SIPCORE Working Group	I.B.C. Baz Castillo
Internet-Draft	J.L.M. Luis Millan
Intended status: Standards Track	XtraTelecom S.A.
Expires: May 27, 2012	V.P. Pascual
	Acme Packet
	November 24, 2011

The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)
draft-ibc-sipcore-sip-websocket-00

Abstract

This document specifies a WebSocket Sub-Protocol for a new transport in SIP (Session Initiation Protocol). The WebSocket protocol enables two-way realtime communication between clients and servers.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet- Drafts is at http://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
This Internet-Draft will expire on May 27, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

This specification defines a new WebSocket Sub-Protocol for transporting SIP messages between a WebSocket client and server, a new transport for the SIP protocol and procedures for SIP servers when bridging WebSocket and other SIP transports.
This specification is focused on integrating the SIP protocol within client applications running a WebSocket stack. Other aspects such as the usage of WebSocket as a transport between SIP servers are not fully covered by this specification.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. The WebSocket Protocol

WebSocket protocol [I-D.ietf-hybi-thewebsocketprotocol] is a transport layer on top of TCP in which both client and server exchange message units in both directions. The protocol defines a connection handshake, WebSocket Sub-Protocol and extensions negotiation, a frame format for sending application and control data, a masking mechanism, and status codes for indicating disconnection causes.
The WebSocket connection handshake is based on HTTP [RFC2616] protocol by means of a specific HTTP GET request sent by the client, typically a web browser, which is answered by the server (if the negotiation succeeded) with HTTP 101 status code. This handshake procedure is designed to reuse the existing HTTP infrastructure. During the connection handshake, client and server agree in the application protocol to use on top of the WebSocket transport. Such application protocol (also known as the "WebSocket Sub-Protocol") defines the format and semantics of the messages exchanged between both endpoints. The WebSocket Sub-Protocol to be used is up to the application developer. It may be a custom protocol or a standarized one (as the WebSocket SIP Sub-Protocol proposed in this document). Once the HTTP 101 response is processed both client and server reuse the existing TCP connection for sending application messages and control frames to each other in a persistent way.
WebSocket defines message units as application data exchange for communication endpoints, becoming a message boundary transport layer.

These messages can contain UTF-8 text or binary data, and can be splitted into various WebSocket text/binary frames. However, the WebSocket API *[WS-API]* for web browsers just includes JavaScript callbacks that are invoked upon receipt of an entire message, regardless it has been received in a single or multiple WebSocket frames.

4. The WebSocket SIP Sub-Protocol

The term WebSocket Sub-Protocol refers to the application-level protocol layered over a WebSocket connection. This document specifies the WebSocket SIP Sub-Protocol for carrying SIP requests and responses through a WebSocket connection.
The WebSocket client and server need to agree on this protocol during the WebSocket handshake procedure as defined in section 1.3 of [I-D.ietf-hybi-thewebsocketprotocol]. The client MUST include the value "sip" in the Sec-WebSocket-Protocol header in its handshake request. The 101 reply from the WebSocket server MUST contain "sip" in its own Sec-WebSocket-Protocol header.

```
  GET / HTTP/1.1
  Host: sip-ws.example.com
  Upgrade: websocket
  Connection: Upgrade
  Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
  Origin: http://www.example.com
  Sec-WebSocket-Protocol: sip
  Sec-WebSocket-Version: 13
```

Below is an example of the WebSocket handshake in which the client requests SIP Sub-Protocol support from the server:

```
  HTTP/1.1 101 Switching Protocols
  Upgrade: websocket
  Connection: Upgrade
  Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
  Sec-WebSocket-Protocol: sip
```

The handshake response from the server supporting the WebSocket SIP Sub-Protocol would look like:
Once the negotiation is done, the WebSocket connection is established with SIP as the WebSocket Sub-Protocol. The WebSocket messages to be transmitted over this connection MUST conform to the established signaling protocol.
WebSocket messages are carried on top of WebSocket UTF-8 text frames or binary frames. SIP protocol [RFC3261] allows both text and binary bodies in SIP messages. Therefore a client and server implementing the WebSocket SIP Sub-Protocol MUST accept both WebSocket text and binary frames.

