SIPCORE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2012

I. Baz Castillo
J. Millan Villegas
Consultant
V. Pascual
Acme Packet
June 27, 2012

The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP) draft-ietf-sipcore-sip-websocket-01

Abstract

The WebSocket protocol enables two-way realtime communication between clients and servers. This document specifies a new WebSocket sub-protocol as a reliable transport mechanism between SIP (Session Initiation Protocol) entities and enables usage of the SIP protocol in new scenarios.

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 December 29, 2012.

Copyright Notice

Copyright (c) 2012 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 3
2. Terminology 3
2.1. Definitions 3
3. The WebSocket Protocol 3
4. The WebSocket SIP Sub-Protocol 4
4.1. Handshake 4
4.2. SIP encoding 5
5. SIP WebSocket Transport 5
5.1. General 5
5.2. Updates to RFC 3261 6
5.2.1. Via Transport Parameter 6
5.2.2. SIP URI Transport Parameter 6
5.3. Locating a SIP Server 6
6. Connection Keep Alive 7
7. Authentication 7
8. Examples 8
8.1. Registration 8
8.2. INVITE dialog through a proxy 10
9. Security Considerations 14
9.1. Secure WebSocket Connection 14
9.2. Usage of SIPS Scheme 14
10. IANA Considerations 14
10.1. Registration of the WebSocket SIP Sub-Protocol 14
10.2. Registration of new Via transports 14
10.3. Registration of new SIP URI transport 15
10.4. Registration of new NAPTR service field values 15
11. Acknowledgements 15
12. References 15
12.1. Normative References 15
12.2. Informative References 16
Appendix A. Implementation Guidelines 17
A.1. SIP WebSocket Client Considerations 18
A.2. SIP WebSocket Server Considerations 18
Appendix B. HTTP Topology Hiding 19
Authors' Addresses 19

1. Introduction

The WebSocket [RFC6455] protocol enables messages exchange between clients and servers on top of a persistent TCP connection (optionally secured with TLS [RFC5246]). The initial protocol handshake makes use of HTTP [RFC2616] semantics, allowing the WebSocket protocol to reuse existing HTTP infrastructure.

Modern web browsers include a WebSocket client stack complying with The WebSocket API [WS-API] as specified by the W3C. It is expected that other client applications (those running in personal computers and devices such as smartphones) will also run a WebSocket client stack. The specification in this document enables usage of the SIP protocol in those new scenarios.

This specification defines a new WebSocket sub-protocol (section 1.9 in [RFC6455]) for transporting SIP messages between a WebSocket client and server, a new reliable and message boundary transport for the SIP protocol, new DNS NAPTR [RFC3403] service values and procedures for SIP entities implementing the WebSocket transport. Media transport is out of the scope of this document.

2. Terminology

All diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

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].

2.1. Definitions

SIP WebSocket Client: A SIP entity capable of opening outbound connections with WebSocket servers and speaking the WebSocket SIP Sub-Protocol as defined by this document.

SIP WebSocket Server: A SIP entity capable of listening for inbound connections from WebSocket clients and speaking the WebSocket SIP Sub-Protocol as defined by this document.

3. The WebSocket Protocol

This section is non-normative.

WebSocket protocol [RFC6455] is a transport layer on top of TCP (optionally secured with TLS [RFC5246]) 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 method with Upgrade request sent by the client which is answered by the server (if the negotiation succeeded) with HTTP 101 status code. Once the handshake is done the connection upgrades from HTTP to the WebSocket protocol. 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. 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 underlying TCP connection for sending WebSocket 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 split into various WebSocket text/binary frames.

> However, the WebSocket API [WS-API] for web browsers just includes callbacks that are invoked upon receipt of an entire message, regardless of whether it was 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 on top of a WebSocket connection. This document specifies the WebSocket SIP Sub-Protocol for carrying SIP requests and responses through a WebSocket connection.

4.1. Handshake

The SIP WebSocket Client and SIP WebSocket Server need to agree on the WebSocket SIP Sub-Protocol during the WebSocket handshake procedure as defined in section 1.3 of [RFC6455]. The client MUST include the value "sip" in the Sec-WebSocket-Protocol header in its

handshake request. The 101 reply from the server MUST contain "sip" in its corresponding Sec-WebSocket-Protocol header.

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

```
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
```

The handshake response from the server supporting the WebSocket SIP Sub-Protocol would look as follows:

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

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 application protocol.

4.2. SIP encoding

WebSocket messages are carried on top of WebSocket UTF-8 text frames or binary frames. The SIP protocol [RFC3261] allows both text and binary bodies in SIP messages. Therefore SIP WebSocket Clients and SIP WebSocket Servers MUST accept both WebSocket text and binary frames.

