

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: July 20, 2014

P. Dawes  
Vodafone Group  
January 16, 2014

Requirements for Marking SIP Messages to be Logged  
draft-dawes-insipid-logme-reqs-00

Abstract

SIP networks use signalling monitoring tools to diagnose user reported problem and for regression testing if network or client software is upgraded. As networks grow and become interconnected, including connection via transit networks, it becomes impractical to predict the path that SIP signalling will take between clients, and therefore impractical to monitor SIP signalling end-to-end.

This draft describes requirements for adding an indicator to the SIP protocol which can be used to mark signalling as of interest to logging. Such marking will typically be applied as part of network testing controlled by the network operator and not used in regular client signalling. However, such marking can be carried end-to-end including the SIP terminals, even if a session originates and terminates in different networks.

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 July 20, 2014.

Copyright Notice

Copyright (c) 2014 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 . . . . .	2
2. Requirements Language . . . . .	2
3. Motivating Scenario . . . . .	3
4. Skeleton Diagnostic Procedure . . . . .	4
5. Requirements for a Log Me Marker . . . . .	5
6. Security Considerations . . . . .	6
6.1. Trust Domain . . . . .	6
6.2. Security Threats . . . . .	6
6.2.1. Log-me marking . . . . .	6
6.2.2. Sending logged information . . . . .	7
7. References . . . . .	7
7.1. Normative References . . . . .	7
7.2. Informative References . . . . .	7
Appendix A. Additional Stuff . . . . .	8
Author's Address . . . . .	8

## 1. Introduction

If users experience problems with setting up sessions using SIP, their service provider needs to find out why by examining the SIP signalling. Also, if network or client software or hardware is upgraded regression testing is needed. Such diagnostics apply to a small proportion of network traffic and can apply end-to-end, even if signalling crosses several networks possibly belonging to several different network operators. It may not be possible to predict the path through those networks in advance, therefore a mechanism is needed to mark a session as being of interest to enable SIP entities along the signalling path to provide diagnostic logging. This draft describes the requirements for such a 'log me' marker for SIP signalling.

## 2. Requirements Language

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 RFC 2119 [RFC2119].

### 3. Motivating Scenario

Signalling for SIP session setup can cross several networks, and these networks may not have common ownership and also may be in different countries. If a single operator wishes to perform regression testing or fault diagnosis end-to-end, the separate ownership of networks that carry the signalling and the explosion in the number of possible signalling paths through SIP entities from the originating to the terminating user make it impractical to pre-configure logging of an end-to-end SIP signalling of a session of interest.

The figure below shows an example of a signalling path through multiple networks.

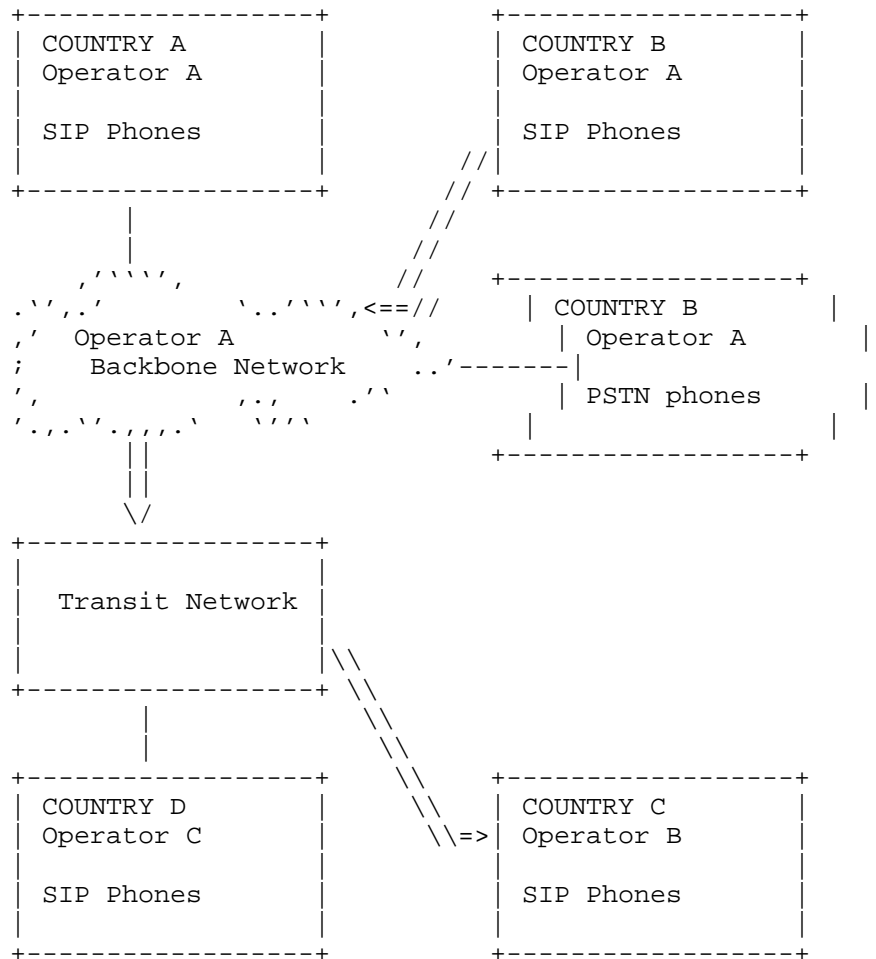


Figure 1: Example signalling path through multiple networks

#### 4. Skeleton Diagnostic Procedure

The skeleton diagnostic procedure is as follows:

- o The user's terminal is placed in debug mode. The terminal logs its own signalling and inserts a log me marker into SIP requests for session setup
- o All SIP entities that the signalling traverses, from the first proxy the terminal connects to at the edge of the network to the destination client terminal, can detect that the log me marker is

present and can log SIP requests and responses that contain the marker if configured to do so.

- o Subsequent responses and requests in the same dialog are logged.
- o Logging stops, either because the dialog has ended or because a 'stop event', typically expiry of a certain amount of time, occurred
- o The user's terminal and any other SIP entity that has logged signalling sends logs to a server that is co-ordinating diagnostics.

#### 5. Requirements for a Log Me Marker

- o REQ1: It shall be possible to mark a SIP request or response as of interest for logging by inserting a log me marker. This is known as log-me marking.
- o REQ2: It shall be possible for a log-me marker to cross network boundaries.
- o REQ3: A log-me marker is most effective if it passes end-to-end. However, source networks should behave responsibly and not leave it to a downstream network to detect and remove a marker that it will not use. A log-me marker should be removed at trust domain boundaries.
- o REQ4: SIP entities should log SIP requests or responses with a log-me marker.
- o REQ5: If a UA receives a request with a log-me marker, it shall echo that log-me marker in responses to that request.
- o REQ6: A SIP proxy may perform log-me marking of requests and responses. Typical cases where a proxy needs to perform log-me marking are when a UA has not marked a request and when responses received on a dialog of interest for logging do not contain a log-me marker. In these cases, the entity that performs log-me marking is stateful inasmuch as it must remember when a dialog is of interest for logging.
- o REQ7: For SIP proxies, logging of SIP requests that contain a log-me marker may be stateless. For example, it is not required for a SIP entity to maintain state of which SIP requests contained a log-me marker in order to log responses to those requests. Echoing a log-me marker in responses is the responsibility of the UA that receives a request.

- o REQ8: A log-me marker may include an identifier that indicates the test case that caused it to be inserted, known as a test case identifier. The test case identifier does not have any impact on session setup, it is used by the diagnostic server to collate all logged SIP requests and responses to the initial SIP request in a dialog or standalone transaction. The Session-ID described in I-D .ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts] could be used as the test case identifier but it would be useful for the UA to log a human readable name together with this Session-ID when it performs log me marking of an initial SIP request.

## 6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

### 6.1. Trust Domain

Since a log me marker may cause a SIP entity to log the SIP header and body of a request or response, the log me marker should be removed at a trust domain boundary. If a prior agreement to log sessions exists with the net hop network then the log me marker might not be removed.

### 6.2. Security Threats

#### 6.2.1. Log-me marking

The log me marker is not sensitive information, although it will sometimes be inserted because a particular device is experiencing problems.

The presence of a log me marker will cause some SIP entities to log signalling. Therefore, this marker must be removed at the earliest opportunity if it has been incorrectly inserted.