Each SIP message MUST be carried within a single WebSocket message and MUST be a complete SIP message, so a Content-Length header field is not mandatory. Sending more than one SIP message within a single WebSocket message is not allowed, neither sending an incomplete SIP message.

> *This makes parsing of SIP messages easier on client side (typically web-based applications with an strict and simple API for receiving WebSocket messages). There is no need to establish boundaries (using Content-Length headers) between different messages. Same advantage is present in other message-based SIP transports as UDP or SCTP *[RFC4168]*.

5. SIP WebSocket Transport

WebSocket [I-D.ietf-hybi-thewebsocketprotocol] is a reliable protocol and therefore the WebSocket sub-protocol for a SIP transport defined by this document is also a reliable transport. Thus, client and server transactions using WebSocket transport MUST follow the procedures and timer values for reliable transports as defined in [RFC3261].

5.1. Via Transport Parameter

Via header fields carry the transport protocol identifier. This document defines the value "WS" to be used for requests over plain WebSocket protocol and "WSS" for requests over secure WebSocket protocol (in which the WebSocket connection is established on top of TLS [RFC5246] over TCP transport).

```
  transport  =  "UDP" / "TCP" / "TLS" / "SCTP" / "WS" / "WSS"
                / other-transport
```

The updated augmented BNF (Backus-Naur Form) [RFC5234] for this parameter is the following:

5.2. SIP URI Transport Parameter

This document defines the value "ws" as the transport parameter value for a SIP URI [RFC3986] to be contacted using WebSocket protocol as transport.

```
  transport-param  =  "transport="
                      ( "udp" / "tcp" / "sctp" / "tls" / "sctp"
                      / "ws"
                      / other-transport )
```

The updated augmented BNF (Backus-Naur Form) [RFC5234] for this parameter is the following:

5.3. Sending Responses

The SIP server transport uses the value of the top Via header field in order to determine where to send a response. If the "sent-protocol" is "WS" or "WSS" the response MUST be sent using the existing WebSocket connection to the source of the original request, if that connection is still open. This requires the server transport to maintain an association between server transactions and transport connections. If that connection is no longer open, the server MUST NOT attempt to open a WebSocket connection to the Via "sent-by"/"received"/"rport".

*This is due to the nature of the WebSocket protocol in which just the WebSocket client can establish a connection with the WebSocket server. A WebSocket client does not listen for incoming connections.

6. Outbound and GRUU Usage

WebSocket requires the client to open a TCP connection with the server and perform the WebSocket handshake. A WebSocket client does not listen for incoming connections so it can only receive SIP requests from the WebSocket server it is connected to. WebSocket clients may use either public or private addressing but it is expected that many of them will run the latter. Unfortunately, some implementations may not have the ability to discover the local transport address which the WebSocket connection is originated from (e.g. a JavaScript stack within a web browser).
Therefore clients and servers implementing SIP over the WebSocket transport MUST implement the Outbound mechanism [RFC5626], being this the most suitable solution for SIP clients behind Network Address Translation (NAT) using reliable transports for contacting SIP servers. A client implementing SIP over the WebSocket transport SHOULD also implement GRUU [RFC5627]. The registrar responsible for the registration of SIP clients using the WebSocket transport SHOULD implement GRUU as well.

*If a REFER request is sent to a SIP User Agent indicating the Contact URI of a WebSocket client as the target in the Refer-To header field, such a URI will be reachable by the SIP UA just in the case it is a globally routable URI obtained from a SIP registrar implementing GRUU.

Both Outbound and GRUU require the client to indicate a Uniform Resource Name (URN) in the "+sip.instance" parameter of the Contact header during the registration. The client device is responsible for getting such a constant and unique value.

*In the case of web browsers it is hard to get a URN value from the browser itself. This specification suggests that value is generated according to [RFC5626] section 4.1 by the web application running in the browser the first time it loads the web page, and then it is stored as a Cookie [RFC6265] within the browser data and loaded every time the same web page is visited. The application developer could choose any other mechanism which accomplishes the requirements of a URN.