5. SIP WebSocket Transport

5.1. General

WebSocket [RFC6455] 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].

Each complete SIP message MUST be carried within a single WebSocket message, and a WebSocket message MUST NOT contain more than one SIP message. Therefore the usage of the Content-Length header field is optional.

> This makes parsing of SIP messages easier on client side (typically web-based applications with a 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 such as UDP or SCTP [RFC4168].

5.2. Updates to RFC 3261

5.2.1. 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 using TLS [RFC5246] with TCP transport).

The updated RFC 3261 augmented BNF (Backus-Naur Form) [RFC5234] for this parameter reads as follows:

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

5.2.2. 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.

The updated RFC 3261 augmented BNF (Backus-Naur Form) for this parameter reads as follows:

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

5.3. Locating a SIP Server

RFC 3263 [RFC3263] specifies the procedures which should be followed by SIP entities for locating SIP servers. This specification defines the NAPTR service value "SIP+D2W" for SIP WebSocket Servers that support plain WebSocket transport and "SIPS+D2W" for SIP WebSocket

Servers that support secure WebSocket transport.

> Unfortunately neither JavaScript stacks nor WebSocket stacks running in current web browsers are capable of performing DNS NAPTR/SRV queries.

In the absence of an explicit port and DNS SRV resource records, the default port for a SIP URI with "ws" transport parameter is 80 in case of SIP scheme and 443 in case of SIPS scheme.

6. Connection Keep Alive

This section is non-normative.

It is RECOMMENDED that the SIP WebSocket Client or Server keeps the WebSocket connection open by sending periodic WebSocket Ping frames as described in [RFC6455] section 5.5.2.

> Note however that The WebSocket API [WS-API] does not provide a mechanism for web applications running in a web browser to decide whether or not to send periodic WebSocket Ping frames to the server. The usage of such a keep alive feature is a decision of each web browser vendor and may depend on the web browser configuration.

Any future WebSocket protocol extension providing a keep alive mechanism could also be used.

The SIP stack in the SIP WebSocket Client MAY also use Network Address Translation (NAT) keep-alive mechanisms defined for SIP connection-oriented transports, such as the CRLF Keep-Alive Technique mechanism described in [RFC5626] section 3.5.1 or [RFC6223].

> Implementing these techniques would involve sending a WebSocket message to the SIP WebSocket Server whose content is a double CRLF, and expecting a WebSocket message from the server containing a single CRLF as response.

7. Authentication

This section is non-normative.

Prior to sending SIP requests, the SIP WebSocket Client connects to the SIP WebSocket Server and performs the connection handshake. As described in Section 3 the handshake procedure involves a HTTP GET request replied with HTTP 101 status code by the server.

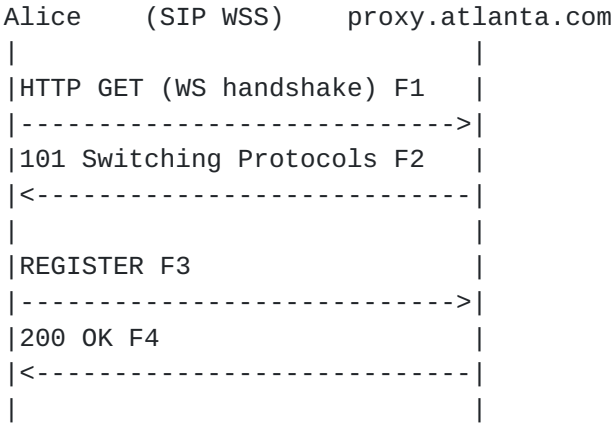
In order to authorize the WebSocket connection, the SIP WebSocket Server MAY inspect the Cookie [RFC6265] header in the HTTP GET request (if present). In case of web applications the value of such a Cookie is usually provided by the web server once the user has authenticated itself with the web server by following any of the multiple existing mechanisms. As an alternative method, the SIP WebSocket Server could request HTTP authentication by replying with a HTTP 401 status code. The WebSocket protocol [RFC6455] 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 whether the SIP WebSocket Server requires authentication during the WebSocket handshake or not, authentication MAY be requested at SIP protocol level. Therefore it is RECOMMENDED for a SIP WebSocket Client to implement HTTP Digest [RFC2617] authentication as stated in [RFC3261].

8. Examples

8.1. Registration



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 (a SIP WebSocket Client) establishes a secure WebSocket connection with a SIP proxy/registrar (a SIP WebSocket Server) 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 a random ".invalid" domain name for the Via sent-by and for the URI hostpart in the Contact header (see Appendix A.1).

