

SIPCORE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: December 23, 2012

C. Holmberg  
Ericsson  
O. Gallego  
Vodafone  
June 21, 2012

Indication of support for reverse keep-alive  
draft-holmberg-sipcore-rkeep-00.txt

## Abstract

This specification defines a new Session Initiation Protocol (SIP) Via header field parameter, "rkeep". The "rkeep" parameter allows a SIP entity to indicate willingness to receive keep-alives from its adjacent downstream SIP entity, the adjacent downstream SIP entity to indicate willingness to send keep-alives, and for SIP entities willing to send keep-alives to inform the receiver about the keep-alive frequency.

## Status of this Memo

This Internet-Draft is submitted to IETF 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 23, 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
1.1. Use-case: UA not able to send keep-alives due to sleep mode . . . . .	3
2. Applicability . . . . .	3
3. Conventions . . . . .	3
4. Definitions . . . . .	4
5. User Agent and Proxy behavior . . . . .	4
5.1. General . . . . .	4
5.2. Lifetime of keep-alives . . . . .	4
5.3. Behavior of a SIP entity willing to receive keep-alives . . . . .	5
5.4. Behavior of a SIP entity willing to send keep-alives . . . . .	6
6. Keep-alive frequency . . . . .	7
7. Examples . . . . .	8
8. Grammar . . . . .	8
8.1. General . . . . .	8
8.2. ABNF . . . . .	8
9. IANA Considerations . . . . .	8
9.1. rkeep . . . . .	8
10. Security Considerations . . . . .	8
11. Acknowledgements . . . . .	8
12. Change Log . . . . .	9
13. References . . . . .	9
13.1. Normative References . . . . .	9
13.2. Informative References . . . . .	9
Authors' Addresses . . . . .	9

## 1. Introduction

RFC 6263 [RFC6263] defines a mechanism, which allows adjacent SIP entities to explicitly negotiate usage of the NAT keep-alive mechanisms defined in SIP Outbound [RFC5626]. The "keep" parameter [RFC6263] allows SIP entities, when sending a request, to indicate willingness to send keep-alives, and it allows SIP entities, when sending a response, indicate willingness to receive keep-alives.

In some cases a SIP device, located behind a NAT, is not able to send keep-alives. One reason can be that the scheduling policy of the operating system used by the SIP device does not meet the requirement for sending keep-alives using a proper frequency, meaning that NAT bindings might not be kept. In such cases, the device needs to request another device to send keep-alives towards the itself. Then, the keep-alive response message sent from the SIP device will ensure that the NAT binding is kept open.

This specification defines a new Session Initiation Protocol (SIP) Via header field parameter [RFC3261], "rkeep". The "rkeep" parameter allows a SIP entity to indicate willingness to receive keep-alives from its adjacent downstream SIP entity, the adjacent downstream SIP entity to indicate willingness to send keep-alives, and for SIP entities willing to send keep-alives to inform the receiver about the keep-alive frequency.

The following sections describe use-cases where a mechanism to explicitly negotiate usage of keep-alives is needed.

### 1.1. Use-case: UA not able to send keep-alives due to sleep mode

## 2. Applicability

The mechanism defined in this specification MUST only be used by SIP entities that are not able, e.g. because the scheduling policy of the operating system used by the SIP device does not meet the requirement for sending keep-alives using a proper frequency.

## 3. Conventions

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 BCP 14, RFC 2119 [RFC2119].

#### 4. Definitions

Keep-alives: The keep-alive messages defined in RFC 5626.

"rkeep" parameter: A SIP Via header field parameter that a SIP entity can insert in the topmost Via header field that it adds to the request, to explicitly indicate willingness to receive keep-alives from its adjacent downstream SIP entity. A SIP entity can add a parameter value to the "rkeep" parameter, or remove an existing parameter value, in a response to explicitly indicate willingness to send keep-alives to its adjacent upstream SIP entity.

SIP entity: SIP User Agent (UA), or proxy, as defined in RFC 3261.

Adjacent downstream SIP entity: The adjacent SIP entity in the direction towards which a SIP request is sent.

Adjacent upstream SIP entity: The adjacent SIP entity in the direction from which a SIP request is received.

#### 5. User Agent and Proxy behavior

##### 5.1. General

This section describes how SIP UAs and proxies negotiate usage of keep-alives associated with a registration, or a dialog, which types of SIP requests can be used in order to negotiate the usage, and the lifetime of the negotiated keep-alives.

SIP entities indicate willingness to receive keep-alives from the adjacent downstream SIP entity using SIP requests. The associated responses are used by SIP entities to indicate willingness to send keep-alives. Both SIP entities can indicate a recommended keep-alive frequency.

The procedures to negotiate usage of keep-alives are identical for SIP UAs and proxies.

NOTE: Usage of keep-alives is negotiated per direction. If a SIP entity has indicated willingness to receive keep-alives from an adjacent SIP entity, sending of keep-alives towards that adjacent SIP entity needs to be separately negotiated.

##### 5.2. Lifetime of keep-alives

The lifetime of negotiated keep-alives depends on whether the keep-alives are associated with a registration or a dialog. The rules

defined in RFC 6223 [RFC6223] also apply when keep-alives are negotiated using the 'rkeep' Via header field parameter.