7. Locating a SIP Server

SIP entities follow normal SIP procedures in [RFC3263] to discover a SIP server. This specification defines the NAPTR service value "SIP+D2W" for servers that support plain WebSocket transport and "SIPS+D2W" for servers that support secure WebSocket transport.
A SIP entity using the WebSocket transport SHOULD perform procedures in [RFC3263] for the given WebSocket URI it will connect to. If the WebSocket URI has "wss" schema the SIP entity MUST only consider "SIPS+D2W" resource records. If the WebSocket URI does not contain a domain in the host part or does include a port, the SIP entity MUST follow procedures in [I-D.ietf-hybi-thewebsocketprotocol] section 3 instead.

*Unfortunately the JavaScript stack running in web browsers cannot perform DNS NAPTR/SRV queries, neither the WebSocket stack running in web browsers can do it. Thus, a WebSocket URI given within a web application needs to have a numeric network address or a hostname with associated DNS A/AAAA resource record(s) in its host part.

8. WebSocket Client Usage

The WebSocket connection MUST be established in order to allow the client application to send and receive SIP requests.

*Based on local policy, this might occur once the JavaScript SIP application has been downloaded from the web server, or when the SIP user using the web browser application registers itself to a SIP registrar (assuming that SIP requests cannot be sent or received before then).

In case the client application decides to close the WebSocket connection (for example when performing "logout" in a web application) it is recommended to remove the existing SIP registration binding (if present) by means of a REGISTER with expiration value of 0 and the associated "+sip.instance" Contact header parameter as per [RFC5626].

8.1. WebSocket Disconnection

In some circumstances the WebSocket connection could be terminated by the WebSocket server (for example when the server is restarted). If the client application wants to become reachable again it SHOULD reconnect to the WebSocket server and perform a new SIP registration with same "+sip.instance" and "reg-id" Contact header parameters (as stated in [RFC5626]).

9. WebSocket Server Usage

How a SIP server authorizes WebSocket connection attemps from clients is out of the scope of this specification. However some informational guidelines are provided in Section 11. Once the WebSocket SIP Sub-Protocol is agreed, both client and server can send SIP messages to each other.

9.1. SIP Proxy Considerations

When a SIP proxy bridges WebSocket and any other SIP transport (including WebSocket transport) it MUST perform Loose Routing as specified in [RFC3261]. Otherwise in-dialog requests would fail since WebSocket clients cannot contact destinations other than their WebSocket server, and non-WebSocket SIP entities cannot establish a connection to WebSocket clients. It is also recommended that SIP proxy implementations use double Record-Route techniques (as specified in [RFC5658]).
In the same way, if the SIP proxy implementing the WebSocket server behaves as an outbound proxy for REGISTER requests, it MUST add a Path header field as described in [RFC3327]. Otherwise the WebSocket client would never receive incoming requests from the SIP registrar server after the lookup procedures in the SIP location service.

10. Connection Keep Alive

It is recommended that the WebSocket client or server keeps the WebSocket connection open by sending periodic WebSocket Ping frames as described in [I-D.ietf-hybi-thewebsocketprotocol] section 5.5.2. The decision for a WebSocket endpoint to maintain, or not, the connection over time is out of scope of this document.
The client application MAY also use Network Address Translation (NAT) keep-alive mechanisms defined for the SIP protocol, such as the CRLF Keep-Alive Technique mechanism described in [RFC5626] section 3.5.1. Therefore, a SIP server implementing the WebSocket transport MUST support the CRLF Keep-Alive Technique.

11. Authentication

Prior to sending SIP requests, the WebSocket client implementing the SIP protocol connects to the WebSocket server and performs the connection handshake. As described in Section 3 the handshake procedure involves an HTTP GET request replied with HTTP 101 status code by the server.
In order to authorize the WebSocket connection the server MAY inspect the Cookie [RFC6265] header in the HTTP GET request (if present). In

case of web applications the value of such a Cookie is typically provided by the web server once the user has authenticated itself against the web application by following any of the multiple existing mechanisms. As an alternative method, the WebSocket server could request Digest [RFC2617] authentication by replying a HTTP 401 status code. The WebSocket protocol [I-D.ietf-hybi-thewebsocketprotocol] covers this usage in section 4.1:

> *If the status code received from the server is not 101, the client handles the response per HTTP [RFC2616] procedures, in particular the client might perform authentication if it receives 401 status code.

Regardless the WebSocket server requires authentication during the WebSocket handshake or not, authentication MAY be requested at SIP protocol level. Therefore a SIP client using the WebSocket transport MUST implement Digest [RFC2617] authentication as stated in [RFC3261].

