. The WONDER Project (WebRTC InterOperability tested in coNtradictive DEployment ScenaRios), was conducted in a partnership between Telekom Innovation Laboratories (T-Labs) and Portugal Telecom Inovação e Sistemas, in order to experimentally evaluate the recommendations of Eurescom study P2252. Since WebRTC is being specified on the user data exchange as well as the WebRTC Client's (or WebRTC Peer's) Application Programming Interface (API) handling, the World Wide Web Consortium (W3C) does not specify any signalling protocol for initiation and maintenance of a communication session between the WebRTC Peers. Such lack of WebRTC signalling protocol standardization makes WebRTC interoperability a very challenging topic. The WONDER project conceived a new interoperability concept called "Signalling-On-the-fly" (SigOfly) that was successfully validated with a first proof of concept that demonstrates there is no need to have Network to Network Interface (NNI) standard signalling protocols - like Session Initiation Protocol (SIP) - to achieve interoperability between any WebRTC service provider domains.

This article presents the main results of the WONDER project starting by describing the reasons that guided the project activities and in particular the reasons that led to the SigOfly concept. Section III introduces the SigOfly concept and Section IV the structure of the WONDER library, which was designed and developed to experimentally validate the SigOfly concept. In section V, the results of experiments conducted with the WONDER library are presented and, at the end, in section VI conclusions and future work is presented, by introducing the reTHINK project where the new Hyperty service paradigm leverages the SigOfly concept to design a new, Web-centric Peer-to-Peer Service Architecture as an alternative to an IMS Architecture.

II. MOTIVATION

The WebRTC technology enables real-time communication between web browsers without the need to install additional applications or plug-ins. Consequently, any device that has a web browser will be able, in a native form, to

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 287581 - OpenLab and from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645342. This publication reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

support voice and video, or any other service, in real time (e.g. conferencing, voice calls, video, or games). To this end, the web browser includes a media engine that can process media and data streams using IETF standardized transport protocols like Real-Time Transport Protocol (RTP) [3] and RTP Control Protocol (RTCP) [4], that are already used today in Voice over IP networks, notably in the user-data plane of IMS network architectures. Additionally, WebRTC technology also supports the transmission of data e.g. file transfer, by means of the Stream Control Transmission Protocol (SCTP), originally designed for the reliable transport of Signalling System No.7 (SS7) over IP [5][6]. The WebRTC technology adopted some extensions to these protocols, and in particular, paid special attention to security aspects by making the encryption of all streams of media and data via Datagram Transport Layer Security (DTLS) technology mandatory [7]. The capabilities of the WebRTC media engine are exposed through JavaScript APIs standardized by W3C in [8], which Web developers can use to incorporate real-time communication features in Web Applications.

The telecommunication service architectures like IMS are based on the standardization of network control and connectivity interfaces, usually called signalling. However, the WebRTC standards did not specify any signalling protocol. In authors' opinion, this decision is positive, by giving developers the freedom to choose the most appropriate protocol to meet the specific needs of each application. The fact that a signalling protocol standard is not required also reduces the time-to-market of WebRTC technology, minimizing standardization tasks, which are always very time consuming.

Since media and data protocols are standardized, the lack of a signalling protocol does not prevent interoperability between users, even if they are subscribers of different service providers. The calling party just has to have a URL from the called party to download its web application and to establish a WebRTC session with them. They both connect to the same web server and will then utilize the same signalling scheme. This is a new paradigm for Telcos that is often not easily embraced by traditional mindsets.

While this approach is acceptable for many applications, it leaves some open questions such as

- "How to ensure control of the user experience and the session by the calling party?" and
- "How do you avoid vendor lock-in to proprietary signalling protocols?".

The WONDER project focused its activities to address these issues while retaining the original WebRTC model, including:

- To ensure interoperability between different service providers, each using different application-specific signalling protocols;
- Avoid the need to have federated domains promoting low barriers for the entry in the market, of new small service providers;
- Use a triangular topology network signalling, where a server is only used for signalling interoperability between two domains of services, minimizing the use of network resources;
- Ensure portability of WebRTC applications between different network solutions, including web-based and IMS technologies, even if they are provided by different vendor solutions.