Activating a debug mode affects the operation of a terminal, therefore it must be supplied by an authorized server to an authorized terminal, it must not be altered in transit, and it must not be readable by an unauthorized third party.

Logged signalling is privacy-sensitive data, therefore it must be passed to an authorized server, it must not be altered in transit, and it must not be readable by an unauthorized third party.

### 6.2.2. Sending logged information

A SIP entity that has logged information should encrypt it, such that it can be decrypted only by the debug server, before sending it to a debug server in order to protect the content of logs from a third party.

## 7. References

### 7.1. Normative References

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

### 7.2. Informative References

- [I-D.ietf-insipid-session-id-reqts]  
Jones, P., Salgueiro, G., Polk, J., Liess, L., and H. Kaplan, "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks", draft-ietf-insipid-session-id-reqts-07 (work in progress), June 2013.
- [RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [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.
- [RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.
- [RFC3428] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.
- [RFC3903] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

[RFC6086] Holmberg, C., Burger, E., and H. Kaplan, "Session Initiation Protocol (SIP) INFO Method and Package Framework", RFC 6086, January 2011.

#### Appendix A. Additional Stuff

This becomes an Appendix.

#### Author's Address

Peter Dawes  
Vodafone Group  
The Connection  
Newbury, Berkshire RG14 2FN  
UK

Phone: +44 7717 275009  
Email: [peter.dawes@vodafone.com](mailto:peter.dawes@vodafone.com)



Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: July 27, 2014

P. Dawes  
Vodafone Group  
January 23, 2014

Solutions for Marking SIP Messages to be Logged  
draft-dawes-insipid-logme-solutions-00

Abstract

SIP networks use signalling monitoring tools to diagnose user reported problem and for regression testing if network or user agent software is upgraded. As networks grow and become interconnected, including connection via transit networks, it becomes impractical to predict the path that SIP signalling will take between user agents, and therefore impractical to monitor SIP signalling end-to-end.

This draft describes potential solutions to meet requirements for adding an indicator to the SIP protocol which can be used to mark signalling as of interest to logging. Such marking will typically be applied as part of network testing controlled by the network operator and not used in regular user agent signalling. However, such marking can be carried end-to-end including the SIP user agents, even if a session originates and terminates in different networks.

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 July 27, 2014.

Copyright Notice

Copyright (c) 2014 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. Requirements Language . . . . .	3
3. Motivating Scenario . . . . .	3
4. Skeleton Diagnostic Procedure . . . . .	4
5. Potential Solutions for Log Me Marking . . . . .	5
5.1. Functionality Common to all Solutions . . . . .	5
5.1.1. Starting and Stopping log-me marking . . . . .	5
5.1.2. Configuration for log-me marking . . . . .	5
5.1.3. End Points of Logme Marking . . . . .	6
5.1.3.1. Originating and Terminating User Agent . . . . .	6
5.1.3.2. Originating Edge Proxy and Terminating Edge Proxy . . . . .	7
5.1.4. Maintaining State . . . . .	7
5.1.5. Missing Log-me Marker in Dialog Being Logged . . . . .	9
5.1.6. Logging Multiple Simultaneous Dialogs . . . . .	10
5.1.7. Forked Requests and Back to Back User Agents . . . . .	10
5.1.7.1. Propagating the Log-me Marker in Forked Requests . . . . .	10
5.1.7.2. B2BUA processing of Log-me Marker . . . . .	10
5.1.8. Sending logs to a debug server . . . . .	10
5.1.8.1. Protecting logs . . . . .	10
5.2. Solution A: Log-Me header field . . . . .	10
5.2.1. Log-me Marker . . . . .	10
5.2.2. Identifying test cases . . . . .	11
5.2.3. Collecting logged information at a debug server . . . . .	11
5.2.4. Examples . . . . .	11
5.3. Solution B: New Value for purpose header field parameter in Call-Info: . . . . .	12
5.3.1. Log-me Marker . . . . .	12
5.3.2. Identifying test cases . . . . .	13
5.4. Solution C: New 'debug' header field parameter to be used in Session-ID header field . . . . .	13
5.4.1. Log-me Marker . . . . .	13
5.4.2. Identifying test cases . . . . .	14
5.5. Comparison of Potential Solutions . . . . .	14
6. Security Considerations . . . . .	14
6.1. Trust Domain . . . . .	15
6.2. Security Threats . . . . .	15

6.2.1. Log-me marking . . . . .	15
6.2.2. Debug server address . . . . .	15
6.2.3. Sending logged information . . . . .	15
7. References . . . . .	16
7.1. Normative References . . . . .	16
7.2. Informative References . . . . .	16
Appendix A. Additional Stuff . . . . .	17
Author's Address . . . . .	17

## 1. Introduction

If users experience problems with setting up sessions using SIP, their service provider needs to find out why by examining the SIP signalling. Also, if network or user agent software or hardware is upgraded regression testing is needed. Such diagnostics apply to a small proportion of network traffic and can apply end-to-end, even if signalling crosses several networks possibly belonging to several different network operators. It may not be possible to predict the path through those networks in advance, therefore a mechanism is needed to mark a session as being of interest to enable SIP entities along the signalling path to provide diagnostic logging. This draft describes potential solutions to meet the requirements for such a 'log me' marker for SIP signalling defined in draft-dawes-insipid-logme-reqs [I-D.dawes-insipid-logme-reqs].

## 2. Requirements Language

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 RFC 2119 [RFC2119].

## 3. Motivating Scenario

Signalling for SIP session setup can cross several networks, and these networks may not have common ownership and also may be in different countries. If a single operator wishes to perform regression testing or fault diagnosis end-to-end, the separate ownership of networks that carry the signalling and the explosion in the number of possible signalling paths through SIP entities from the originating to the terminating user make it impractical to pre-configure logging of an end-to-end SIP signalling of a session of interest.

The figure below shows an example of a signalling path through multiple networks.

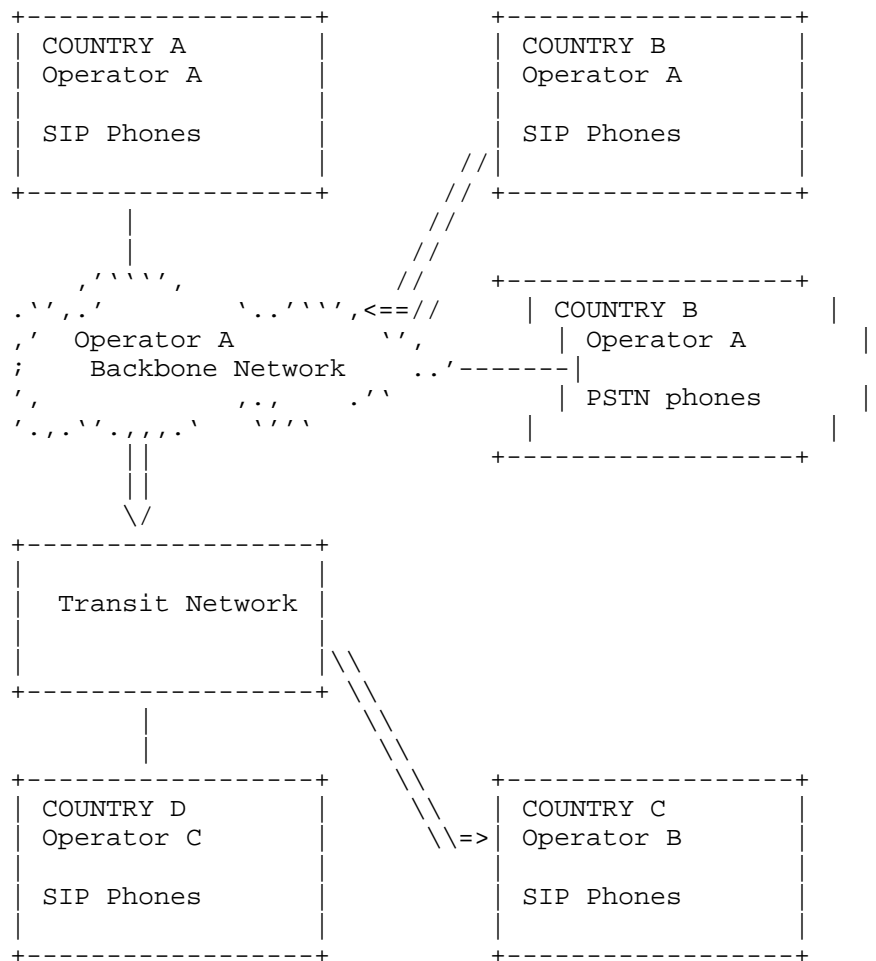


Figure 1: Example signalling path through multiple networks

#### 4. Skeleton Diagnostic Procedure

The skeleton diagnostic procedure is as follows:

- o The user's user agent is placed in debug mode. The user agent logs its own signalling and inserts a log me marker into SIP requests for session setup
- o All SIP entities that the signalling traverses, from the first proxy the user agent connects to at the edge of the network to the destination user agent, can detect that the log me marker is

present and can log SIP requests and responses that contain the marker if configured to do so.