12. Examples

12.1. Registration

```
Alice    (SIP WSS)    proxy.atlanta.com
|                             |
|REGISTER F1                  |
|---------------------------->|
|200 OK F2                    |
|<----------------------------|
|                             |
```

Alice loads a web page using her web browser and retrieves a JavaScript code implementing the WebSocket SIP Sub-Protocol defined in this document. The JavaScript code obtained from the web server establishes a secure WebSocket connection with a SIP proxy/registrar at proxy.atlanta.com. Upon WebSocket connection, Alice constructs and sends a SIP REGISTER by requesting Outbound and GRUU support. Since the JavaScript stack in a browser has no way to determine the local address from which the WebSocket connection is made, this implementation uses anonymous.invalid in Via sent-by for every SIP requests and anonymous.invalid as URI hostpart in the Contact header of the initial REGISTER request.
Message details (authentication and SDP bodies are omitted for simplicity):

```
F1 REGISTER  Alice -> proxy.atlanta.com (transport WSS)

REGISTER sip:proxy.atlanta.com SIP/2.0
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bKasudf
From: sip:alice@atlanta.com;tag=65bnmj.34asd
To: sip:alice@atlanta.com
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Supported: path, outbound, gruu
Route: <sip:proxy.atlanta.com:443;transport=ws;lr>
Contact: <sip:alice@anonymous.invalid;transport=ws>
  ;reg-id=1
  ;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"


F2 200 OK  proxy.atlanta.com -> Alice (transport WSS)

SIP/2.0 200 OK
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bKasudf
From: sip:alice@atlanta.com;tag=65bnmj.34asd
To: sip:alice@atlanta.com;tag=12isjljn8
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Supported: outbound, gruu
Contact: <sip:alice@anonymous.invalid;transport=ws>
  ;reg-id=1
  ;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
  ;pub-gruu="sip:alice@atlanta.com;gr=urn:uuid:f81-7dec-14a06cf1"
  ;temp-gruu="sip:87ash54=3dd.98a@atlanta.com;gr"
  ;expires=3600
```

12.2. INVITE dialog through a proxy

```
Alice    (SIP WSS)    proxy.atlanta.com    (SIP UDP)       Bob
|                             |                             |
|INVITE F1                    |                             |
|---------------------------->|                             |
|100 Trying F2                |                             |
|<----------------------------|                             |
|                             |INVITE F3                    |
|                             |---------------------------->|
|                             |200 OK F4                    |
|                             |<----------------------------|
|200 OK F5                    |                             |
|<----------------------------|                             |
|                             |                             |
|ACK F6                       |                             |
|---------------------------->|                             |
|                             |ACK F7                       |
|                             |---------------------------->|
|                             |                             |
|                    Both Way RTP Media                     |
|<=========================================================>|
|                             |                             |
|                             |BYE F8                       |
|                             |<----------------------------|
|BYE F9                       |                             |
|<----------------------------|                             |
|200 OK F10                   |                             |
|---------------------------->|                             |
|                             |200 OK F11                   |
|                             |---------------------------->|
|                             |                             |
```