These are the basic questions that guided the WONDER project, which led to the design and experimental validation of the new concept of SigOfly, presented in the next section.

III. The Signalling-On-the-fly (SigOfly) Concept

A. Terminology

The SigOfly concept leverages the use of scripts (JavaScript) by WebRTC Applications to implement signalling protocol stacks. This means, the signalling protocol stack can be selected, loaded and instantiated during runtime. Such characteristic enables signalling protocols to be selected per WebRTC Conversation to ensure full signalling interoperability among peers using Triangle based Network topologies. The SigOfly procedures should be applied at the end-user client to benefit from WebRTC model as introduced earlier in the previous section. However, the concept is also feasible between Messaging Servers supporting Javascript execution engines (e.g. nodejs or vertx.io).

Before the SigOfly concept is described in detail, some terms require a definition:

Messaging Server: the server that supports the exchange of signalling messages required for the establishment of WebRTC sessions. Each Messaging Server belongs to a domain;

Domain Channel: the signalling channel that is established with domain's messaging server as soon as a domain's user is registered and is online;

Transient Channel: the signalling channel that is established, typically with a foreign messaging server (i.e. from another domain) in scope of a inter-domain conversation;

Messaging Stub: the script containing the protocol stack and all the logic needed to establish a channel to a certain Messaging Server;

Conversation Hosting Messaging Server: is the Messaging Server that is used to support the exchange of all signalling messages among peers belonging to different domains. The Hosting peer uses the Domain Channel to exchange signalling messages, while other peers use Transient Channels that connect to the Hosting Messaging Server.

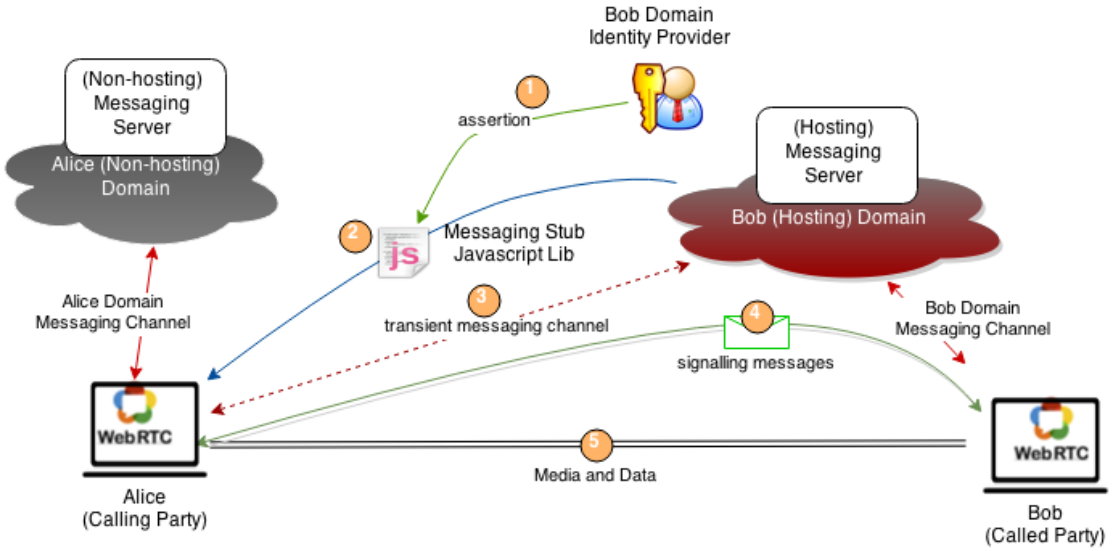

Fig. 1: Conversation hosted by called party domain

B. Conversation hosted by called party domain

The classic Alice and Bob example is used to explain the SigOfly concept. We assume that Alice and Bob are registered in different Service Provider domains having each one a Domain Channel established with their own Messaging Server (see Fig. 1). In case Alice wants to talk to Bob by using Bob's WebRTC identity e.g. bob@domain.com, the following steps will be performed:

Step 1: Information about the Identity of Bob, including Bob's Messaging Stub provider, is provided and asserted by Bob's Identity Provider (IdP).