- o Subsequent responses and requests in the same dialog are logged.
- o Logging stops, either because the dialog has ended or because a 'stop event', typically expiry of a certain amount of time, occurred
- o The user's user agent and any other SIP entity that has logged signalling sends logs to a server that is co-ordinating diagnostics.

## 5. Potential Solutions for Log Me Marking

This clause describes and compares potential solutions to the log-me requirements described in draft-dawes-insipid-logme-reqs [I-D.dawes-insipid-logme-reqs].

### 5.1. Functionality Common to all Solutions

#### 5.1.1. Starting and Stopping log-me marking

A proxy or user agent needs to determine when it needs to log-me mark a SIP request or response. A user agent or proxy log-me marks a request or response for two reasons: either it is configured to do so or it has detected that a dialog is being log-me marked and maintains state to ensure that all requests and responses in the dialog are log-me marked. A regression test might be configured to log-me mark all SIP dialogs created during a given time period whereas a troubleshooting test might be configured to mark a dialog based on criteria specific to a reported fault. When configuration has caused a user agent or proxy to start log-me marking requests and responses, marking continues until the dialog ends.

#### 5.1.2. Configuration for log-me marking

Configuration of a user agent or proxy to perform log-me marking can be done in any way that is convenient to the configured entity. For example, an XML file might be used to list conditions for starting and stopping based on time.

```
<start>09:00:00</start>  
<stop>09:10:00</stop>
```

Figure 2: Simple example log-me configuration

Logging is on a per-dialog basis and individual logs are differentiated by their test identifier (test identifier is described in draft-dawes-insipid-logme-reqs [I-D.dawes-insipid-logme-reqs]). Therefore, an individual log for an individual dialog is closed when that dialog ends. Logging is typically done separately from regular operation, which means that tests can be designed to be short enough to troubleshoot quickly and to limit the size of individual logs. If logging is configured so that everything is logged for a specified number of minutes then several separate dialogs might start and finish meaning that several logs may be generated, each one distinguished by its test identifier.

#### 5.1.3. End Points of Logme Marking

Log-me marking is initiated on a dialog creating side controlled by configuration. The dialog terminating side detects an incoming log-me marker and reacts accordingly.

##### 5.1.3.1. Originating and Terminating User Agent

In the simplest case, an originating user agent will insert a log-me marker in the dialog-creating SIP request and all subsequent SIP requests within that dialog. The log-me marker is carried to the terminating user agent and the terminating user agent echoes the log-me marker in responses. If the terminating user agent sends an in-dialog request on a dialog that is being log-me marked, it inserts a log-me marker and the originating user agent echoes the log-me marker in responses. This basic case suggests the following principles:

- o The originating user agent is configured for debug
- o The terminating user agent is not configured for debug and cannot initiate log-me marking.
- o The originating user agent logs its own signalling and inserts a log me marker into the dialog-creating SIP request and subsequent in-dialog SIP requests.
- o The terminating user agent can detect that a dialog is of interest to logging by the existence of a log-me marker in an incoming dialog-creating SIP request.
- o The terminating user agent must echo a log-me marker in responses to a SIP request that included a log-me marker.
- o If the terminating user agent has detected that a dialog is being log-me marked, it inserts a log-me marker in any in-dialog SIP requests that it sends.

#### 5.1.3.2. Originating Edge Proxy and Terminating Edge Proxy

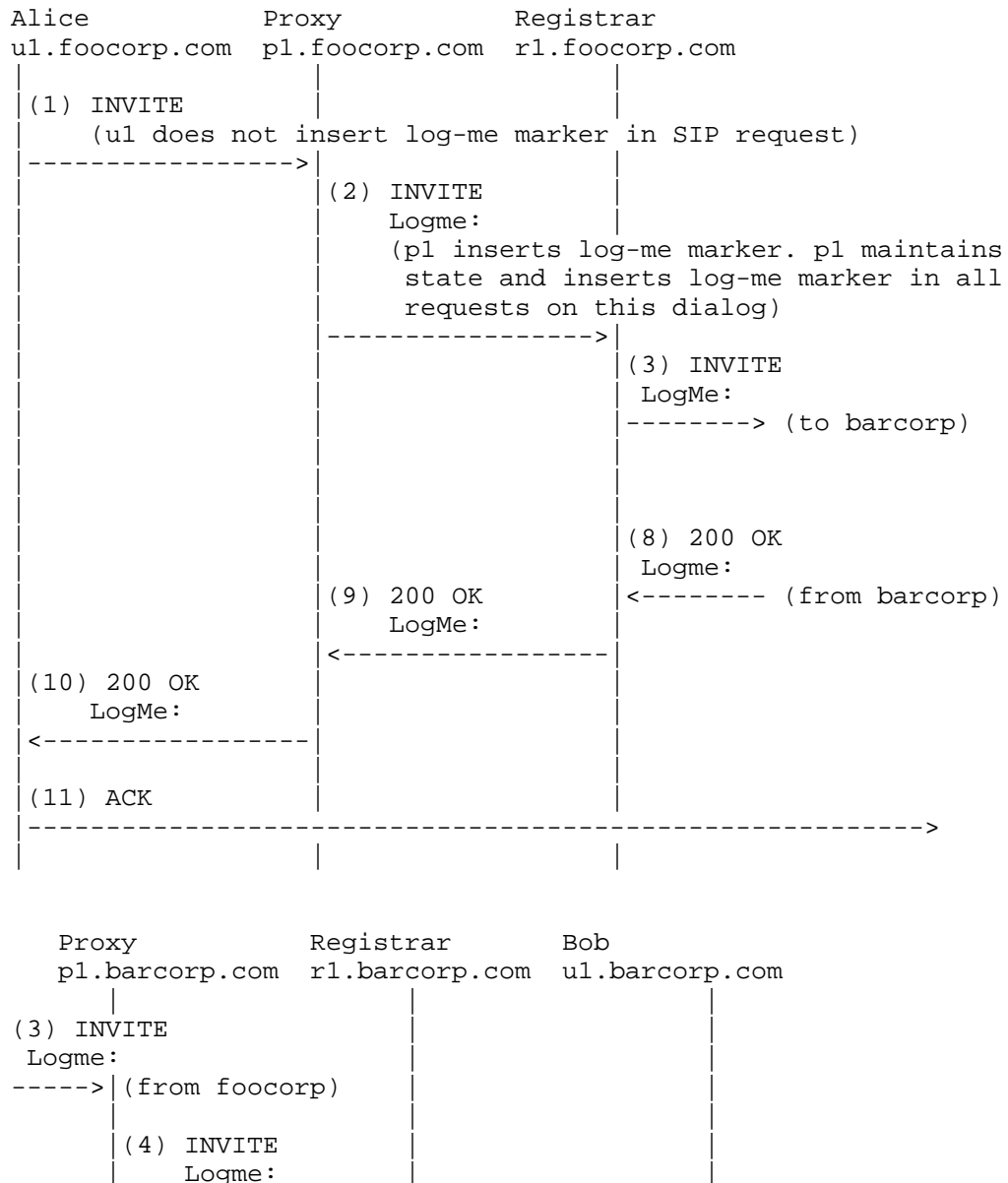
Some user agents might not support log-me marking. In order to test sessions involving such user agents, the log-me marker is inserted by edge proxies on the originating and terminating sides. The log-me marker is carried to the terminating user agent but the terminating user agent is not able to echo the log-me marker in responses to that request. Therefore the terminating edge proxy inserts a log-me marker in responses to the request. Likewise, if the terminating user agent sends an in-dialog request, the terminating edge proxy inserts a log-me marker and the originating edge proxy echoes the log-me marker in responses to that request. This case suggests the following principles:

- o The originating edge proxy is configured for debug.
- o The terminating edge proxy is not configured for debug and cannot initiate log-me marking.
- o The originating edge proxy logs its own signalling and inserts a log me marker into SIP requests for session setup.
- o The terminating edge proxy can detect that a dialog is of interest to logging by the existence of a log-me marker in an incoming SIP request.
- o The terminating edge proxy must echo a log-me marker in responses to a SIP request that included a log-me marker.
- o If the terminating edge proxy has detected that a dialog is being log-me marked, it inserts a log-me marker in in-dialog SIP requests from the terminating user agent.
- o The originating edge proxy echoes the log-me marker in responses to in-dialog requests received from the terminating side.