Message details (authentication and SDP bodies are omitted for simplicity):

F1 HTTP GET (WS handshake) Alice -> proxy.atlanta.com (TLS)

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

F2 101 Switching Protocols proxy.atlanta.com -> Alice (TLS)

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

F3 REGISTER Alice -> proxy.atlanta.com (transport WSS)

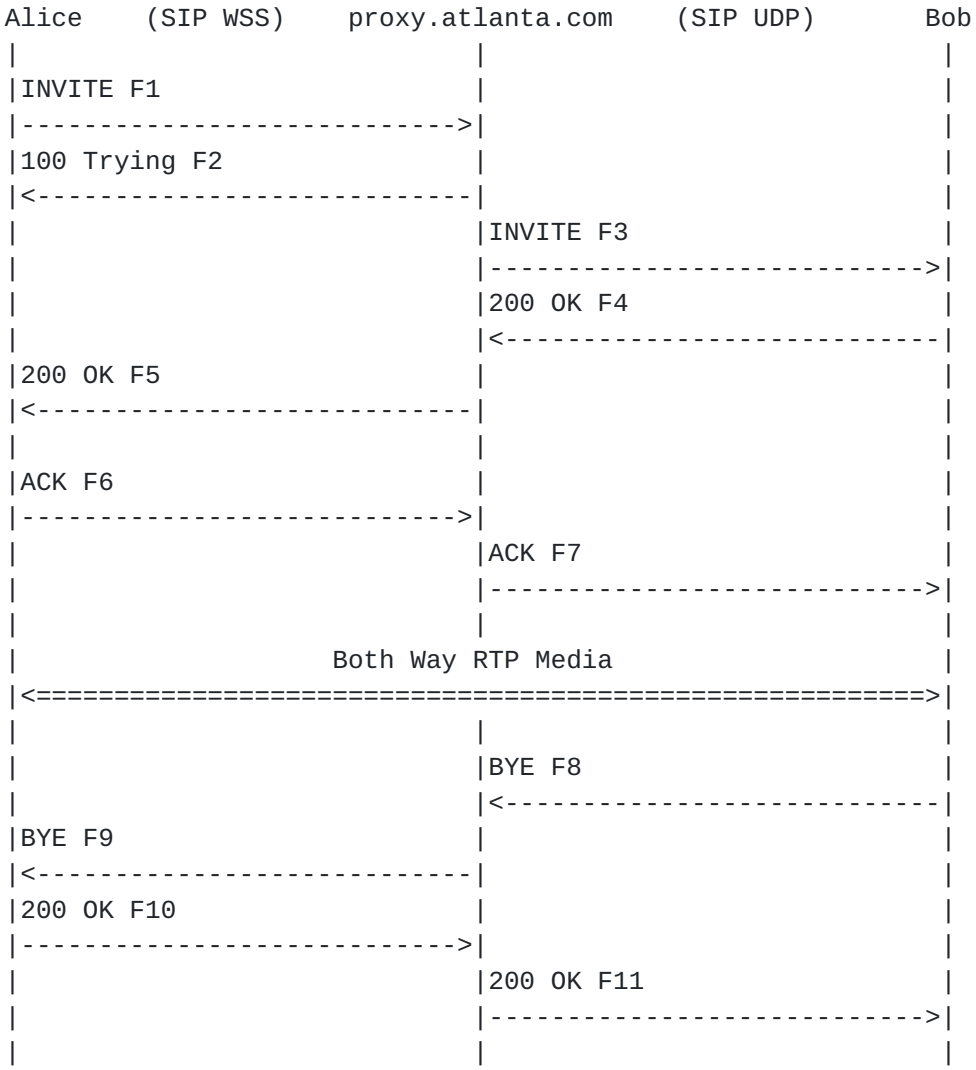
```
REGISTER sip:proxy.atlanta.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@atlanta.com;tag=65bnmj.34asd
To: sip:alice@atlanta.com
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Supported: path, outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
  ;reg-id=1
  ;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
```

F4 200 OK proxy.atlanta.com -> Alice (transport WSS)

```
SIP/2.0 200 OK
Via: SIP/2.0/WSS df7jal23ls0d.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@df7jal23ls0d.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
```