Step 2: Alice downloads and instantiates Bob's Messaging Stub in her browser to setup a Transient Channel with Bob's domain Messaging Server.

Step 3: As soon as the Transient Channel is established, Alice can send an Invitation message to Bob containing her Session Description Protocol (SDP) based communication offer.

Step 4: Since Bob is connected in the same Message Server via his Domain Channel, he will receive Alice's invitation in his Browser. If Bob accepts the invitation, an Accepted message containing Bob SDP response will be send

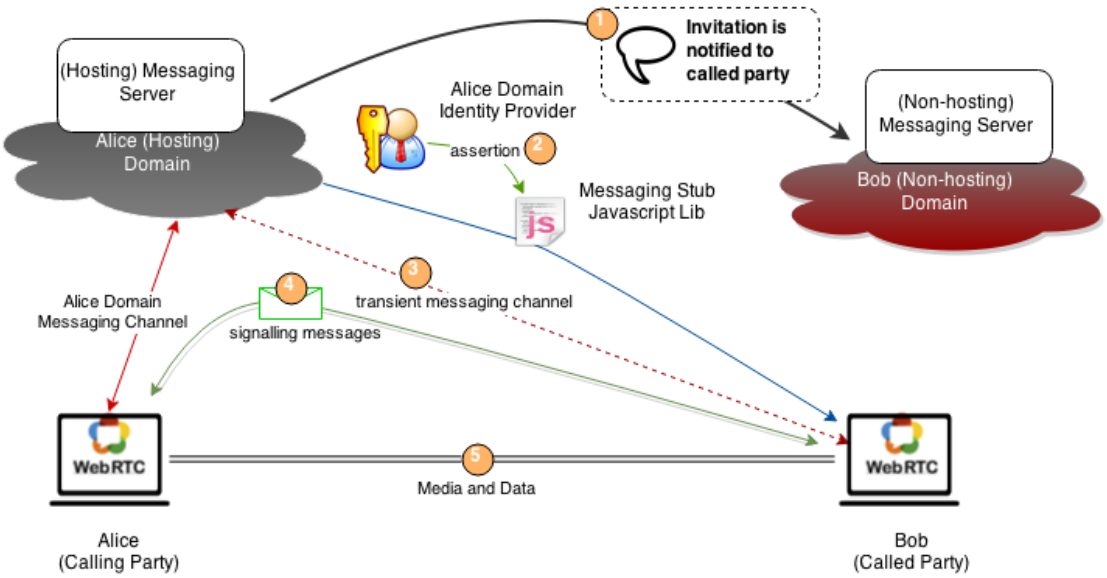

Fig. 2: Conversation hosted by calling party domain

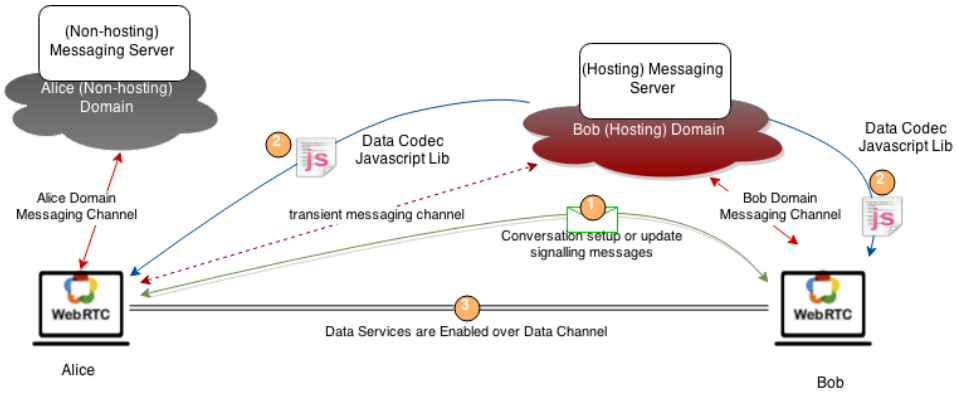

Fig. 3. Data Codec On-the-fly procedures

to Alice.