#### 5.1.4. Maintaining State

If a proxy inserts a log-me marker in a SIP request (because a user agent did not) then it must ensure that a log-me marker is also inserted in responses to that request. A proxy on the terminating side that receives a SIP request with a log-me marker may also ensure that responses to that request contain a log-me marker by inserting one if the terminating user agent did not. Entities that perform this log-me marking or checking must maintain a record of which dialogs are being log-me marked.

In the figure below, the edge proxy in the originating network maintains state to ensure log-me marking of SIP requests and in the terminating network the registrar maintains state to ensure log-me marking of SIP responses. Such behaviour is useful for logging if end devices do not insert or echo a log-me marker.





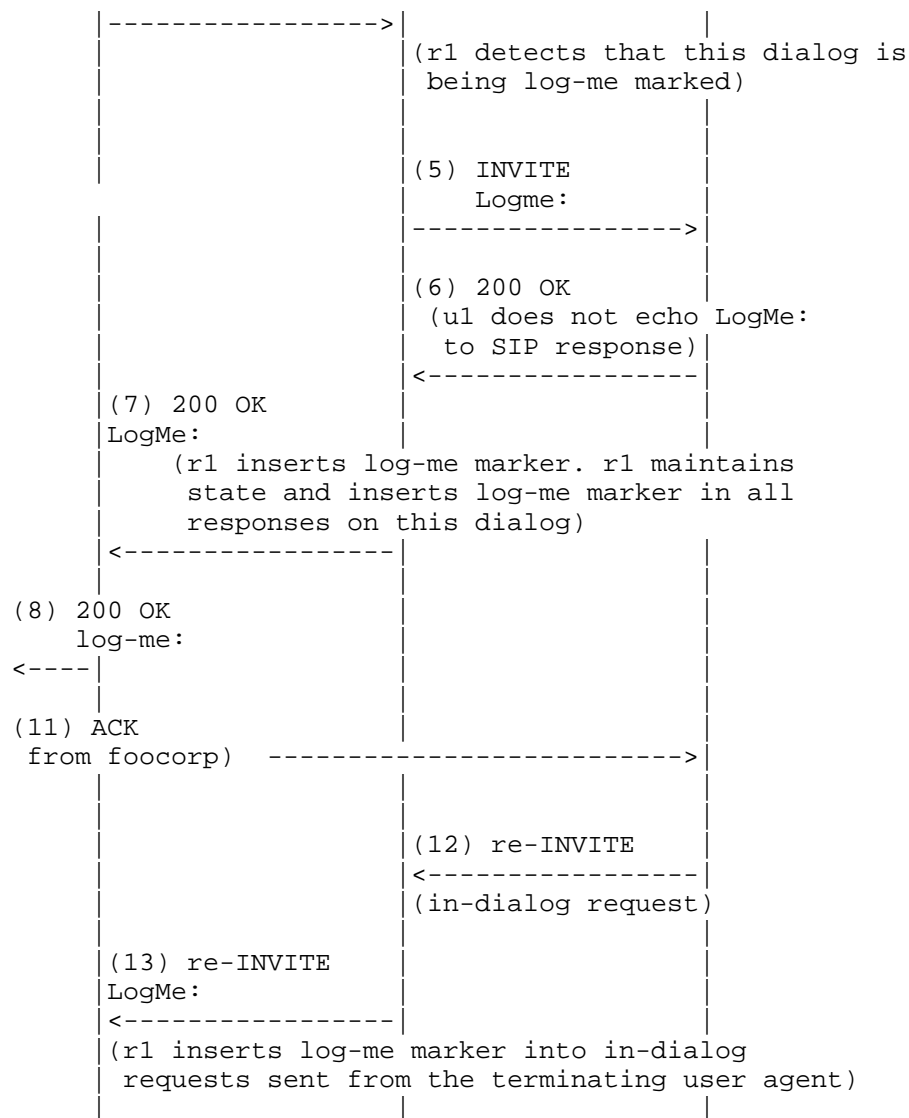


Figure 3: Maintaining state for log-me marking

## 5.1.5. Missing Log-me Marker in Dialog Being Logged

A terminating user agent or terminating edge proxy that has been echoing markers in responses for a given dialog might receive a SIP request that has not been log-me marked. Since log-me marking is done per dialog, this is an error. In such cases, the proxy should

consider log-me marking to have ended and not mark a response to the unmarked request, responses to subsequent requests in the dialog, or in-dialog requests sent from the terminating side. Similarly, log-me marking that begins mid-dialog is an error case and the terminating user agent or edge proxy must not log-me mark responses to the marked request, responses to subsequent requests in the dialog, or in-dialog requests from the terminating side.

#### 5.1.6. Logging Multiple Simultaneous Dialogs

An originating or terminating user agent and SIP entities on the signaling path can log multiple SIP dialogs simultaneously, these dialogs can be differentiated by their test identifier.

#### 5.1.7. Forked Requests and Back to Back User Agents

##### 5.1.7.1. Propagating the Log-me Marker in Forked Requests

Log-me marking is copied into forked requests.

##### 5.1.7.2. B2BUA processing of Log-me Marker

A B2BUA should act on the terminating side as described for a terminating user agent and therefore log marked incoming requests, echo log-me marking in responses to log-me marked requests, and log-me mark in-dialog SIP requests that it sends if the dialog is being log-me marked.

A B2BUA should act on the originating side as described for an originating user agent and therefore mark SIP requests if and only if configured to do so, and echo log-me marking in responses to in-dialog requests received from the terminating side.

#### 5.1.8. Sending logs to a debug server

##### 5.1.8.1. Protecting logs

A SIP entity that has logged information should encrypt it, such that it can be decrypted only by the debug server, before sending in order to protect the content of logs from a third party.

#### 5.2. Solution A: Log-Me header field

##### 5.2.1. Log-me Marker

A new SIP header field, e.g. 'LogMe:', is defined as the log-me marker. The LogMe header field is defined with one header field

parameter that contains a free-text name of the test case being performed.