In the same scenario Alice places a call to Bob's AoR by using the public GRUU retrieved from the registrar as Contact URI of the INVITE. The WebSocket SIP server at proxy.atlanta.com acts as a SIP proxy routing the INVITE to the UDP location of Bob, who answers the call and terminates it later.
Message details (authentication and SDP bodies are omitted for simplicity):

```
F1 INVITE  Alice -> proxy.atlanta.com (transport WSS)

INVITE sip:bob@atlanta.com SIP/2.0
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bK56sdasks
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 70
Supported: path, outbound, gruu
Route: <sip:proxy.atlanta.com:443;transport=ws;lr>
Contact: <sip:alice@atlanta.com
 ;gr=urn:uuid:f81-7dec-14a06cf1;ob>"
Content-Type: application/sdp


F2 100 Trying  proxy.atlanta.com -> Alice (transport WSS)

SIP/2.0 100 Trying
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bK56sdasks
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com
Call-ID: asidkj3ss
CSeq: 1 INVITE


F3 INVITE  proxy.atlanta.com -> Bob (transport UDP)

INVITE sip:bob@203.0.113.22:5060 SIP/2.0
Via: SIP/2.0/UDP proxy.atlanta.com;branch=z9hG4bKhjhjqw32c
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.atlanta.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.atlanta.com:443;transport=ws;lr>
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 69
Supported: path, outbound, gruu
Contact: <sip:alice@atlanta.com
  ;gr=urn:uuid:f81-7dec-14a06cf1;ob>"
Content-Type: application/sdp


F4 200 OK  Bob -> proxy.atlanta.com (transport UDP)

SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.atlanta.com;branch=z9hG4bKhjhjqw32c
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.atlanta.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.atlanta.com:443;transport=ws;lr>
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 69
Contact: <sip:bob@203.0.113.22:5060;transport=udp>
Content-Type: application/sdp


F5 200 OK  proxy.atlanta.com -> Alice (transport WSS)

SIP/2.0 200 OK
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.atlanta.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.atlanta.com:443;transport=ws;lr>
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 69
Contact: <sip:bob@203.0.113.22:5060;transport=udp>
Content-Type: application/sdp


F6 ACK  Alice -> proxy.atlanta.com (transport WSS)

ACK sip:bob@203.0.113.22:5060;transport=udp SIP/2.0
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bKhgqqp090
Route: <sip:h7kjh12s@proxy.atlanta.com:443;transport=ws;lr>,
  <sip:proxy.atlanta.com;transport=udp;lr>,
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 ACK
Max-Forwards: 70


F7 ACK  proxy.atlanta.com -> Bob (transport UDP)

ACK sip:bob@203.0.113.22:5060;transport=udp SIP/2.0
Via: SIP/2.0/UDP proxy.atlanta.com;branch=z9hG4bKhwpoc80zzx
Via: SIP/2.0/WSS anonymous.invalid;branch=z9hG4bKhgqqp090
From: sip:alice@atlanta.com;tag=asdyka899
To: sip:bob@atlanta.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 ACK
Max-Forwards: 69
```

```
F8 BYE  Bob -> proxy.atlanta.com (transport UDP)

BYE sip:alice@atlanta.com;gr=urn:uuid:f81-7dec-14a06cf1;ob SIP/2.0
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
Route: <sip:proxy.atlanta.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.atlanta.com:443;transport=ws;lr>
From: sip:bob@atlanta.com;tag=bmqkjhsd
To: sip:alice@atlanta.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
Max-Forwards: 70


F9 BYE  proxy.atlanta.com -> Alice (transport WSS)

BYE sip:alice@atlanta.com;gr=urn:uuid:f81-7dec-14a06cf1;ob SIP/2.0
Via: SIP/2.0/WSS proxy.atlanta.com:443;branch=z9hG4bKmma01m3r5
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@atlanta.com;tag=bmqkjhsd
To: sip:alice@atlanta.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
Max-Forwards: 69


F10 200 OK  Alice -> proxy.atlanta.com (transport WSS)

SIP/2.0 200 OK
Via: SIP/2.0/WSS proxy.atlanta.com:443;branch=z9hG4bKmma01m3r5
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@atlanta.com;tag=bmqkjhsd
To: sip:alice@atlanta.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE


F11 200 OK  proxy.atlanta.com -> Bob (transport UDP)

SIP/2.0 200 OK
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@atlanta.com;tag=bmqkjhsd
To: sip:alice@atlanta.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
```

13. Security Considerations

13.1. Secure WebSocket Connection

It is recommended to protect the privacy of the SIP traffic through the WebSocket communication by using a secure WebSocket connection (tunneled over TLS [RFC5246]). For this, the client application MUST be provided with a secure "wss" WebSocket URI.

13.2. WebSocket Topology Hiding

RFC 3261 *[RFC3261]* section 18.2.1 "Receiving Requests" states the following:

> *When the server transport receives a request over any transport, it MUST examine the value of the "sent-by" parameter in the top Via header field value. If the host portion of the "sent-by" parameter contains a domain name, or if it contains an IP address that differs from the packet source address, the server MUST add a "received" parameter to that Via header field value. This parameter MUST contain the source address from which the packet was received.