Step 5: As soon as Alice's browser receives Bob's SDP, the media and/or data streams can be directly connected between the two browse

It should be noted that SigOfly does not directly address identity management aspects but aims to be compliant with ongoing WebRTC Identity Management work from W3C in [8] and IETF in [9], mainly by extending RTCIdentityAssertion to also include the assertion of MessagingStubs. This means, Alice and Bob authentication is done outside SigOfly procedures, described above, which are agnostic of the IdP and authentication protocols used.

C. *Conversation hosted by calling party domain*

The procedures described above are applicable to conversations hosted by the called party domain where the messaging server, used to exchange signalling messages, belongs to the called party domain. This means the called party domain is spending more resources than the calling party domain. In case the called party does not accept to spend more resources, the conversation can be hosted by the calling party domain. The main difference is that, a RESTful notification service endpoint is asserted from Bob's IdP, which is used to push an invitation message containing Alice's offer towards Bob device (see Fig. 2). From this point, procedures are very similar to conversations hosted by called party, but now peer roles are swapped: Bob is the one that downloads Alice's Messaging Stub to setup a Transient Channel with Alice's domain Messaging Server.

D. *Multiparty Conversation*

Multiparty conversations with more than one user coming from different domains are also supported by the SigOfly concept. Different Network Topologies can be used including:

- Mesh Topology with a Hosting peer, where all peers have direct media and data streams established with all remaining peers and a single Hosting Messaging server is used i.e. all peers have a signalling channel established with the same Messaging Server.
- Multipoint Control Unit (MCU) based Topology with a Hosting peer, where peers have media and data streams established with a central media server that mixes and distributes streams among the peers, and a single

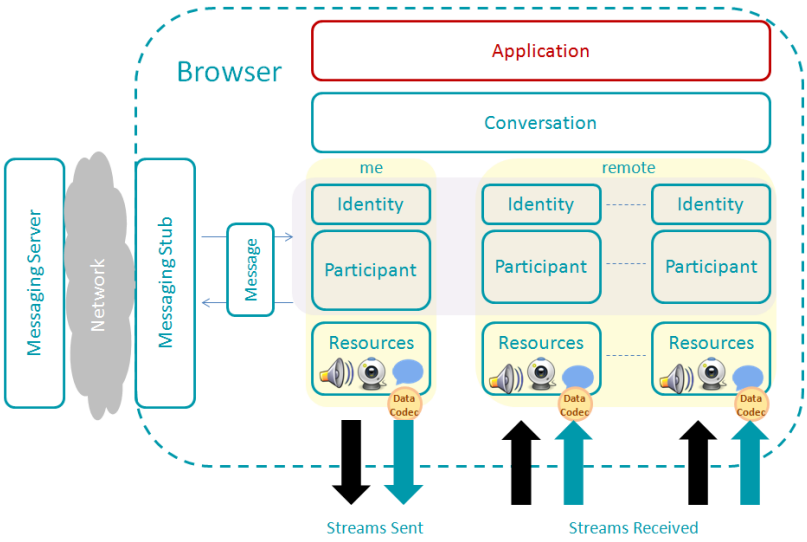

Fig. 4. Main WONDER Classes

Hosting Messaging server is used i.e. all peers have a signalling channel established with the same Messaging Server.

To bring multiparty functionality into conversations requires a more complex messaging management on each participant, because each connected peer may have a certain connection state which differs from the overall multiparty conversation. Multiparty Conversations were not the main topic of the WONDER project and since this is a complex problem it will be addressed in future activities.

E. Data Channel Services Interoperability

WebRTC standards don't specify how different services, like chat or file transfer, can be carried on top of WebRTC data channels. The use of the SigOfly concept per se is not sufficient to ensure full interoperability between different service domains that are offering Data Channel-based services. For example, one service provider may offer a Chat service that uses Message Session Relay Protocol (MSRP) [10] on top of the Data Channel and another service provider can use a different proprietary application specific chat protocol e.g. plain JavaScript Object Notation (JSON) on top of Data Channel. In this case, the SigOfly concept will provide full interoperability for audio and video services but not for Data Channel based chat communication.