### 5.3. Behavior of a SIP entity willing to receive keep-alives

As defined in RFC 5626, a SIP entity that supports receiving of keep-alives must act as a STUN server [RFC5389]. The SIP entity must support those aspects of STUN that are required in order to apply the STUN keep-alive mechanism defined in RFC 5626, and it must support the CRLF keep-alive mechanism defined in RFC 5626.

When a SIP entity sends or forwards a request, if it wants to negotiate the receiving of keep-alives associated with a registration, or a dialog, it MUST insert a "rkeep" parameter in the topmost Via header field that it adds to the request, to indicate willingness to receive keep-alives. The SIP entity MAY add a parameter value to the "rkeep" parameter before sending or forwarding the request. The parameter value, if present and with a value other than zero, represents a recommended keep-alive frequency, given in seconds.

When the SIP entity receives the associated response, if the "rkeep" parameter in the topmost Via header field of the response contains a "rkeep" parameter value with a non-zero value, or a value which is different from the value sent in the request (counting a "rkeep" parameter without a value) it MUST be prepared to start receiving keep-alives from the same destination from where it received the response which was used to negotiate the receiving of keep-alives. If the "rkeep" parameter does not contain a value, the SIP entity MUST assume that keep-alives will be sent using the recommended keep-alive frequency, if any, that the SIP entity added to the associated request.

If the associated response contains the same "rkeep" parameter value as the request (counting a "rkeep" parameter without a value), the SIP entity MUST assume that keep-alives will not be sent.

Once a SIP entity has negotiated receiving of keep-alives associated with a dialog towards an adjacent SIP entity, it MUST NOT insert a "rkeep" parameter in any subsequent SIP requests, associated with the dialog, towards that adjacent SIP entity. Such "rkeep" parameter MUST be ignored, if received.

Since an ACK request does not have an associated response, it can not be used to negotiate usage of keep-alives. Therefore, a SIP entity MUST NOT insert a "rkeep" parameter in the topmost Via header field of an ACK request. Such "rkeep" parameter MUST be ignored, if received.

A SIP entity **MUST NOT** indicate willingness to receive keep-alives associated with a dialog, unless it has also inserted itself in the dialog route set [RFC3261].

If an INVITE request is used to indicate willingness to receive keep-alives, as long as at least one response (provisional or final) to the INVITE request contains a "rkeep" parameter with a parameter value which is different from the value in the request (counting a "rkeep" parameter without a value) it is seen as an indication that the adjacent downstream SIP entity is willing to send keep-alives associated with the dialog on which the response is received.

#### 5.4. Behavior of a SIP entity willing to send keep-alives

As defined in RFC 5626, a SIP entity that supports sending of keep-alives must act as a Session Traversal Utilities for NAT (STUN) client [RFC5389]. The SIP entity must support those aspects of STUN that are required in order to apply the STUN keep-alive mechanism defined in RFC 5626, and it must support the CRLF keep-alive mechanism defined in RFC 5626. RFC 5626 defines when to use STUN, respectively double-CRLF, for keep-alives.

When a SIP entity sends or forwards a response, and the adjacent upstream SIP entity indicated willingness to receive keep-alives without a recommended keep-alive frequency, if the SIP entity is willing to send keep-alives associated with the registration, or the dialog, to the adjacent upstream SIP entity it **MUST** add a parameter value to the "rkeep" parameter, before sending or forwarding the response. The parameter value, represents the keep-alive frequency, given in seconds, that the sender of the keep-alives will use.

If the adjacent upstream SIP entity provided a recommended keep-alive frequency, and if the SIP entity is willing to send keep-alives using that frequency, it **MUST** remove the "rkeep" parameter value which indicates the recommended keep-alive frequency, before sending or forwarding the response.

NOTE: The reason for removing the "rkeep" parameter value is to indicate that the SIP entity supports the mechanism, and simply does not forward an unknown parameter.

If the adjacent upstream SIP entity provided a recommended keep-alive frequency, and if the SIP entity is willing to send keep-alives using a different frequency, it **MUST** modify the "rkeep" parameter value which indicates the recommended keep-alive frequency, before sending or forwarding the response.

There might be multiple responses to an INVITE request. When a SIP

entity indicates willingness to send keep-alives in a response to an INVITE request, it MUST add a parameter value to the "rkeep" parameter in at least one reliable response to the request. The SIP entity MAY add identical parameter values to the "rkeep" parameters in other responses to the same request. The SIP entity MUST NOT add different parameter value to the "rkeep" parameters in responses to the same request. The SIP entity SHOULD indicate the willingness to send keep-alives as soon as possible.

A SIP entity MUST NOT indicate willingness to send keep-alives associated with a dialog, unless it has also inserted itself in the dialog route set [RFC3261].

When the SIP entity has indicated willingness to send keep-alives, it MUST start sending keep-alives towards the same destination where it sent the response used to indicate willingness to send keep-alives.

