WebRTC Signaling Protocol (WSP)

Gijsbert van der Linden, ViePlus
On behalf of the OpenBeeldZorg initiative
4-3-2015 version 1.0

Contents

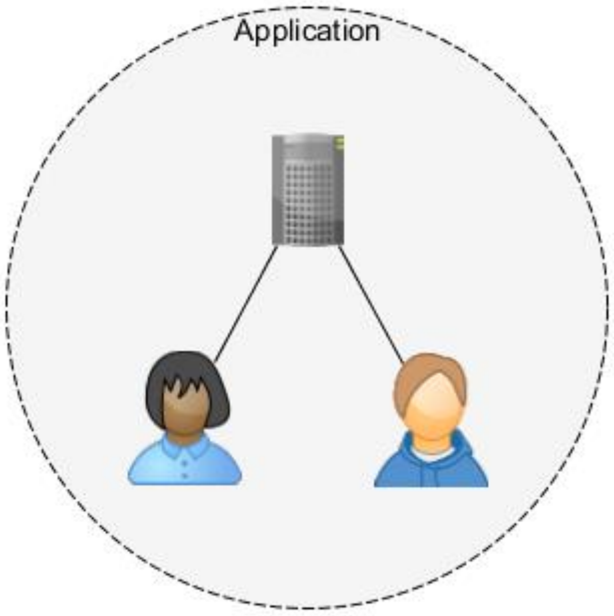
Introduction

Video communication can play an important role in the area of social care, supporting elderly people to keep on living in their own homes for a longer period, allowing handicapped people to live more independently, etc. Currently many different applications and protocols are used for video communication (Skype, Facetime, etc.). But there is no way to communicate between these applications. If for example a user uses Facetime on an iPad, there is no way to communicate with another user using an Android device or a PC. Especially in the area of Social Care this is leading to frustrations and delays in the adaptation of video communication.

Scope

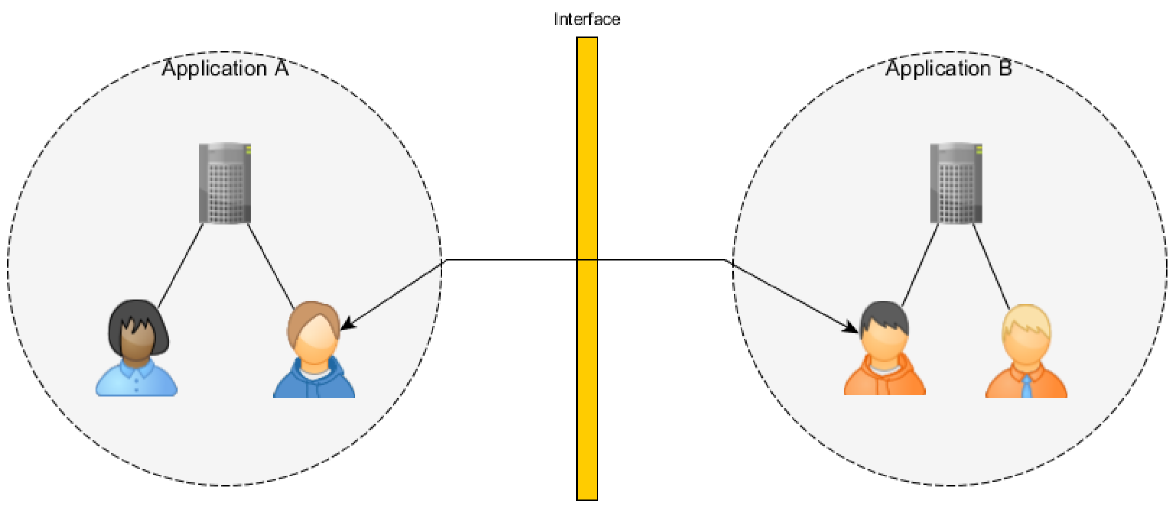
Web enabling applications is a trend avoiding dependencies on specific hardware and operating systems, by running applications within a browser. WebRTC is a new extension to HTTP, the language used in the browsers, allowing among others the creation of video communication applications running within a browser, without requiring any other software installed in the background. It is gaining strong support from software developers and is currently supported by browsers like Chrome and Firefox.