To address Data Channel Services Interoperability, the WONDER project has conceived the "Data Codec On-the-fly" mechanism which applies the SigOfly principles to ensure all peers are using the same protocol on top of the Data Channel. A Data Codec is a JavaScript library that implements a data communication processing algorithm to code and send data to the Data Channel and to receive and decode data from the Data Channel. As with Messaging Stubs in the SigOfly concept, Data Codecs are downloaded and instantiated by peers according to URLs identified in Conversation setup or update signalling messages (see Fig. 3).

IV. WONDER JAVASCRIPT FRAMEWORK

A JavaScript framework, the WONDER lib, was designed and implemented to validate the SigOfly concept. Main WONDER library classes are:

- The *Identity,* representing a user and containing all information needed to support Conversation services. This also includes the service endpoint to retrieve the protocol stack (Messaging Stub) which will be used to establish a signalling channel with the Messaging Server of the Identity domain. The Identity entity extends the current Identity concept defined in [8] to support seamless interoperability by using the SigOfly mechanism.
- The *MessagingStub* implementing the protocol Stack used to communicate with a certain Messaging server.
- The *Conversation* class managing all participants including the setup, update or close of media and data connections.
- The *Participant* class, handling all operations needed to manage the participation of an Identity (User) in a conversation including the WebRTC PeerConnection functionalities. The Local Participant is associated with the Identity that is using the Browser while the Remote Participant is associated to remote Identities (users) involved in the conversation.
- The *Resource* class representing the digital assets that are shared among participants in the conversation including participants' voice, video, screens, photos, video Clips, music clips, documents, etc. These assets are usually managed by the Participant that owns it. For local participants assets are sent (e.g. WebRTC outgoing stream tracks) while for remote participants assets are received (e.g. WebRTC incoming stream tracks). Some Resource types like Chat are not managed by a Participant but by the Conversation.
- The *Data Codec, which* is used by Resources that are shared on top of the Data Channel, like file sharing and Textual Chat, to decode and encode the data in a consistent way by all the peers. The Data Codec may also be downloaded on-the-fly by the peers.
- The *Message,* which is used to exchange all data needed to setup, update and close media and data connection between peers via the Messaging Server. It may also be used for other purposes e.g. presence information management. Each message is comprised by a Header and a Body.

A. Procedure to setup a Conversation

There are only some small steps to create a WebRTC Conversation with the WONDER Framework. Fig. 5 depicts the sequences to start a conversation with another WebRTC peer. The Application developer will only have to program the steps that are depicted in black colour while internal WONDER library steps are depicted in red colour.

- Inside the web application of Alice, an Identity will be created by calling the IdP as well as the Identity with the targeted RTCIdentity (Bob).
- The web application then creates the Conversation object and performs the creation of an InvitationMessage, too. Then, the message will be used to open the Conversation.
- The Conversation creates Participant objects for each of the Conversation's peers, i.e. a MySelf Participant object (that will handle WebRTC getUserMedia primitives) and the remote Bob Participant object (that will handle WebRTC peerConnection primitives). For simplification purposes only the instantiation of the remote Bob Participant is depicted.
- The Bob Participant object then triggers the download and instantiation of Bob's domain MessagingStub to establish a Transient Channel between Alice's browser and Bob's MessagingServer.
- The original InvitationMessage is then forwarded to Bob's domain MessagingStub which will translate it into ProtocolInvitationMessage, according to the

signalling protocol used by Bob's domains, and send it to the targeted MessagingServer on the Network side.