8.2. INVITE dialog through a proxy



In the same scenario Alice places a call to Bob's AoR. 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 df7jal23ls0d.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 df7jal23ls0d.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 df7jal23ls0d.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 df7jal23ls0d.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 df7jal23ls0d.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 df7jal23ls0d.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 df7jal23ls0d.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
```

9. Security Considerations

9.1. 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]).

9.2. Usage of SIPS Scheme

SIPS scheme within a SIP request dictates that the entire request path to the target be secured. If such a path includes a WebSocket node it MUST be a secure WebSocket connection.

10. IANA Considerations

10.1. 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: WebSocket Transport for SIP (Session Initiation Protocol)

Subprotocol Definition: TBD, it should point to this document

10.2. 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.

10.3. 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.

10.4. Registration of new NAPTR service field values

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:

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

11. 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, Adam Roach, Ranjit Avasarala, Xavier Marjou, Kevin P. Fleming, Nataraju A. B.

Special thanks to Alan Johnston, Christer Holmberg and Salvatore Loreto for their full reviews, and also to Saul Ibarra Corretge for his contribution and suggestions.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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.

[RFC3403] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

12.2. Informative References

[RFC2606] Eastlake, D. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, June 1999.

[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.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3327] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[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.

[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.

[RFC6223] Holmberg, C., "Indication of Support for Keep-Alive", RFC 6223, April 2011.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[WS-API] Hickson, I., "The Web Sockets API", May 2012.

Appendix A. Implementation Guidelines

This section is non-normative.

Let us assume a scenario in which the users access with their web browsers (probably behind NAT) to an intranet, perform web login by entering their user identifier and credentials, and retrieve a JavaScript code (along with the HTML code itself) implementing a SIP WebSocket Client.

Such a SIP stack connects to a given SIP WebSocket Server (an outbound SIP proxy which also implements classic SIP transports such as UDP and TCP). The HTTP GET request sent by the web browser for the WebSocket handshake includes a Cookie [RFC6265] header with the value previously retrieved after the successful web login procedure. The Cookie value is then inspected by the WebSocket server for authorizing the connection. Once the WebSocket connection is established, the SIP WebSocket Client performs a SIP registration and common SIP stuf begins. The SIP registrar server is located behind the SIP outbound proxy.

This scenario is quite similar to the one in which SIP UAs behind NAT connect to an outbound proxy and need to reuse the same TCP connection for incoming requests. In both cases, the SIP clients are just reachable through the outbound proxy they are connected to.

Outbound [RFC5626] seems an appropriate solution for this scenario. Therefore these SIP WebSocket Clients and the SIP registrar implement both Outbound and Path [RFC3327], and the SIP outbound proxy becomes an Outbound Edge Proxy (as defined in [RFC5626] section 3.4).

SIP WebSocket Clients in this scenario receive incoming SIP requests via the SIP WebSocket Server they are connected to. Therefore, in some call transfer cases the usage of GRUU [RFC5627] (which should be implemented in both the SIP WebSocket Clients and SIP registrar) is

valuable.

If a REFER request is sent to a thirdy SIP user agent indicating the Contact URI of a SIP WebSocket Client as the target in the Refer-To header field, such a URI will be reachable by the thirdy SIP UA just in the case it is a globally routable URI. GRUU (Globally Routable User Agent URI) is a solution for those scenarios, and would enforce the incoming request from the thirdy SIP user agent to reach the SIP registrar which would route the request via the Outbound Edge Proxy.

A.1. SIP WebSocket Client Considerations

The JavaScript stack in web browsers does not have the ability to discover the local transport address which the WebSocket connection is originated from. Therefore the SIP WebSocket Client creates a domain consisting of a random token followed by .invalid top domain name, as stated in [RFC2606], and uses it within the Via and Contact header.

The Contact URI provided by the SIP clients requesting Outbound support is not later used for routing purposes, thus it is safe to set a random domain in the Contact URI hostpart.

Both Outbound and GRUU specifications require the SIP 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 scenario 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 JavaScript SIP stack code, and then it is stored as a Cookie within the browser.

A.2. SIP WebSocket Server Considerations

The SIP WebSocket Server in this scenario behaves as a SIP Outbound Edge Proxy, which involves support for Outbound [RFC5626] and Path [RFC3327].

The proxy performs Loose Routing and remains in dialogs path as specified in [RFC3261]. Otherwise in-dialog requests would fail since SIP WebSocket Clients make use of their SIP WebSocket Server in order to send and receive SIP requests and responses.

Appendix B. HTTP Topology Hiding

This section is non-normative.

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 SIP WebSocket Client and 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 could reveal sensitive information about the internal topology of the provider network to the 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 SIP WebSocket Server is of little use. Therefore, in order to allow hiding possible sensitive information about the provider infrastructure, the implementer could decide not to satisfy the requirement in RFC 3261 [RFC3261] section 18.2.1 "Receiving Requests" and not add the "received" parameter to the Via header.

> However, keep in mind that this would involve a violation of the RFC 3261.

Authors' Addresses

Inaki Baz Castillo
Consultant
Barakaldo, Basque Country
Spain

Email: ibc@aliax.net

Jose Luis Millan Villegas
Consultant
Bilbao, Basque Country
Spain

Email: jmillan@aliax.net

Victor Pascual
Acme Packet
Anabel Segura 10
Madrid, Madrid 28108
Spain

Email: vpascual@acmepacket.com