WebRTC is an API (Application Program Interface). That means that developers can create video communication applications with relatively little effort, allowing users of the same server / website to communicate with each other. These video communication applications will run in any environment, as long as the application runs in a browser supporting WebRTC or the application makes use of WebRTC natively.



The WebRTC standard however does not prescribe how different WebRTC applications communicate with each other. As a result the users of one WebRTC application cannot normally communicate with the users of another WebRTC application.

The standard described in this document augments the WebRTC standard, allowing users of one WebRTC application to setup video communication connections to users of another WebRTC application.



The protocol used to actually exchange voice and video streams between two users has been defined in the WebRTC standard. What needs to be added to that standard is a way to find the other user and to setup a WebRTC conversation with that user (the so-called signaling protocol).

A user of one WebRTC based video communication system should be able to setup a video call with a user of any other WebRTC based video communication system supporting this signaling protocol.

The protocol has been defined in such a way that it could be created, validated and published in a relatively short period (less than 6 month). For developers of WebRTC based video systems it is easy to implement (with an effort of typically less than 2 man-weeks), given a WebRTC based application has already been developed.

This document describes the standard for this WebRTC Signaling Protocol. The standard is fully open and can be downloaded from the OpenBeeldZorg website (http://www.openbeeldzorg.nl/).

Architecture of the protocol

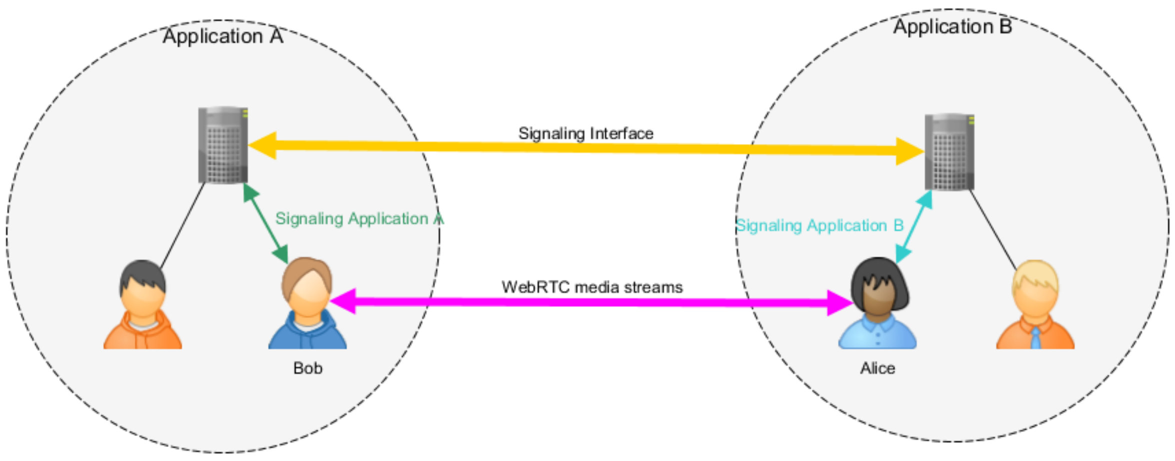
To setup a WebRTC call between a user in system A (Bob) with a user in system B (Alice) the two parties need to exchange information about the setup of the WebRTC connection before that connection can be established. Several options exist to handle this signaling of the WebRTC sessions among different WebRTC applications. Options evaluated but not chosen include the use of SIP (including SIP over WebSockets) and XMPP. This standard uses JSON objects to be initially sent across a WebSocket connection between the parties involved. The reasons for this choice include:

- A simple solution to a simple problem
- Quick to define
- Easy to implement
- Easy to extend when needed

Once the WebRTC connection is established the signaling is handled by a WebRTC data cannel.

This protocol is hereafter referred to as WSP (WebRTC Signaling Protocol, to be pronounced as Wesp).