## 6. Keep-alive frequency

If a SIP entity sends a SIP response, where the topmost Via header field contains a "rkeep" parameter with a non-zero value that indicates the keep-alive frequency, given in seconds, it MUST use the procedures defined for the Flow-Timer header field [RFC5626]. According to the procedures, the SIP entity must send keep-alives at least as often as the indicated keep-alive frequency, and if the SIP entity uses the indicated keep-alive frequency then it should send its keep-alives so that the interval between each keep-alive is randomly distributed between 80% and 100% of the indicated keep-alive frequency.

This specification does not define any semantic for a "rkeep" Via header parameter value with a zero value. A SIP entity MUST NOT insert a zero value to a "rkeep" parameter. A SIP entity that receives a "rkeep" parameter with a zero value SHOULD NOT assume that the sender of the response will send keep-alives towards the SIP entity.

This specification does not specify actions to take if negotiated keep-alives are not received. As defined in RFC 5626, the receiving SIP entity may consider a connection to be dead in such situations.

NOTE: The Flow-Timer header field [RFC5626] has no impact on keep-alives negotiated using the "rkeep" Via header field parameter.

## 7. Examples

## 8. Grammar

### 8.1. General

This section describes the syntax extensions to the ABNF syntax defined in RFC 3261, by defining a new Via header field parameter, "rkeep". The ABNF defined in this specification is conformant to RFC 5234 [RFC5234].

### 8.2. ABNF

```
via-params =/ rkeep
```

```
rkeep      = "rkeep" [ EQUAL 1*(DIGIT) ]
```

## 9. IANA Considerations

### 9.1. rkeep

This specification defines a new Via header field parameter called keep in the "Header Field Parameters and Parameter Values" sub-registry as per the registry created by [RFC3968]. The syntax is defined in Section 8. The required information is:

Header Field	Parameter Name	Predefined Values	Reference
Via	rkeep	No	[RFCXXXX]

## 10. Security Considerations

The Security Considerations in RFC 6223 apply.

## 11. Acknowledgements



## 12. Change Log

[RFC EDITOR NOTE: Please remove this section when publishing]

Changes from draft-holmberg-sipcore-rkeep-xx

- o TBA

## 13. References

### 13.1. 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.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.
- [RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.
- [RFC6223] Holmberg, C., "Indication of Support for Keep-Alive", RFC 6223, April 2011.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, June 2011.

### 13.2. Informative References

- [RFC3968] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

Authors' Addresses

Christer Holmberg  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [christer.holmberg@ericsson.com](mailto:christer.holmberg@ericsson.com)

Oscar Gallego  
Vodafone  
One Kingdom Street, Paddington Central  
London W2 6BY  
UK

Email: [oscar.gallego@vodafone.com](mailto:oscar.gallego@vodafone.com)



SIPCORE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: December 16, 2012

C. Holmberg  
I. Sedlacek  
Ericsson  
H. Kaplan  
Acme Packet  
June 14, 2012

Mechanism to indicate support of features and capabilities in the  
Session Initiation Protocol (SIP)  
draft-ietf-sipcore-proxy-feature-04.txt

#### Abstract

This specification creates a new IANA registry, "Proxy-Feature Feature Caps Trees", for registering "feature caps", which are used by SIP entities not represented by the URI of the Contact header field to indicate support of features and capabilities, where media feature tags cannot be used to indicate the support.

This specification also defines a new SIP header field, Feature-Caps, to convey feature caps in SIP messages.

#### Status of this Memo

This Internet-Draft is submitted to IETF 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 16, 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 . . . . .	4
2. Conventions . . . . .	4
3. Definitions . . . . .	4
4. Feature Caps . . . . .	4
4.1. Introduction . . . . .	5
4.2. Registration Trees . . . . .	5
4.2.1. General . . . . .	5
4.2.2. Global Tree . . . . .	5
4.2.3. SIP Tree . . . . .	6
4.3. Registration Template . . . . .	6
4.4. Feature Cap Specification Requirements . . . . .	8
4.4.1. General . . . . .	8
4.4.2. Overall Description . . . . .	8
4.4.3. Feature Cap Values . . . . .	8
4.4.4. Usage Restrictions . . . . .	9
4.4.5. Examples . . . . .	9
5. Feature-Caps Header Field . . . . .	9
5.1. Introduction . . . . .	9
5.2. User Agent and Proxy Behavior . . . . .	10
5.2.1. General . . . . .	10
5.2.2. B2BUA Behavior . . . . .	10
5.2.3. Registrar Behavior . . . . .	11
5.2.4. Proxy behavior . . . . .	11
5.3. SIP Message Type and Response Code Semantics . . . . .	11
5.3.1. General . . . . .	11
5.3.2. SIP Dialog . . . . .	12
5.3.3. SIP Registration (REGISTER) . . . . .	12
5.3.4. SIP Stand-Alone Transactions . . . . .	13
6. Syntax . . . . .	13
6.1. General . . . . .	13
6.2. Syntax: feature cap . . . . .	13
6.2.1. General . . . . .	13
6.2.2. ABNF . . . . .	13
6.3. Syntax: Feature-Caps header field . . . . .	14
6.3.1. ABNF . . . . .	14
7. IANA Considerations . . . . .	14
7.1. Registration of the Feature-Caps header field . . . . .	14
7.2. Proxy-Feature Feature Caps Trees . . . . .	15