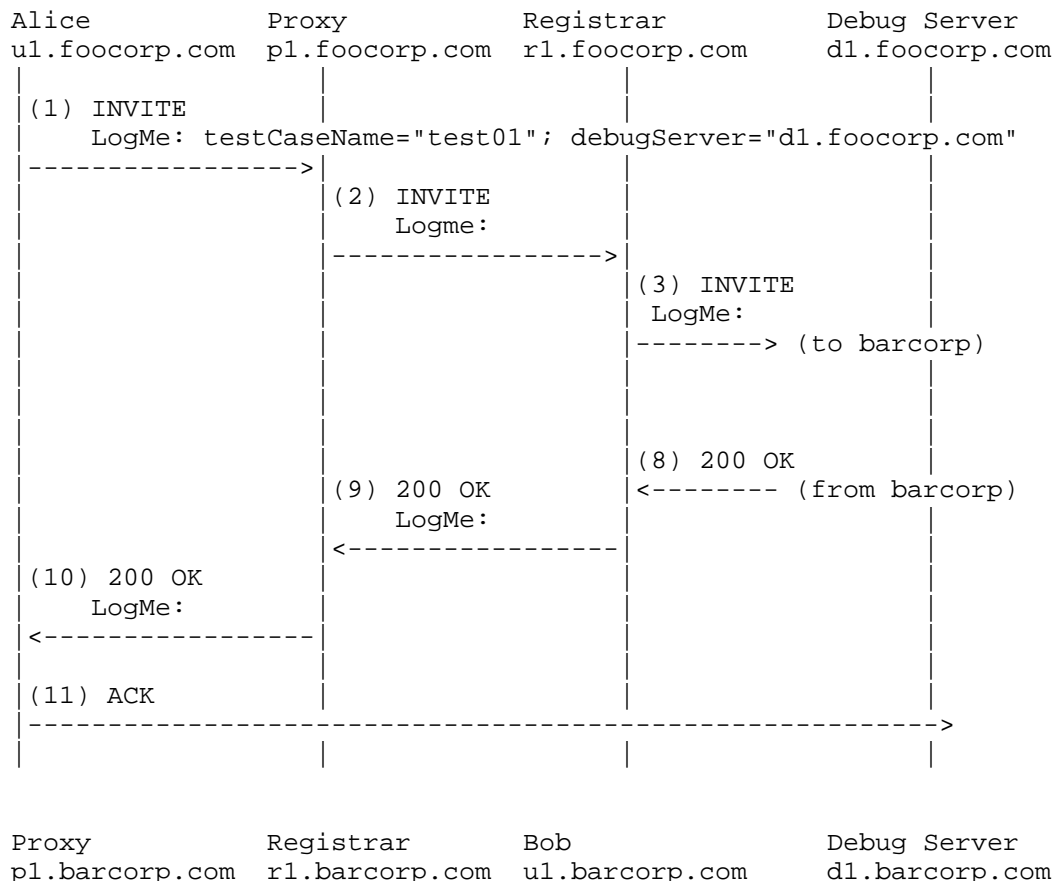
#### 5.2.2. Identifying test cases

The new header field has a parameter that contains a free-text name of the test case being performed which acts as the test case identifier (test identifier is described in draft-dawes-insipid-logme-reqs [I-D.dawes-insipid-logme-reqs]).

#### 5.2.3. Collecting logged information at a debug server

User agents and SIP proxies may send logged information to a debug server.

#### 5.2.4. Examples



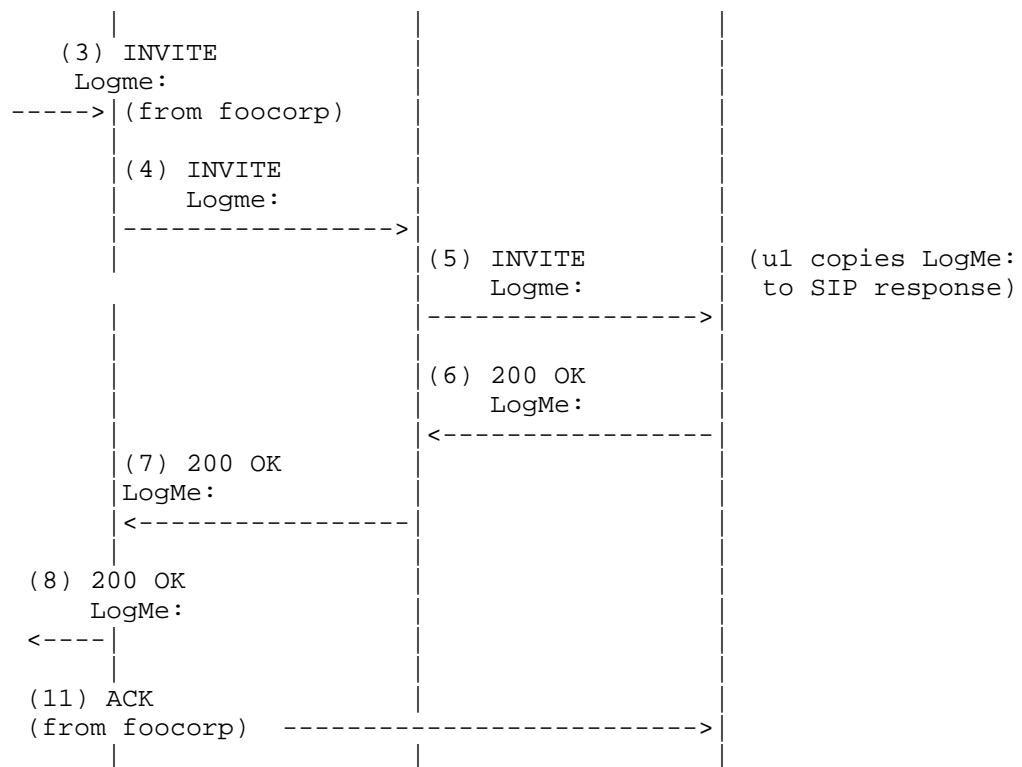


Figure 4: Signalling example for the LogMe header field solution

### 5.3. Solution B: New Value for purpose header field parameter in Call-Info:

#### 5.3.1. Log-me Marker

A new value is defined for the purpose header field parameter used in Call-Info header field as the log-me marker.

The Call-Info: header field is defined in clause 20.9 of RFC 3261 [RFC3261].

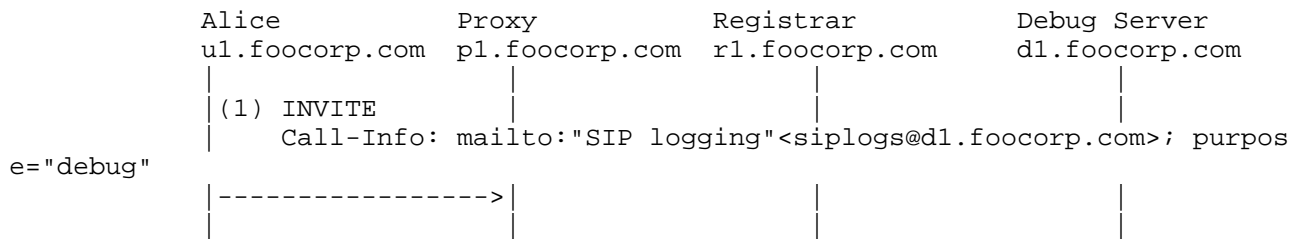


Figure 5: Signalling example for the Call-Info: purpose parameter solution

The Call-Info: header field can be included in methods INVITE, OPTIONS, REGISTER (Table 2: Summary of header fields, A--O in RFC 3261 [RFC3261] clause 20.1), INFO (RFC 6086 [RFC6086]), MESSAGE (RFC 3428 [RFC3428]), PUBLISH (RFC 3903 [RFC3903]), and UPDATE (RFC 3311 [RFC3311]), and in responses to those methods. Call-Info: header field cannot be included in methods NOTIFY, SUBSCRIBE, PRACK, or REFER.

### 5.3.2. Identifying test cases

The Call-Info: header field has no protocol element that can be used to indicate the test case name, therefore in this solution the test case is identified by the Session-ID header field (described in I-D .ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts]).

## 5.4. Solution C: New 'debug' header field parameter to be used in Session-ID header field

### 5.4.1. Log-me Marker

A new header field parameter called debug is defined to be used with the Session-ID header field (described in I-D.ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts]).

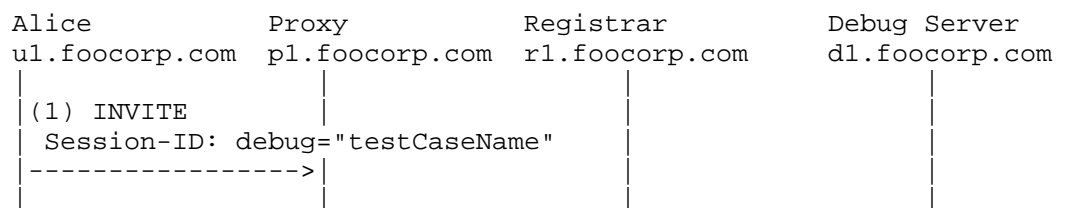


Figure 6: Signalling example for the Session-ID: header field parameter solution

#### 5.4.2. Identifying test cases

In this solution the test case is identified by the Session-ID header field (described in I-D.ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts]).

#### 5.5. Comparison of Potential Solutions

The table below summarizes the features of each potential solution. Other solutions are not excluded.

	Solution	Summary
A	Log-Me: header field	Specify a new SIP header field. Could be included in all SIP requests and responses. All behaviour including proxy handling in terms of add, delete, modify, and read, and which requests may or may not include the header field must be defined.
B	New value for the purpose parameter of the Call-Info header field e.g. "debug"	Rules for including, reading, modifying etc. are already defined by Call-Info. Call-Info cannot be inserted in all requests and responses, but can be included for the SIP methods of most interest to debugging and regression testing. No element to hold a test case name so test case is identified by the Session-ID header field.
C	New header field parameter for Session-ID header field e.g. debug	Might be viewed as a reason to remove the Session-ID header field, which would violate the Session-ID requirement: "REQ3: The solution must require that the identifier, if present, pass unchanged through SIP B2BUAs or other intermediaries." in I-D.ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts]

Table 1: Summary comparison of potential solutions

#### 6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

### 6.1. Trust Domain

Since a log me marker may cause a SIP entity to log the SIP header and body of a request or response, the log me marker should be removed at a trust domain boundary. If a prior agreement to log sessions exists with the net hop network then the log me marker might not be removed.