The exchange of the signaling information will initially be handled through the servers involved with each WebRTC application. User Bob will communicate the required setup information to the server hosting application A. This server will setup a bi-directional connection with the server hosting application B. The server hosting application B communicates the setup information with its user Alice. The communication between the servers will use JSON objects to be sent across a secure WebSocket connection. The way user Bob communicates the setup information with the server hosting his application A and the way user Alice communicates the setup information with the server hosting her application B is outside the scope of this standard and can be freely chosen by the designers of these applications.



The reasons to handle the initial signaling information through the server of the sending user, instead of sending it from the browser of the sender to the server of the receiving user directly are:

- It allows the receiving server to authorize the sending server if there is a need to do so. This authorization could use any authorization mechanism for http(s) connections like certificates and IP addresses.
- It allows both servers to monitor the status of the connections and handle accordingly if needed.
- It involves only a small amount of data, so the overhead on the servers is minimal.

Notice that, although the initial signaling information is handled through the respective servers of the applications involved, the actual media streams, as handled by WebRTC, will of course flow directly between the client applications (unless hampered by firewall restrictions; in those cases a TURN server will pop-in, as prescribed by the WebRTC standard).

In order for Application A to know which server to contact to access a user of application B the address of a foreign user includes a userid part and a host part in the following form:

wsp:<userid>@<host>[:<port>]

The prefix *wsp* (WebRTC Signaling Protocol) is intended to be used in directories containing a mix of address types and indicates that the protocol defined in this document should be used to setup the connection. It must not be used inside the signaling bus described in this document, since that signaling bus is already dedicated to wsp messages (see below).

The <userid> part is the id of the user in the other system, according to the conventions of that other system.

The <host> part indicates the server to contact to setup the connection to user <userid>. This <host> part must either be the IP address of the server or must be defined in the DNS structure such that <host> resolves to the IP address of the server.

Optionally a port number could be added, if a port other than the default port for https (443) is to be used.

So in our example user Bob would identify user Alice using a uri like:

wsp:alice@wsp.application-b.com

For the hostname *wsp.application-b.com* any name could be used that resolves to the server handling the signaling on behalf of the users of application B. The server name does not have to start with *wsp*.

Notice that, since a secure (wss) connection is used, a certificate for this name must be present.

Notice that the server handling the signaling for an application could be the same server as hosting that application, or not, up to the designer of that application.

Once a WebRTC connection has been setup this way a data channel will be setup by the calling party using this WebRTC connection. The label of the data cannel must be *wsp*. Once this data channel has been setup between the clients, the WebSocket connection between the servers will be closed.

After establishing this WebRTC data channel it will handle any further signaling information between the clients directly. From this point on there is no dependency of the servers anymore.

The message structure

To exchange the signaling information between the applications involved a signaling bus is setup from the server hosting the application of the caller to the server hosting the application of the callee (identified in the host part of the uri of the callee).

Initially a WebSocket connection is used as the signaling bus. This WebSocket is created with the uri:

wss://<host>[:<port>]/wsp

and with *wsp-1.0* as the (sub-) protocol.

<host> is the name or IP address of the server handling the wsp requests, <port> is the optional port number this server is listening on (default 443).

So the call in JavaScript could look like:

```
var ws = new WebSocket('wss://wsp.server-b.com/wsp','wsp-1.0')
```

In case a new version of this protocol is created in the future the (sub-)protocol string *wsp-1.0* will be changed accordingly.

For each call a separate WebSocket must be setup.

The general format of the signaling messages consists of a JSON structure containing an array of one, two or three elements. The first (compulsory) element is the keyword of the message, the second (optional) element is the content of the message and the third (optional) element could contain options. The keyword is a (case-sensitive) string, the content can be anything representable in JSON, depending on the value of the keyword and the options could be an object with properties. So in JavaScript terms the WebSocket signaling messages could look like:

```
JSON.Stringify([ keyword , content, options])
```

The options element is reserved for future extension. In this version of the standard it is not actually used. So the message array must have a length of at most 2 or the third element must be "undefined".