7.2.1. Introduction . . . . .	15
7.2.2. Global Feature Cap Registration Tree . . . . .	15
7.2.3. SIP Feature Cap Registration Tree . . . . .	15
8. Security Considerations . . . . .	15
9. Acknowledgements . . . . .	16
10. Change Log . . . . .	16
11. References . . . . .	17
11.1. Normative References . . . . .	17
11.2. Informative References . . . . .	17
Authors' Addresses . . . . .	18

## 1. Introduction

The Session Initiation Protocol (SIP) [RFC3261] "Caller Preferences" extension, defined in RFC 3840 [RFC3840], provides a mechanism that allows a SIP message to convey information relating to the originator's features and capabilities, using the Contact header field.

This specification creates a new IANA registry, "Proxy-Feature Feature Caps Trees", for registering "feature caps", which are used by SIP entities not represented by the URI of the Contact header field to indicate support of features and capabilities, where media feature tags cannot be used to indicate the support. Such cases are:

- o - The SIP entity acts as a SIP proxy.
- o - The SIP entity acts as a SIP registrar.
- o - The SIP entity acts as a B2BUA, where the Contact header field URI represents another SIP entity.

This specification also defines a new SIP header field, Feature-Caps, to convey feature caps in SIP messages.

NOTE: Unlike media feature tags, feature caps are intended to only be used with the SIP protocol.

## 2. Conventions

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 BCP 14, RFC 2119 [RFC2119].

## 3. Definitions

Downstream SIP entity: SIP entity in the direction towards which a SIP request is sent.

Upstream SIP entity: SIP entity in the direction from which a SIP request is received.

## 4. Feature Caps

#### 4.1. Introduction

Feature caps are used by SIP entities not represented by the URI of the Contact header field to indicate support of features and capabilities, where media feature tags cannot be used to indicate the support.

A value, or a list of values, that provides additional information about the supported feature or capability, can be associated with a feature cap.

Section 5 defines how feature caps are conveyed using the Feature-Caps header field.

The feature cap ABNF is defined in Section 6.2.2.

#### 4.2. Registration Trees

##### 4.2.1. General

The following subsections define registration trees, distinguished by the use of faceted names (e.g., names of the form "tree.feature-name"). The registration trees are defined in the IANA "Proxy-Feature Feature Caps Trees" registry.

The trees defined herein are similar to the global tree and sip tree defined for media feature tags, in RFC 2506 [RFC2506] and RFC 3840 [RFC3840]. Other registration trees are outside the scope of this specification.

NOTE: In contrast to RFC 2506 and RFC 3840, this specification only defines a global tree and a sip tree, as they are the only trees defined in those RFCs that have been used for defining SIP-specific media feature tags.

When a feature cap is registered in any registration tree, no leading "+" is used in the registration.

##### 4.2.2. Global Tree

The global feature cap tree is similar to the media feature tag global tree defined in RFC 2506 [RFC2506].

A feature cap for the global tree will be registered by the IANA after review by a designated expert. That review will serve to ensure that the feature cap meets the technical requirements of this specification.



A feature cap in the global tree will be distinguished by the leading facet "g.". An organization can propose either a designation indicative of the feature, (e.g., "g.blinktags") or a faceted designation including the organization name (e.g., "g.organization.blinktags").

When a feature cap is registered in the global tree, it needs to meet the "Expert Review" policies defined in RFC 5226 [RFC5226]. A designated area expert will review the proposed feature cap, and consult with members of related mailing lists.

#### 4.2.3. SIP Tree

The sip feature cap tree is similar to the media feature tag sip tree defined in RFC 3840 [RFC3840].

A feature cap in the sip tree will be distinguished by the leading facet "sip.".

When a feature cap is registered in the sip tree, it needs to meet the "IETF Consensus" policies defined in RFC 5226 [RFC5226]. An RFC, which contains the registration of the feature cap, MUST be published.

#### 4.3. Registration Template

To: sip-feature-caps@apps.ietf.org (feature caps mailing list)  
Subject: Registration of feature cap XXXX

| Instructions are preceded by '|'. Some fields are optional.

Feature cap name:

Summary of feature indicated by this feature cap:

| The summary should be no longer than 4 lines. More  
| detailed information can be provided in the SIP feature  
| cap specification.

Feature cap specification reference:

| The referenced specification MUST contain the information  
| listed in section XX of XXXX (IANA: Replace XXXX with  
| assigned RFC number of this specification.

Values appropriate for use with this feature cap:

| If no values are defined for the feature cap,  
| indicate "N/A". Details about feature cap values  
| MUST be defined in the feature cap specification.

The feature cap is intended primarily for  
use in the following applications, protocols,  
services, or negotiation mechanisms: [optional]

| For applications, also specify the number of the  
| first version which will use the feature cap,  
| if applicable.