### 6.2. Security Threats

#### 6.2.1. Log-me marking

The log me marker is not sensitive information, although it will sometimes be inserted because a particular device is experiencing problems.

The presence of a log me marker will cause some SIP entities to log signalling. Therefore, this marker must be removed at the earliest opportunity if it has been incorrectly inserted.

Activating a debug mode affects the operation of a user agent, therefore it must be supplied by an authorized server to an authorized user agent, it must not be altered in transit, and it must not be readable by an unauthorized third party.

Logged signalling is privacy-sensitive data, therefore it must be passed to an authorized server, it must not be altered in transit, and it must not be readable by an unauthorized third party.

#### 6.2.2. Debug server address

Log me marking may include the address of a debug server in the form of a URL. In order to prevent sending of logs to an unauthorised server a SIP entity that supports logging should authenticate the debug server, for example by referring to a statically configured white list of allowed destination domains.

#### 6.2.3. Sending logged information

A SIP entity that has logged information should encrypt it, such that it can be decrypted only by the debug server, before sending it to a debug server in order to protect the content of logs from a third party.

## 7. References

### 7.1. Normative References

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

### 7.2. Informative References

- [I-D.dawes-insipid-logme-reqs]  
Dawes, P., "Requirements for Marking SIP Messages to be Logged", draft-dawes-insipid-logme-reqs-00 (work in progress), January 2014.
- [I-D.ietf-insipid-session-id-reqts]  
Jones, P., Salgueiro, G., Polk, J., Liess, L., and H. Kaplan, "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks", draft-ietf-insipid-session-id-reqts-07 (work in progress), June 2013.
- [RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [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.
- [RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.
- [RFC3428] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.
- [RFC3903] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.



[RFC6086] Holmberg, C., Burger, E., and H. Kaplan, "Session Initiation Protocol (SIP) INFO Method and Package Framework", RFC 6086, January 2011.

#### Appendix A. Additional Stuff

This becomes an Appendix.

#### Author's Address

Peter Dawes  
Vodafone Group  
The Connection  
Newbury, Berkshire RG14 2FN  
UK

Phone: +44 7717 275009  
Email: [peter.dawes@vodafone.com](mailto:peter.dawes@vodafone.com)

Network Working Group  
Internet Draft  
Intended status: Standards Track  
Expires: August 14, 2014

P. Jones (Ed.)  
C. Pearce  
J. Polk (Ed.)  
G. Salgueiro  
Cisco Systems  
February 14, 2014

End-to-End Session Identification in IP-Based Multimedia  
Communication Networks  
draft-ietf-insipid-session-id-05

Abstract

This document describes an end-to-end Session Identifier for use in IP-based multimedia communication systems that enables endpoints, intermediate devices, and management systems to identify a session end-to-end, associate multiple endpoints with a given multipoint conference, track communication sessions when they are redirected, and associate one or more media flows with a given communication session.

This document also describes a backwards compatibility mechanism for an existing "in the wild" session identifier implementation that is sufficiently different from the procedures defined in this document.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on August 14, 2014.

## Copyright Notice

Copyright (c) 2014 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. Conventions used in this document.....	3
3. Session Identifier Requirements and Use Cases.....	4
4. Constructing and Conveying the Session Identifier.....	4
4.1. Constructing the Session Identifier.....	4
4.2. Conveying the Session Identifier.....	4
5. Transmitting the Session Identifier in SIP.....	6
6. Endpoint Behavior.....	7
7. Processing by Intermediaries.....	8
8. Associating Endpoints in a Multipoint Conference.....	9
9. Various Call Flow Operations Utilizing the Session ID.....	10
9.1. Basic Session ID Construction with 2 UUIDs.....	10
9.2. Basic Call Transfer using REFER.....	11
9.3. Basic Call Transfer using reINVITE.....	13
9.4. Single Focus Conferencing.....	14
9.5. Single Focus Conferencing using WebEx.....	16
9.6. Cascading Conference Bridge Support for the Session ID...17	
9.6.1. Calling into Cascaded Conference Bridge for the Session ID.....	18
9.7. Basic 3PCC for two UAs.....	19
9.8. Session ID Handling in 100 Trying SIP Response and CANCEL Request.....	20
9.8.1. Session ID Handling in a 100 Trying SIP Response....	20
9.8.2. Session ID in a CANCEL SIP Request.....	21
9.9. Session ID in an out-of-dialog REFER Transaction.....	22
10. Compatibility with a Previous Implementation.....	23
11. Security Considerations.....	24
12. IANA Considerations.....	25
12.1. Registration of the "Session-ID" Header Field.....	25
12.2. Registration of the "remote" Parameter.....	25
13. Acknowledgments.....	25
14. References.....	26
14.1. Normative References.....	26
14.2. Informative References.....	26

Author's Addresses.....	27
-------------------------	----

## 1. Introduction

IP-based multimedia communication systems like SIP [RFC3261] and H.323 [H.323] have the concept of a "call identifier" that is globally unique. The identifier is intended to represent an end-to-end communication session from the originating device to the terminating device. Such an identifier is useful for troubleshooting, session tracking, and so forth.

For several reasons, however, the current call identifiers defined in SIP and H.323 are not suitable for end-to-end session identification. A fundamental issue in protocol interworking is the fact that the syntax for the call identifier in SIP and H.323 is different. Thus, if both protocols are used in a call, it is impossible to exchange the call identifier end-to-end.

Another reason why the current call identifiers are not suitable to identify a session end-to-end is that, in real-world deployments, devices like session border controllers often change the session signaling as it passes through the device, including the value of the call identifier. While this is deliberate and useful, it makes it very difficult to track a session end-to-end.

This draft presents a new identifier, referred to as the Session Identifier, or "Session ID", and associated syntax intended to overcome the issues that exist with the currently defined call identifiers. The proposal in this document attempts to comply with the requirements specified in [I-D.ietf-insipid-session-id-reqts]. This proposal also has capabilities not mentioned in [I-D.ietf-insipid-session-id-reqts], shown in call flows in section 9. Additionally, this proposal attempts to account for a previous, proprietary version of a SIP Session ID header, proposing a backwards compatibility approach, described in section 10.

## 2. Conventions used in this document

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 RFC 2119 [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

The terms "Session Identifier" and "Session ID" refer to the value of the identifier, whereas "Session-ID" refers to the header used to convey the identifier.

### 3. Session Identifier Requirements and Use Cases

Requirements and use cases for the end-to-end Session Identifier, along with a definition of "session identifier" and "communication session", can be found in [I-D.ietf-insipid-session-id-reqts].

### 4. Constructing and Conveying the Session Identifier

#### 4.1. Constructing the Session Identifier

The Session Identifier comprises two UUIDs [RFC4122], with each UUID representing one of the endpoints participating in the session.

The version number in the UUID indicates the manner in which the UUID is generated, such as using random values or using the MAC address of the endpoint. To satisfy the requirement that no user or device information be conveyed, endpoints SHOULD generate version 4 (random) or version 5 (SHA-1) UUIDs.

When generating a version 5 UUID, endpoints or intermediaries MUST utilize the following "name space ID" (see Section 4.3 of [RFC4122]):

```
uuid_t NameSpace_SessionID = {  
    /* a58587da-c93d-11e2-ae90-f4ea67801e29 */  
    0xa58587da,  
    0xc93d,  
    0x11e2,  
    0xae, 0x90, 0xf4, 0xea, 0x67, 0x80, 0x1e, 0x29  
}
```

Further, the "name" to utilize for version 5 UUIDs is the concatenation of the Call-ID header value and the "tag" parameter that appears on the "From" or "To" line associated with the device for which the UUID is created. Once an endpoint generates a UUID for a session, the UUID never changes, even if values originally used as input into its construction change over time.

Intermediaries that insert a Session-ID header into a SIP message on behalf of a sending User Agent MUST utilize version 5 UUIDs to ensure that UUIDs for the communication session are consistently generated. If an intermediary does not know the tag value for an endpoint, the intermediary MUST NOT attempt to generate a UUID for that endpoint. Note that if an intermediary is stateless and the endpoint on one end of the call is replaced with another endpoint due to some service interaction, the values used to create the UUID might change and, if so, the intermediary will compute a different UUID.