The JSON representation of the signaling message is sent as a string (utf8) on the WebSocket connection.

If the signaling message received by a server does not comply with the specifications in this standard, the (WebSocket and/or WebRTC) connection must be closed immediately (so without first sending a bye message). This is especially the case if:

- The text received does not contain a valid JSON structure.
- The signaling message is not an array.
- The array does not contain either 1, 2 or 3 elements.
- The first element is not a string.
- The first element is not a valid keyword.
- The second element is neither not present, undefined nor an object.
- The third element is neither not present nor undefined.

Signaling Messages

The following messages are used on the WebSocket connection to setup calls between users of two applications.

Keyword	Content	Remarks

invite	{"callee":{
	"uri":"<id>@<host>"
	,"name":"<descriptive name>"
	},
	"caller":{
	"uri":"<id>@<host>"
	,"name":"<descriptive name>"
	}
	}
		The *name* attributes are optional.

ringing	undefined	Indicates that the receiving server is signaling the call to its client.
offer	SDP_Offer	Object as generated by the createOffer method of the RTCPeerConnection instance.
answer	SDP_Answer	Object as generated by the createAnswer method of the RTCPeerConnection instance.
icecandidate	ICE_Candidate	Object as returned by the onicecandidate event of the RTCPeerConnection instance.
bye	{"code":"<reason code>" ,"description":"<reason description>" }	Triggers the controlled closing of the WebSocket connection.

Responses

When closing the WebSocket connection the *bye* message is sent, with the reason for the close as the content of the message. The server receiving the *bye* message subsequently closes the WebSocket connection. Notice that the server receiving the bye message has the opportunity to check if no messages are in the pipeline of being transferred through the WebSocket connection, before actually closing the WebSocket connection.

The following reason codes and descriptions must be used in the *bye* message:

Code	**Reason**
101	Transferred
200	User ended call normally
310	General user error
311	User unknown
312	User not logged on
313	User set to not disturb
314	Call request timed out
315	User refused call

Notice that when things go wrong the WebSocket connection could also be closed without first sending a *bye* message. The servers participating in the WebSocket connection must treat this as an unexpected termination of the call without a known reason.

The WebRTC data channel message structure

When establishing the WebRTC connection a WebRTC data channel must be created by the party receiving the call, just before creating the WebRTC offer. This WebRTC data channel will be used as the signaling bus once it is available.

The WebRTC data channel is created using *wsp* as its label and using the default configuration options. As soon as the data channel has been established this data channel will be used to send and receive signaling messages between the parties from then on. Once that is the case, either server will send a *bye* message with reason code 101 on the WebSocket connection (notice that the clients must instruct the servers to do so of course). When this message is received by the other server, that server will ascertain that no messages are in the pipeline of being sent through the WebSocket connection (notice it will typically have to check this with its client). If there are messages already in the pipeline, these messages will still be handled through the WebSocket connection. Once the other server has verified that no messages are pending for the WebSocket connection anymore it will actually close that WebSocket connection.

Notice that as a result there might be a relatively small time-span where clients could receive signaling messages through either the WebSocket connection (via the servers) as well as through the WebRTC data channel (directly).

The WebRTC data channel exists for as long as the WebRTC connection exists.

The structure of the messages on the WebRTC data channel is the same as on the WebSocket connection. The following messages are supported:

Keyword	Content	Remarks
offer	SDP_Offer	Object as generated by the createOffer method of the RTCPeerConnection instance.
answer	SDP_Answer	Object as generated by the createAnswer method of the RTCPeerConnection instance.
icecandidate	ICE_Candidate	Object as returned by the onicecandidate event of the RTCPeerConnection instance.
bye	{"code":"<reason code>" ,"description":"<reason description>" }	Initiates the closing of the WebRTC connection.

Notice that the *invite* and *ringing* messages are not applicable at this stage, since the WebRTC connection has already been established.