Examples of typical use: [optional]

Considerations particular to use in individual  
applications, protocols, services, or negotiation  
mechanisms: [optional]

Interoperability considerations: [optional]

Security considerations:

Privacy concerns, related to exposure of personal  
information:

Denial of service concerns related to consequences  
of specifying incorrect values:

Other:

Additional information: [optional]

Keywords: [optional]

Related feature caps: [optional]

Name(s) & email address(es) of person(s) to  
contact for further information:

Intended usage:

| one of COMMON, LIMITED USE or OBSOLETE

Author/Change controller:

Other information: [optional]

| Any other information that the author deems

| interesting may be added here.

Figure 1: Registration Template

#### 4.4. Feature Cap Specification Requirements

##### 4.4.1. General

A feature cap specification **MUST** address the issues defined in the following subsections, or document why an issue is not applicable for the specific feature cap. A reference to the specification **MUST** be provided when the feature cap is registered with IANA (see Section 4.3).

It is bad practice for feature cap specifications to repeat procedures (e.g. general procedures on the usage of the Feature-Caps header field and feature caps) defined in this specification, unless needed for clarification or emphasis purpose.

A feature cap specification **MUST NOT** weaken any behavior designated with "SHOULD" or "MUST" in this specification. However, a specification **MAY** strengthen "SHOULD", "MAY", or "RECOMMENDED" requirements to "MUST" strength if features and capabilities associated with the SIP feature cap require it.

##### 4.4.2. Overall Description

The feature cap specification **MUST** contain an overall description of the feature cap: how it is used to indicate support of a feature, a description of the feature associated with the SIP feature cap, a description of any additional information (conveyed using one or more feature cap values) that can be conveyed together with the feature cap, and a description of how the associated feature may be exercised/invoked.

##### 4.4.3. Feature Cap Values

A feature cap can have an associated value, or a list of values.

The feature cap specification **MUST** define the syntax and semantics of any value defined for the feature cap, including possible restrictions related to the usage of a specific value. The feature cap specification **MUST** define the value(s) in accordance with the syntax defined in section 6.2.2.

A feature cap value is only applicable for the feature cap for which it has been defined. For other feature caps, the value has to be

defined explicitly, even if the semantics are identical.

It is **STRONGLY RECOMMENDED** to not re-use a value that already has been defined for another feature cap, unless the semantics of the values are the same.

#### 4.4.4. Usage Restrictions

If there are restrictions on how SIP entities can insert a SIP feature cap, the feature cap specification **MUST** document such restrictions.

There might be restrictions related to whether entities are allowed to insert a feature cap in registration related messages, standalone transaction messages, or dialog related messages, whether entities are allowed to insert a feature cap in requests or responses, whether entities also need to support other features and capabilities in order to insert a feature cap, and whether entities are allowed to indicate support of a feature in conjunction with another feature.

#### 4.4.5. Examples

It is **RECOMMENDED** that the feature cap specification provide demonstrative message flow diagrams, paired with complete messages and message descriptions.

Note that example message flows are by definition informative, and do not replace normative text.

### 5. Feature-Caps Header Field

#### 5.1. Introduction

The Feature-Caps header field is used by SIP entities to convey support of features and capabilities, by setting feature caps. Feature caps conveyed in a Feature-Caps header field indicate that the SIP entity that inserted the header field supports the associated features and capabilities.

**NOTE:** It is not possible to, as a Feature-Caps header field value, convey the address of the SIP entity that inserted the Feature-Caps header field. If additional data about a supported feature needs to be conveyed, such as the address of the SIP entity that indicated support of the feature, then the feature definition needs to define a way to convey that information as a value of the associated feature cap.

The feature cap specification MUST specify for which SIP methods and message types, and the associated semantics, the feature cap is applicable. See Section 4 for more information. No semantics is defined for feature caps present in SIP methods and message types not covered by the associated feature cap specification.

Within a given Feature-Caps header field, feature caps are listed in a non-priority order, and for a given header field any order of listed SIP feature caps have the same meaning. For example, "foo;bar" and "bar;foo" have the same meaning (i.e. that the SIP entity that inserted the feature caps supports the features and capabilities associated with the "foo" and "bar" feature caps.

## 5.2. User Agent and Proxy Behavior

### 5.2.1. General

If the URI in a Contact header field of a request or response represents a SIP entity, the entity MUST NOT indicate supported features and capabilities using a Feature-Caps header field within that request or response.

When a SIP entity receives a SIP request, or response, that contains one or more Feature-Caps header fields, the feature caps in the header field inform the entity about the features and capabilities supported by the entities that inserted the header fields. Procedures how features and capabilities are invoked are outside the scope of this specification, and MUST be described by individual feature cap specifications.

When a SIP entity adds a Feature-Caps header field to a SIP message, it MUST place the header field before any existing Feature-Caps header field in the message to be forwarded, so that the added header field becomes the top-most one. Then, when another SIP entity receives a SIP request or the response, the SIP feature caps in the top-most Feature-Caps header field will represent the supported features and capabilities "closest" to the entity.

### 5.2.2. B2BUA Behavior

The procedures in this Section applies to UAs that are part of B2BUAs that are referenced in the message by a Record-Route header field rather than by the URI of the Contact header field.

When a UA sends a SIP request, if the UA wants to indicate support of features and capabilities towards its downstream SIP entities, it inserts a Feature-Caps header field to the request, containing one or more feature caps associated with the supported features and

capabilities, before it forwards the request.