#### 4.2. Conveying the Session Identifier

The SIP user agent (UA) initially transmitting the SIP request, i.e., a User Agent Client (UAC), will create a UUID and transmit that to

the ultimate destination UA. Likewise, the responding UA, i.e., a User Agent Server (UAS), will create a UUID and transmit that to the first UA. These two distinct UUIDs form what is referred to as the Session Identifier and is represented in this document in set notation of the form {A,B}, where A is UUID value from the UA transmitting a message and B is the UUID value of the intended recipient of the message, i.e., not an intermediary server along the signaling path. The Session Identifier {A,B} is equal to the Session Identifier {B,A}.

In the case where only one UUID is known, such as when a UA first initiates a SIP request, the Session ID would be {A,N}, where "A" represents the UUID value transmitted by the UA and "N" is what is referred to as the null UUID (see section 5).

Since SIP sessions are subject to any number of service interactions, SIP INVITE messages might be forked as sessions are established, and since conferences might be established or expanded with endpoints calling in or the conference focus calling out, the construction of the Session Identifier as a set of UUIDs is important.

To understand this better, consider that a UA participating in a communication session might be replaced with another, such as the case where two "legs" of a call are joined together by a PBX. Suppose that UA A and UA B both call UA C. Further suppose that UA C uses a local PBX function to join the call between itself and UA A with the call between itself and UA B, resulting in a single remaining call between UA A and UA B. This merged call can be identified using two UUID values assigned by each entity in the communication session, namely {A,B} in this example.

In the case of forking, UA A might send an INVITE that gets forked to five different UAs, as an example. A means of identifying each of these separate communication sessions is needed and allowing the set of {A, B1}, {A, B2}, {A, B3}, {A, B4}, and {A, B5} makes this possible.

For conferencing scenarios, it is also useful to have a two-part Session Identifier where the conference focus specifies one UUID for each conference participant. This will allow for correlation among the participants in a single conference. For example, in a conference with three participants, the Session Identifiers might be {A,M}, {B,M}, and {C,M}, where "M" is assigned by the conference focus.

How a device acting on Session Identifiers stores, processes, or utilizes the Session Identifier is outside the scope of this document.

## 5. Transmitting the Session Identifier in SIP

Each session initiated or accepted MUST have a unique local UA-generated UUID. This value MUST remain unchanged throughout the duration of the session.

A SIP UA MUST convey its Session Identifier UUID in all transmitted messages by including the Session-ID header. The Session-ID header has the following ABNF [RFC5234] syntax:

```
session-id      = "Session-ID" HCOLON local-uuid
                  *(SEMI sess-id-param)

local-uuid      = sess-uuid / null

remote-uuid     = sess-uuid / null

sess-uuid       = 32(DIGIT / %x61-66) ;32 chars of [0-9a-f]

sess-id-param   = remote-param / generic-param

remote-param    = "remote" EQUAL remote-uuid

null            = 32("0")
```

The productions "SEMI", "EQUAL", and "generic-param" are defined in [RFC3261]. The production DIGIT is defined in [RFC5234].

The Session-ID header MUST NOT have more than one "remote" parameter. In the case where an entity compliant with this specification is interworking with an entity that implemented [I-D.kaplan-insipid-session-id], the "remote" parameter might be absent, but otherwise the remote parameter MUST be present. The details under which those conditions apply are described in Section 10. Except for backwards compatibility with [I-D.kaplan-insipid-session-id], the "remote" parameter MUST be present.

A special null UUID value composed of 32 zeros is required in certain situations. A null UUID is expected in the "remote" UUID of every initial standard SIP request since the initiating endpoint would not initially know the UUID value of the remote endpoint. This null value will get replaced by the ultimate destination UAS when that UA generates a UUID in response. One caveat is explained in Section 10 for a possible backwards compatibility case. A null UUID value is also returned by some intermediary devices that send provisional replies as a "local-uuid", as described in Section 6.

The "local-uuid" in the Session-ID header represents the UUID value of the UA transmitting the message. If the UA transmitting the message previously received a UUID value from its peer endpoint, it

MUST include that UUID as the "remote" parameter in each message it transmits. For example, a Session-ID header might appear like this:

```
Session-ID: ab30317f1a784dc48ff824d0d3715d86;  
           remote=47755a9de7794ba387653f2099600ef2
```

The UUID values are presented as strings of lower-case hexadecimal characters, with the most significant octet of the UUID appearing first.

## 6. Endpoint Behavior

To comply with this specification, non-intermediary SIP UAs MUST include a Session-ID header-value in all SIP messages transmitted as a part of a communication session. The UUID of the transmitter of the message MUST appear in the "local-uuid" portion of the Session-ID header-value. The UUID of the peer device, if known, MUST appear as the "remote" parameter following the transmitter's UUID. The null UUID value MUST be used the peer device's UUID is not known.

Once a UA allocates a UUID value for a communication session, the UA MUST NOT change that UUID value for the duration of the session, including when

- communication attempts are retried due to receipt of 4xx messages or request timeouts;
- the session is redirected in response to a 3xx message; or
- a session is transferred via a REFER message [RFC3515]; or
- a SIP dialog is replaced via an INVITE with Replaces [RFC3891].

A non-intermediary UA that receives a Session-ID header MUST take note of the first UUID value (i.e., the "local-uuid") that it receives in the Session-ID header and assume that that is the UUID of the peer endpoint within that communications session. UAs MUST include this received UUID value as the "remote" parameter when transmitting subsequent messages, making sure not to change this UUID value in the process of moving the value internally from the "local-uuid" field to the "remote-uuid" field.

It should be noted that messages received by a UA might contain a "local-uuid" value that does not match what the UA expected the far end UA's UUID to be. It is also possible for the UA to receive a "remote-uuid" value that does not match the UA's assigned UUID for the session. Either might happen as a result of service interactions by intermediaries and MUST NOT negatively affect the communication session. However, the UA may log this event for the purposes of troubleshooting.

A UA MUST assume that the UUID value of the peer UA MAY change at any time due to service interactions. If the UUID value of the peer UA changes, the UA MUST accept the new UUID as the peer's UUID and



include this new UUID as the "remote" parameter in any subsequent messages.

It is also important to note that if an intermediary in the network forks a session, the initiating UA may receive multiple responses back from different endpoints, each of which contains a different UUID ("local-uuid") value. UAs MUST take care to ensure that the correct UUID value is returned in the "remote" parameter when interacting with each endpoint.

A Multipoint Control Unit (MCU) is a special type of conferencing endpoint and is discussed in Section 8.

## 7. Processing by Intermediaries

Intermediaries MUST NOT alter the UUID values found in the Session-ID header, except as described in this section.

Intermediary devices that transfer a call, such as by joining together two different "call legs", MUST properly construct a Session-ID header that contains the correct UUID values and correct placement of those values. As described above, the recipient of any message initiated by the intermediary will assume that the first UUID value belongs to the peer endpoint.

If an intermediary receives a SIP message without a Session-ID header value, the intermediary MAY assign a "local-uuid" value to represent the sending endpoint and insert that value into all signaling messages on behalf of the sending endpoint. If the intermediary is aware of a "remote" value that identifies the receiving UA, it MUST insert that value if also inserting the "local-uuid" value.

Devices that initiate communication sessions following the procedures for third party call control MUST fabricate a UUID value that will be utilized only temporarily. Once the responding endpoint provides a UUID value in a response message, the temporary value MUST be discarded and replaced with the endpoint-provided UUID value. Refer to the third-party call control example for an illustration.

Whenever there is a UA that does not implement this specification communicating through a B2BUA, the B2BUA MAY become dialog stateful and insert a UUID value into the Session-ID header on behalf of the UA according to the rules stated in Section 6.

When intermediaries transmit provisional responses, such as 100 (Trying) or the 181 (Call Forwarding), and when the UUID of the destination UA is unknown, the sending intermediary MUST place the one known UUID in the "remote-uuid" field and set the "local-uuid" field to the null UUID value.

A CANCEL request sent by an intermediary that has not received a UUID from the destination UA MUST construct a Session-ID header value exactly like the INVITE to that UA, with the known "local-uuid" value for the initiating UA and the null UUID as the "remote-uuid" value for the destination UA.