- The Messaging Server of Bob's domain interprets the incoming message and forwards it towards Bob browser by using its Domain Channel. Bob's MessagingStub will receive the ProtocolInvitationMessage and will translate it into the WONDER InvitationMessage. Then this stub forwards the InvitationMessage to Bob's App.
- In case the Conversation invitation is accepted by Bob, a Bob's Conversation object is created and the Accept function is called having the incoming Invitation message as parameter. The Conversation object will create MySelf Participant (that will handle all needed WebRTC getUserMedia procedures), remote Alice participant (that will handle all needed WebRTC peerConnection procedures) and an AcceptedMessage. These transactions are not depicted for simplification purposes. The AcceptedMessage is send via Bob's MessagingStub which will translate it into the signalling protocol used and forward it towards Bob's MessagingServer.
- Bob's MessagingServer handles the ProtocolAcceptedMessage and forwards it towards Alice's browser via the Transient Channel. The ProtocolAcceptedMessage is translated into WONDER Accepted Message and forwarded to Bob's Participant that will handle all WebRTC peerConnection procedures to setup the Media Streams.

V. Tests and Results

The WONDER library was used in different experiments to validate the SigOfly concept. Experiments were performed by using an OpenIMS based testbed effectively operated by University of Patras in the OpenLab project [19]. The testbed was extended and configured to emulate four different WebRTC domains, namely:

- IMS - the ims.wonder domain uses an IMS-Signalling Gateway, which translates a JSON over WebSocket signalling protocol into SIP and vice-versa. Therefore there is no need for any SIP library in the browser.
- Node.js - the nodejs.wonder domain uses a JSON over WebSocket protocol provided by a Node.js message server. This domain acts as a pure web domain with no relation to any Telco pattern.
- Asterisk - the asterisk.wonder domain works with the SIP over WebSocket signalling protocol and uses a non-IMS infrastructures backend based on the Doubango SIPML5 and Webrtc2sip gateway solution integrated with Asterisk VoIP platform
- Vertx - the vertx.wonder domain uses a JSON over WebSocket signalling protocol provided by a vertx.io Message Server.

Table I, lists the experimentation results. In general inter-domain experiments were very successful, demonstrating that the SigOfly concept can be used to enable seamless interoperability between any WebRTC domains with no use of NNI standard protocols.