If the SIP request is triggered by another SIP request that the B2BUA has received, the UA MAY forward received Feature-Caps header fields by copying them to the outgoing SIP request, similar to a SIP proxy, before it inserts its own Feature-Caps header field to the SIP request.

When a UA receives a SIP response, if the UA wants to indicate support of features and capabilities towards its upstream SIP entities, it inserts a Feature-Caps header field to the response, containing one or more feature caps associated with the supported features and capabilities, before it forwards the response.

If the SIP response is triggered by another SIP response that the B2BUA has received, the UA MAY forward received Feature-Caps header field by copying them to the outgoing SIP response, similar to a SIP proxy, before it inserts its own Feature-Caps header field to the SIP response.

#### 5.2.3. Registrar Behavior

If a SIP registrar wants to indicate support of features and capabilities towards its upstream SIP entities, it inserts a Feature-Caps header field, containing one or more feature caps associated with the supported features and capabilities, to a REGISTER response.

#### 5.2.4. Proxy behavior

When a SIP proxy receives a SIP request, if the proxy wants to indicate support of features and capabilities towards its downstream SIP entities, it inserts a Feature-Caps header field to the request, containing one or more SIP feature caps associated with the supported features and capabilities, before it forwards the request.

When a proxy receives a SIP response, if the proxy wants to indicate support of features and capabilities towards its upstream SIP entities, it inserts a Feature-Caps header field to the response, containing one or more SIP feature caps associated with the supported features and capabilities, before it forwards the response.

### 5.3. SIP Message Type and Response Code Semantics

#### 5.3.1. General

This Section describes the general usage and semantics of the Feature-Caps header field for different SIP message types and response codes. The usage and semantics of a specific feature cap

MUST be described in the associated feature cap specification.

NOTE: Future specifications can define usage and semantics of the Feature-Caps header field for SIP methods, response codes and request types not specified in this specification.

The Feature-Caps header field ABNF is defined in Section 6.3.1.

#### 5.3.2. SIP Dialog

The Feature-Caps header field can be used within an initial SIP request for a dialog, within a target refresh SIP request, and within any 18x or 2xx response associated with such requests.

If a feature cap is inserted in a Feature-Caps header field of an initial request for a dialog, or within a response of such request, it indicates to the receivers of the request (or response) that the feature associated with the feature cap is supported for the duration of the dialog, until a target refresh request is sent for the dialog, or the dialog is terminated.

Unless a feature cap is inserted in a Feature-Caps header field or a target refresh request, or within a response of such request, it indicates to the receivers of the request (or response) that the feature is no longer supported for the dialog.

For a given dialog a SIP entity MUST insert the same feature caps in all 18x and 2xx responses associated with a given transaction.

#### 5.3.3. SIP Registration (REGISTER)

The Feature-Caps header field can be used within a SIP REGISTER request, and within the 200 (OK) response associated with such request.

If a feature cap is conveyed in a Feature-Caps header field of a REGISTER request, or within an associated response, it indicates to the receivers of the message that the feature associated with the feature cap is supported for the registration, until the registration of the contact that was explicitly conveyed in the REGISTER request expires, or until the registered contact is explicitly refreshed and the refresh REGISTER request does not contain the feature cap associated with the feature.

NOTE: While a REGISTER response can contain contacts that have been registered as part of other registration transactions, support of any indicated feature only applies to the contact(s) that were explicitly conveyed in the associated REGISTER request.

This specification does not define any semantics for usage of the Feature-Caps header field in pure registration binding fetching messages (see Section 10.2.3 of RFC 3261), where the REGISTER request does not contain a Contact header field. Unless such semantics is defined in a future extension, fetching messages will not have any impact on previously indicated support of features and capabilities, and SIP entities MUST NOT insert a Feature-Caps header field to such messages.

If SIP Outbound [RFC5626] is used, the rules above apply. However, supported features and capabilities only apply for the registration flow on which support has been explicitly indicated.

#### 5.3.4. SIP Stand-Alone Transactions

The Feature-Caps header field can be used within a standalone SIP request, and within any 18x or 2xx response associated with such request.

If a feature cap is inserted in a Feature-Caps header field of a standalone request, or within a response of such request, it indicates to the receivers of the request (or response) that the feature associated with the feature cap is supported for the duration of the standalone transaction.

## 6. Syntax

### 6.1. General

This Section defines the ABNF for Feature-Caps, and for the Feature-Cap header field.

### 6.2. Syntax: feature cap

#### 6.2.1. General

In a feature cap name (ABNF: fcap-name), dots can be used to implement a SIP feature cap tree hierarchy (e.g. tree.feature.subfeature). The description of usage of such tree hierarchy must be described when registered.

#### 6.2.2. ABNF

The ABNF for the feature cap:



```
feature-cap      = "+" fcap-name [EQUAL LDQUOTE (fcap-value-list
                               / fcap-string-value ) RDQUOTE]
fcap-name        = ftag-name
fcap-value-list  = tag-value-list
fcap-string-value = string-value
;; ftag-name, tag-value-list, string-value defined in RFC 3840
```

NOTE: In comparison with media feature tags, the "+" sign in front of the feature cap name is mandatory.