The requirement of adding the "received" parameter does not fit well into WebSocket protocol nature. The WebSocket handshake connection reuses existing HTTP infrastructure in which there could be certain number of HTTP proxies and/or TCP load balancers between the client and the WebSocket server, so the source IP the server would write into the Via "received" parameter would be the IP of the HTTP/TCP intermediary in front of it. This would reveal sensitive information about the internal topology of the provider network to the WebSocket client. Thus, given the fact that SIP responses can only be sent over the existing WebSocket connection, the meaning of the Via "received" parameter added by the server is of little use. Therefore, in order to allow hiding possible sensitive information about the provider infrastructure, this specification relaxes the requirement in RFC 3261 *[RFC3261]* section 18.2.1 "Receiving Requests" by stating that a WebSocket server receiving a SIP request from a WebSocket client MAY choose not to add the Via "received" parameter nor honor the Via "rport" [RFC3581] parameter. A SIP client implementing the WebSocket transport MUST be ready to receive SIP responses in which the topmost Via header field does not contain the "received" and "rport" parameters.

14. IANA Considerations

14.1. Registration of the WebSocket SIP Sub-Protocol

This specification requests IANA to create the WebSocket SIP Sub-Protocol in the registry of WebSocket sub-protocols with the following data:

Subprotocol Identifier: sip

Subprotocol Common Name: SIP over WebSocket

Subprotocol Definition: TBD, it should point to this document

14.2. Registration of new Via transports

This specification registers two new transport identifiers for Via headers:

WS: MUST be used when constructing a SIP request to be sent over a plain WebSocket connection.

WSS: MUST be used when constructing a SIP request to be sent over a secure WebSocket connection (tunneled over TLS [RFC5246]).

14.3. Registration of new SIP URI transport

This specification registers a new value for the "transport" parameter in a SIP URI:

ws: Identifies a SIP URI to be contacted using a WebSocket connection.

14.4. Registration of new NAPTR service field values

Services Field	Protocol	Reference
SIP+D2W	WS	TBD: this document
SIPS+D2W	WSS	TBD: this document

This document defines two new NAPTR service field values (SIP+D2W and SIPS+D2W) and requests IANA to register these values under the "Registry for the SIP SRV Resource Record Services Field". The resulting entries are as follows:

15. Acknowledgements

Special thanks to the following people who participated in discussions on the SIPCORE and RTCWEB WG mailing lists and contributed ideas and/or provided detailed reviews (the list is likely to be incomplete): Hadriel Kaplan, Paul Kyzivat, Ranjit Avasarala.

Special thanks also to Saul Ibarra Corretgé for his detailed review and provided suggestions.

16. References

16.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2617]	Franks, J., Hallam-Baker, P.M., Hostetler, J.L., Lawrence, S.D., Leach, P.J., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
[RFC5234]	Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
[RFC3261]	Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
[RFC3263]	Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.
[RFC5626]	Jennings, C., Mahy, R. and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.
[RFC5627]	Rosenberg, J., "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)", RFC 5627, October 2009.
[I-D.ietf-hybi-thewebsocketprotocol]	Fette, I and A Melnikov, "The WebSocket protocol", Internet-Draft draft-ietf-hybi-thewebsocketprotocol-17, September 2011.

16.2. Informative References

[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[RFC3986]	Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
[RFC4168]	

	Rosenberg, J., Schulzrinne, H. and G. Camarillo, "The Stream Control Transmission Protocol (SCTP) as a Transport for the Session Initiation Protocol (SIP)", RFC 4168, October 2005.
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
[RFC3327]	Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.
[RFC3581]	Rosenberg, J. and H. Schulzrinne, "An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing", RFC 3581, August 2003.
[RFC5658]	Froment, T., Lebel, C. and B. Bonnaerens, "Addressing Record-Route Issues in the Session Initiation Protocol (SIP)", RFC 5658, October 2009.
[RFC6265]	Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.
[WS-API]	Hickson, I., "The Web Sockets API", September 2010.

Authors' Addresses

Inaki Baz Castillo Baz Castillo XtraTelecom S.A. Barakaldo, Basque Country Spain EMail: ibc@aliax.net

Jose Luis Millan Luis Millan XtraTelecom S.A.
Bilbao, Basque Country Spain EMail: jmillan@aliax.net

Victor Pascual Pascual Acme Packet Anabel Segura 10 Madrid, Madrid 28108 Spain EMail: vpascual@acmepacket.com