For IMS-based domains the tests for multiparty

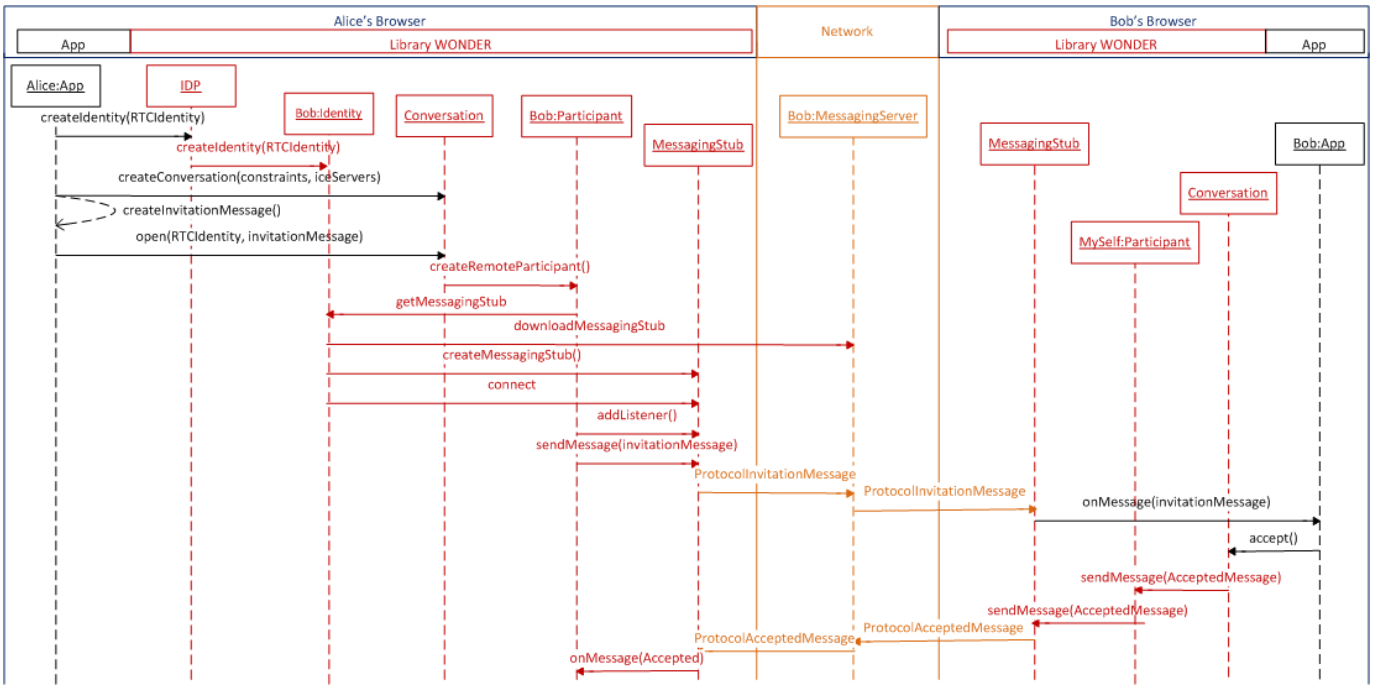

Figure 5: Procedure to Setup a WONDER conversation

conversations were not performed since the algorithm used implies the exchange of signalling messages outside of SIP dialogs. This requirement is challenging for IMS-based architectures. Such mechanism may be enabled by using the event invoking SIP message types SUBSCRIBE, PUBLISH and NOTIFY. Nevertheless, this solution requires additional resources on the signalling level. Furthermore, it would also require changes in standard IMS-Clients to process such messages accordingly.

No major impact was noticed in the Conversation setup performance introduced by the protocol stack download procedure, used by the SigOFly concept. Only a few milliseconds are needed to download Javascript files which are smaller than 100KB. In addition, the protocol stacks are cached for further conversations. For more complex protocol stacks, e.g. SIP over WebSockets, implemented in large Javascript files (>1MB), in-advance file download mechanisms may be used to minimize impact on Conversation setup performance.

Looking into the summary experimentation results tables we may conclude Web centric delivery approach had more successful results than the IMS centric. This result can be seen as a surprise since IMS is a mature architecture with a large set of services available, while WebRTC is still in very early stages (not a standard yet). In reality WONDER experimentation didn't take much advantage of existing services namely Presence and XML Document Management Server (XDMS) services, due to the amount of integration effort it would demand. Nevertheless, this also indicates how IMS option implies further integration efforts when compared with the Web centric option.

TABLE I. INTER-DOMAIN INTEROPERABILITY TEST RESULTS

Tested Domains	Service Features			
	Basic-AV	***Rich Features***	***MCU Multiparty***	***Rich Mesh Multiparty***
Vertx<>nodejs	OK	OK	OK	OK
Vertx <> IMS	OK	OK	OK	NOK
Vertx<>Asterisk	OK	OK	OK	NOK
Nodejs<> IMS	OK	OK	OK	NOK
Nodejs<>Asterisk	OK	OK	OK	NOK
IMS<>Asterisk	OK	OK	OK	NOK

VI. CONCLUSION AND FUTURE WORK

The WONDER project has demonstrated that no NNI standard protocols are needed to achieve seamless interoperability between any WebRTC domains. A standard and protocol-agnostic JavaScript API, like the WONDER API, should be used instead, promoting portability of Applications among different back-end solutions. Such approach, also benefits service providers by minimizing dependencies between Applications and back-end vendors. Until now, one of the rationales to use IMS based back-end solutions was the need to have NNI standard interfaces based on SIP to ensure full interoperability between different Service Provider domains. The successful demonstration of the SigOfly concept also means this rational is not valid anymore. At the end this means a web centric delivery approach using more agile and simpler architectures is feasible and paves the way for a future Web centric standard Service Architecture as an alternative to IMS.

The WONDER JavaScript library has been published in a GitHub repository in [13] along with tutorials in [14][15][16] and live demos in [17].