Figure 2: ABNF

### 6.3. Syntax: Feature-Caps header field

#### 6.3.1. ABNF

The ABNF for the Feature-Caps header fields is:

```
Feature-Caps = "Feature-Caps" HCOLON fc-value
              *(COMMA fc-value)
fc-value     = "*" *(SEMI feature-cap)
```

Figure 3: ABNF

NOTE: A "\*" value means that no information regarding which SIP entity, or domain, that indicate support of features and capabilities is provided.

## 7. IANA Considerations

### 7.1. Registration of the Feature-Caps header field

This specification registers a new SIP header field, Feature-Caps, according to the process of RFC 3261 [RFC3261].

The following is the registration for the Feature-Caps header field:

RFC Number: RFC XXX

Header Field Name: Feature-Caps

## 7.2. Proxy-Feature Feature Caps Trees

### 7.2.1. Introduction

This specification creates a new sub registry to the IANA "Session Initiation Protocol (SIP) Parameters" Protocol Registry, per the guidelines in RFC 5226 [RFC5226]. The name of the sub registry is "Proxy-Feature Feature Caps Trees".

### 7.2.2. Global Feature Cap Registration Tree

This specification creates a new feature cap tree in the IANA "Proxy-Feature Feature Caps Trees" registry. The name of the tree is "Global Feature Cap Registration Tree", and its leading facet is "g.". It is used for the registration of feature caps.

The addition of entries into this tree occurs through the Expert Review policies, as defined in RFC 5226. A designated area expert will review the proposed feature cap, and consult with members of related mailing lists. The information required in the registration is defined in Section 4.3 of RFC XXX.

Note that all feature caps registered in the global tree will have names with a leading facet "g.". No leading "+" is used in the registrations in any of the feature cap registration trees.

### 7.2.3. SIP Feature Cap Registration Tree

This specification creates a new feature cap tree in the IANA "Proxy-Feature Feature Caps Trees" registry. The name of the tree is "SIP Feature Cap Registration Tree", and its leading facet is "sip.". It is used for the registration of feature caps.

The addition of entries into this tree occurs through the IETF Consensus, as defined in RFC 5226. This requires the publication of an RFC that contains the registration. The information required in the registration is defined in Section 4.3 of RFC XXX.

Note that all feature caps registered in the SIP tree will have names with a leading facet "sip.". No leading "+" is used in the registrations in any of the feature cap registration trees.

## 8. Security Considerations

The security issues for feature caps are similar to the ones defined in RFC 3840 for media feature tags. However, as feature caps will typically not be used to convey capability information of end-user

devices, those aspects of RFC 3840 do not apply to feature caps.

In addition, the RFC 3840 security issue regarding an attacker using the SIP caller preferences extension [RFC3841] in order to affect routing decisions does not apply, as the mechanism is not defined to be used with feature caps.

Feature caps can provide capability and characteristics information about the SIP entity, some of which might be sensitive. The Feature-Caps header field does not convey address information about SIP entities. However, individual feature caps might provide address information as feature cap values. Therefore, mechanisms for guaranteeing confidentiality and authenticity SHOULD be provided.

## 9. Acknowledgements

The authors wish to thank everyone in the SIP community that provided input and feedback on the work of this specification.

## 10. Change Log

[RFC EDITOR NOTE: Please remove this Section when publishing]

Changes from draft-ietf-sipcore-proxy-feature-03

- o Additional Security Considerations text added.
- o IANA Considerations modified.
- o Editorial corrections.

Changes from draft-ietf-sipcore-proxy-feature-02

- o Changes based on WGLC comments from Shida Schubert.
- o - Document title changed
- o - Terminology alignment
- o - Note text clarifications
- o Changes based on WGLC comments from Lili Yang.

Changes from draft-ietf-sipcore-proxy-feature-01

- o Changes based on comments from Paul Kyzivat.
- o IANA Considerations text added.

Changes from draft-holmberg-sipcore-proxy-feature-04/  
draft-ietf-sipcore-proxy-feature-00

- o Media feature tags replaced with feature caps, based on SIPCORE consensus at IETF#83 (Paris).
- o Editorial corrections and modifications.

Changes from draft-holmberg-sipcore-proxy-feature-03

- o Hadriel Kaplan added as co-author.
- o Terminology change: instead of talking of proxies, talk about entities which are not represented by the URI in a Contact header field (<http://www.ietf.org/mail-archive/web/sipcore/current/msg04449.html>).
- o Clarification regarding the usage of the header field in 18x/2xx responses (<http://www.ietf.org/mail-archive/web/sipcore/current/msg04449.html>).
- o Specifying that feature support can also be indicated in target refresh requests (<http://www.ietf.org/mail-archive/web/sipcore/current/msg04454.html>).
- o Feature Cap specification registration information added.

Changes from draft-holmberg-sipcore-proxy-feature-02

- o Definition, and usage of, a new header field, instead of Path, Record-Route, Route and Service-Route.