Offer, *answer* and *icecandidate* messages could still be initiated even when the WebRTC connection and data channel have already been established (for example in case of renegotiations). In these cases these messages will be sent and received across the WebRTC data channel.

The *bye* keyword is used to signal the end of the connection. It can be sent by either party. Once it is sent or received the WebRTC connection will be closed.

If the WebRTC connection is closed without receiving the bye keyword the connection apparently was closed unintentionally. It that case the connection could be restored automatically. Automatically restoring an unintentionally closed WebRTC connection will involve a new WebSocket signaling connection. This is however outside the scope of this version of the standard.

The data channel could also be used to exchange other information between the calling parties, other than the standard WebRTC renegotiations through offer/answer and/or icecandidate messages. This is however outside the scope of the current version of the standard.

Message flow

The following diagram shows a typical message flow to setup a WebRTC session, once the WebSocket has been established:

Initiating server		**Receiving server**	
Actions	**Message →**	**← Message**	**Actions**
Create WebSocket			
	invite		
			Make the bell ring
		ringing	
Inform user			
			User accepts call
			Create WebRTC data channel Create offer Set local description
		offer	
Set remote description Create answer Set local description			
	answer		
			Set remote description
		icecandidate	
Add Ice candidate			
	icecandidate		
			Add Ice candidate
WebRTC connection established, users having a conversation			
Accept WebRTC data channel Send new messages through WebRTC data channel from now on			Accept WebRTC data channel Send new messages through WebRTC data channel from now on
	bye (101)	bye (101)	
Close WebSocket, continue with WebRTC data channel			Close WebSocket, continue with WebRTC data channel
WebRTC connection ended			
Either party could end the conversation			Either party could end the conversation
	bye (200)	bye (200)	
Close the WebRTC connection Inform user			Close the WebRTC connection Inform user

Correlation of messages

The table below describes in what states each message can be sent.

Message	
invite	Always the first message to be transmitted. Can be sent only once
ringing	Only to be sent after having received an invite and before sending an offer. Optional (might for example not be sent when callee is in autoanswer mode).
offer	Only to be sent after having received an invite. Can be sent multiple times.
answer	Only to be sent after having received an offer. Can be sent only once for each offer received.
icecandidate	Only to be sent after having sent or received an answer. Can be sent multiple times.
bye	Can be sent on the WebSocket connection at any stage after the WebSocket connection has been established. The *bye* message must always be the last message to be sent on the WebSocket connection. The webSocket connection must be closed only after receiving a *bye* message. Once the WebRTC data channel is available, a server must terminate the WebSocket connection by sending the *bye/tranferred* message on the WebSocket connection. It must only do so when it is informed that its client can send and receive messages on the WebRTC data channel. After sending this *bye/transferred* message the server sending this *bye/transferred* message must still listen for messages on the WebSocket connection, until the connection is closed by the other server. The server receiving the *bye/tranferred* message must close the WebSocket connection. It must only do so when it is informed that its client can send and receive messages on the WebRTC data channel and has assured that no messages are in the pipeline for the WebSocket connection anymore. Notice that both servers might send a *bye* message at about the same time. Whoever receives the *bye* message first will actually close the connection. Can be sent on the WebRTC data channel at any stage after the WebRTC connection has been established. The *bye* message must always be the last message to be sent on the WebRTC data channel connection. The WebRTC connection (including the data channel) must be closed after receiving the *bye* message. Can be sent by either party.

Credits

OpenBeeldZorg is an initiative of:

- Foppe Rauwerda (Beeldzorgadvies B.V.)
- Corné Eshuis (Carescreen B.V.)
- Maartje van Hees (ITimacy)
- Gijsbert van der Linden (ViePlus B.V.)

In the preparation and testing of the standard the following parties were involved:

- Bas Goossen (Mibida B.V.)
- Guido van Alphen & Karste de Vries (Stichting TriVici)
- Corné Eshuis (Carescreen B.V.)
- Gijsbert van der Linden (ViePlus B.V.)

The latest version of the standard can be downloaded from the OpenBeeldZorg website (http://www.openbeeldzorg.nl/).