The WONDER library is currently being enriched with further features including contact list and group management features, user presence management and chat adhoc features (i.e. chat supported on top of the signalling channel). It is also planned, to improve multiparty conversations handling and to adapt the library to be compliant with emerging ORTC (Object Real Time Communication) work [19].

The WONDER SigOfly and "Data Codec on-the-fly" concepts will be further investigated in more use cases (e.g. supplementary services) and applied in new application domains including Internet of Things and Content Delivery service domains, as part of the reTHINK (Trustful hyper-linked entities in dynamic networks) project, where special attention will be given to security aspects [20]. The reTHINK projects main goal is the design and demonstration of a new, Web-centric Peer-to-Peer Service Architecture moving from a Client-Server paradigm (e.g. RESTful architectures) towards a more powerful service concept paradigm called Hyperlinked Entities or just Hyperties. Hyperty is a Web Service or a Web application that provides a communication endpoint or peer to exchange data with another Hyperty or with a set of other Hyperties in a peer-to-peer fashion. The Hyperty service concept leverages the simplicity of REST-based Web Services in a managed peer-to-peer network. Policy Management and Identity Management architectural approaches will be investigated to enforce certain service execution behaviour e.g. access control to conversations (Communication Barring) and Communication diversion.

References

[1] D. Bubley,, WebRTC Market Status & Forecasts, 2014 Edition, Disruptive Analysis report, October 2014, http://disruptive-analysis.com/webrtc2014.htm

[2] "P2252 - Telco strategic positioning options regarding WebRTC", Eurescom Study, 2012

[3] "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-11, H. Alvestrand, August 18, 2014, http://tools.ietf.org/html/draft-ietf-rtcweb-overview-11

[4] "Transports for WebRTC", draft-ietf-rtcweb-transports-06, H. Alvestrand, August 11, 2014, http://tools.ietf.org/html/draft-ietf-rtcweb-transports-06

[5] "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-07, R. Jesup, S. Loreto, M. Tuexen, July 4, 2014, http://tools.ietf.org/html/draft-ietf-rtcweb-data-protocol-07

[6] Stream Control Transmission Protocol, RFC 4960, R. Stewart, September 2007, http://tools.ietf.org/html/rfc4960

[7] Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP), RFC 5764, D. McGrew, E. Rescorla, May 2010, https://tools.ietf.org/html/rfc5764

[8] C. Jennings, A. Narayanan, D. Burnett, and A. Bergkvist, "WebRTC 1.0: Real-time communication between browsers," W3C, W3C Editor's Draft, Aug. 2014, http://dev.w3.org/2011/webrtc/editor/archives/20140704/webrtc.html [retrieved: Aug., 2014].

[9] http://www.ietf.org/id/draft-ietf-rtcweb-security-arch-10.txt

[10] The Message Session Relay Protocol (MSRP), RFC 4975, B. Campbell, R. Mahy, C. Jennings, September 2007, https://tools.ietf.org/html/rfc4975

[11] OpenIMSCore Project Website, http://openimscore.org

[12] "Overview of WONDER Classes", https://github.com/hypercomm/wonder/wiki/Overview-of-WONDER-Classes

[13] WONDER GitHub repository, https://github.com/hypercomm/wonder

[14] WONDER Tutorials - How to Develop a WONDER App, https://github.com/hypercomm/wonder/wiki/How-to-Develop-a-WONDER-App

[15] WONDER Tutorials - How to Develop a Messaging Stub, https://github.com/hypercomm/wonder/wiki/How-to-Develop-a-Messaging-Stub

[16] WONDER Tutorials - How to setup your own WONDER environment, https://github.com/hypercomm/wonder/wiki/How-to-Develop-a-Messaging-Stub

[17] WONDER Live Demos, https://github.com/hypercomm/wonder/wiki/Live-Demos

[18] WONDER Project Site, http://hypercomm.github.io/wonder/

[19] Object RTC (ORTC) API for WebRTC, Draft Community Group Report, Robin Raymond, Bernard Aboba, Justin Uberti, 20 August 2014, http://ortc.org/wp-content/uploads/2014/08/ortc.html

[20] reTHINK project Site, http://rethink-project.eu/