Changes from draft-holmberg-sipcore-proxy-feature-01

- o Requirement section added
- o Use-cases and examples updated based on work in 3GPP

Changes from draft-holmberg-sipcore-proxy-feature-00

- o Additional use-cases added
- o Direction section added

## 11. References

### 11.1. 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.

### 11.2. Informative References

- [RFC2506] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.
- [RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.
- [RFC3841] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller

Preferences for the Session Initiation Protocol (SIP)",  
RFC 3841, August 2004.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an  
IANA Considerations Section in RFCs", BCP 26, RFC 5226,  
May 2008.

[RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-  
Initiated Connections in the Session Initiation Protocol  
(SIP)", RFC 5626, October 2009.

[3GPP.23.237]  
3GPP, "IP Multimedia Subsystem (IMS) Service Continuity;  
Stage 2", 3GPP TS 23.237 10.9.0, March 2012.

[3GPP.24.837]  
3GPP, "IP Multimedia (IM) Core Network (CN) subsystem  
inter-UE transfer enhancements; Stage 3", 3GPP TR 24.837  
10.0.0, April 2011.

#### Authors' Addresses

Christer Holmberg  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: christer.holmberg@ericsson.com

Ivo Sedlacek  
Ericsson  
Scheelewaegen 19C  
Lund 22363  
Sweden

Email: ivo.sedlacek@ericsson.com

Hadriel Kaplan  
Acme Packet  
71 Third Ave.  
Burlington, MA 01803  
USA

Email: [hkaplan@acmepacket.com](mailto:hkaplan@acmepacket.com)



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 standardized 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      (SIP WSS)      proxy.atlanta.com
|
| HTTP GET (WS handshake) F1 |
|----->|
| 101 Switching Protocols F2 |
|<-----|
|
| REGISTER F3                |
|----->|
| 200 OK F4                  |
|<-----|
|
```

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
```

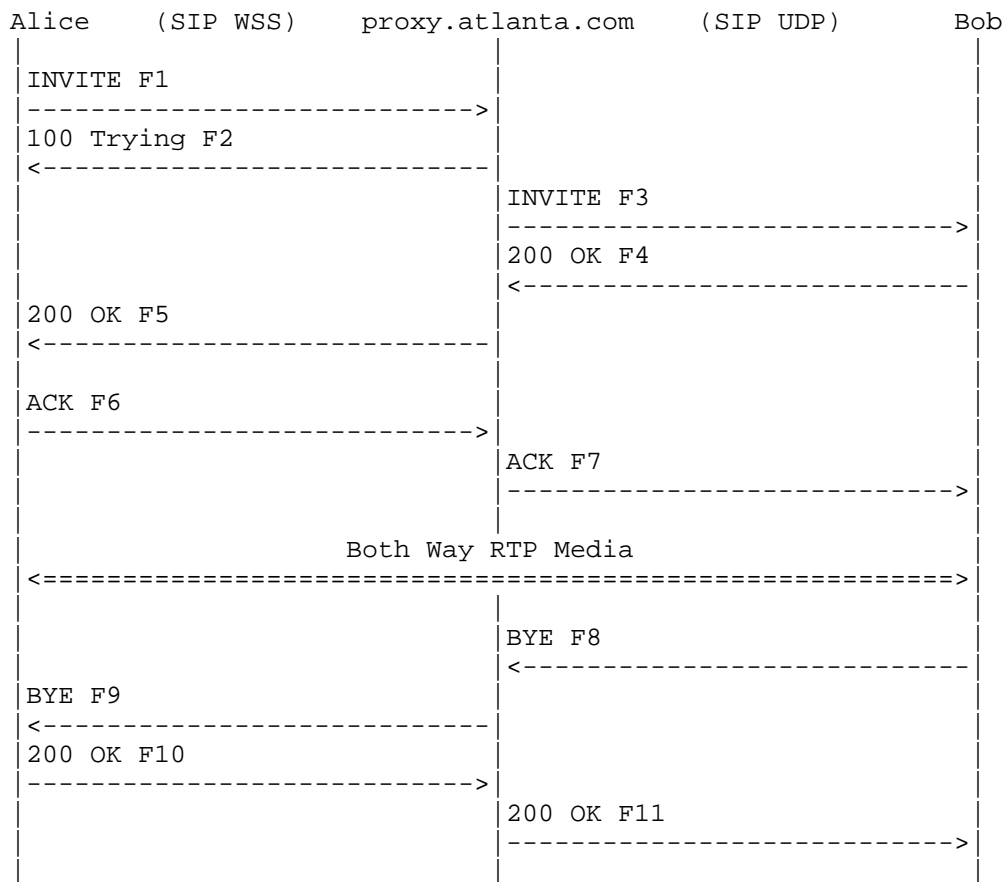


```

From: sip:alice@atlanta.com;tag=65bnmj.34asd
To: sip:alice@atlanta.com;tag=12isjln8
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-14a06cfl;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=z9hG4bKKhjhjqw32c
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-14a06cfl;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=z9hG4bKhgqpp090  
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=z9hG4bKhwpc80zzx  
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 stuff 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](mailto:jmillan@aliax.net)

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

Email: [vpascual@acmepacket.com](mailto:vpascual@acmepacket.com)