If a SIP intermediary initiates a dialog between two UAs in a 3PCC scenario, the SIP request in the initial INVITE will have a non-null "local-uuid" value; call this temporary UUID X. The request will still have a null "remote-uuid" value; call this value N. The SIP server MUST be transaction stateful. The UUID pair in the INVITE will be {X,N}. A non-redirected or rejected response will have a UUID pair {A,X}. This transaction stateful, dialog initiating SIP server MUST replace its own UUID, i.e., X, with a null UUID (i.e., {A,N}) as expected by other UAs (see Section 9.7 for an example).

## 8. Associating Endpoints in a Multipoint Conference

Multipoint Control Units (MCUs) group two or more sessions into a single multipoint conference. MCUs, including cascaded MCUs, MUST utilize the same UUID value ("local-uuid" portion of the Session-ID header-value) with all participants in the conference. In so doing, each individual session in the conference will have a unique Session Identifier (since each endpoint will create a unique UUID of its own), but will also have one UUID in common with all other participants in the conference.

When creating a cascaded conferencing, an MCU MUST convey the UUID value to utilize for a conference via the "local-uuid" portion of the Session-ID header-value in an INVITE to a second MCU when using SIP to establish the cascaded conference. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. RFC 4579 [RFC4579] defines the "isfocus" Contact: header parameter just for this purpose. The initial MCU MUST include the UUID of that particular conference in the "local-uuid" of an INVITE to the other MCU(s) participating in that conference. Also included in this INVITE is an "isfocus" Contact header parameter identifying that this INVITE is coming from an MCU and that this UUID is to be given out in all responses from UAs into those MCUs participating in this same conference. This ensures a single UUID is common across all participating MCUs of the same conference, but is unique between different conferences.

Intermediary devices, such as proxies or session border controllers, or network diagnostics equipment might assume that when they see two or more sessions with different Session Identifiers, but with one UUID in common, that the sessions are part of the same conference. However, the assumption that two sessions having one common UUID being part of the same conference is not always correct. In a SIP forking scenario, for example, there might also be what appears to be multiple sessions with a shared UUID value; this is intended. The

desire is to allow for the association of related sessions, regardless of whether a session is forked or part of a conference.

## 9. Various Call Flow Operations Utilizing the Session ID

Seeing something frequently makes understanding easier. With that in mind, we include several call flow examples with the initial UUID and the complete Session ID indicated per message, as well as when the Session ID changes according to the rules within this document during certain operations/functions.

This section is for illustrative purposes only and is non-normative. In the following flows, RTP refers to the Real-time Transport Protocol [RFC3550].

"N" represents a null UUID in those examples in this section that have an N.

### 9.1. Basic Session ID Construction with 2 UUIDs

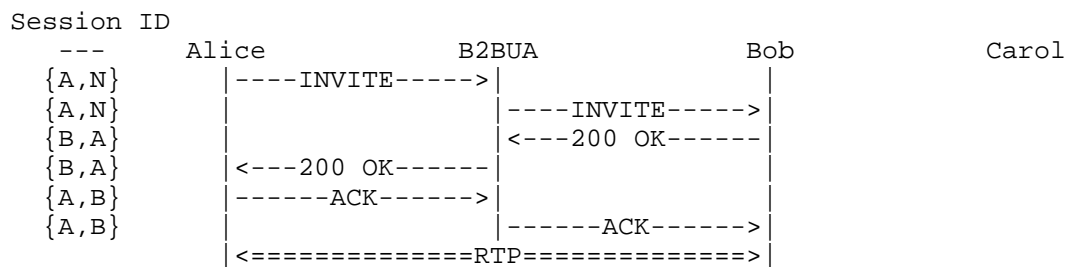


Figure 1 - Session ID Creation when Alice calls Bob

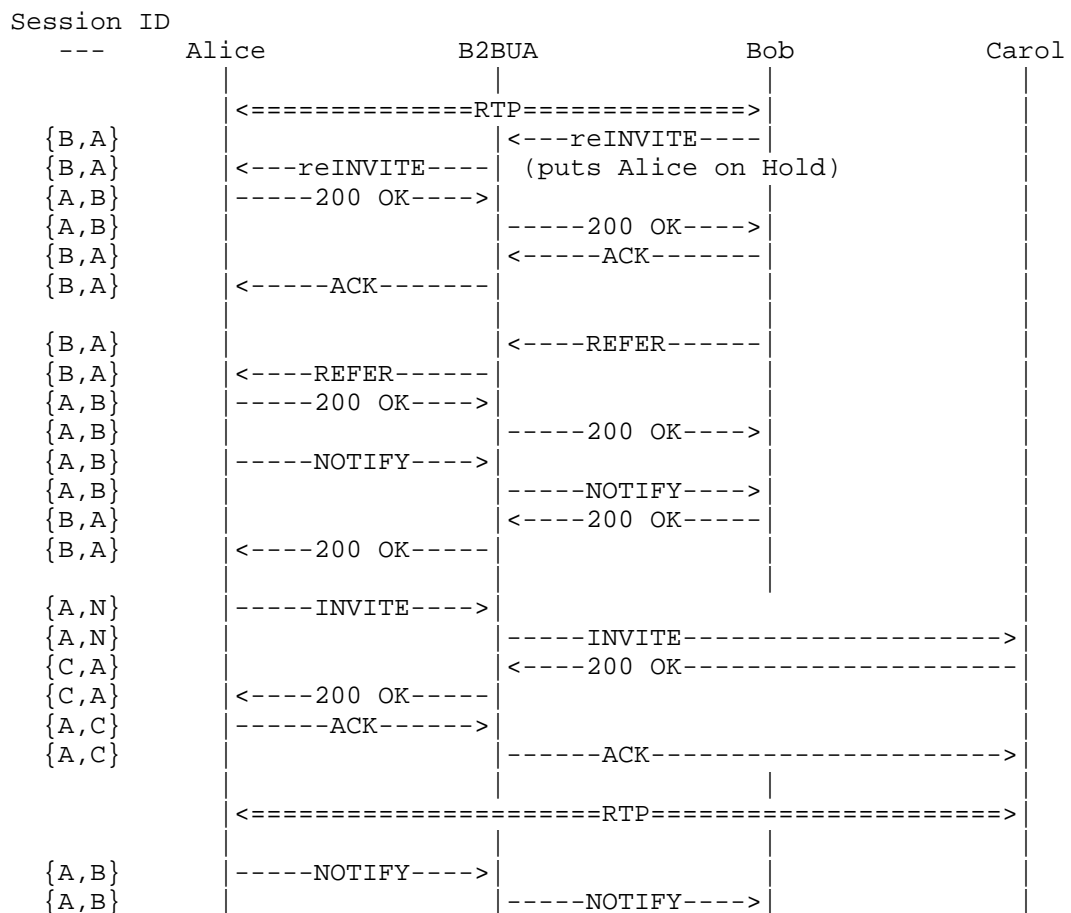
General operation of this example:

- o UA-Alice populates the "local-uuid" portion of the Session-ID header-value.
- o UA-Alice sends its UUID in the SIP INVITE, and populates the "remote" parameter with a null value (32 zeros).
- o B2BUA receives an INVITE with both a "local-uuid" portion of the Session-ID header-value from UA-Alice as well as the null "remote" UUID, and transmits the INVITE towards UA-Bob with an unchanged Session-ID header-value.
- o UA-Bob receives Session-ID and generates and replaces its "local-uuid" portion of the Session-ID header-value UUID to construct the whole/complete Session-ID header-value, at the same time transferring Alice's UUID unchanged to the "remote-uuid" portion of the Session-ID header-value in the 200 OK SIP response.

- o B2BUA receives the 200 OK response with a complete Session-ID header-value from UA-Bob, and transmits 200 OK towards UA-Alice with an unchanged Session-ID header-value.
- o UA-Alice, upon reception of the 200 OK from the B2BUA, transmits the ACK towards the B2BUA. The construction of the Session-ID header-value in this ACK is that of Alice's UUID is the "local-uuid", and Bob's UUID populates the "remote-uuid" portion of the header-value.
- o B2BUA receives the ACK with a complete Session-ID header-value from UA-Alice, and transmits ACK towards UA-Bob with an unchanged Session-ID header-value.

## 9.2. Basic Call Transfer using REFER

From the example built within Section 9.1 (the basic session ID establishment), we proceed to this 'Basic Call Transfer using REFER' example.



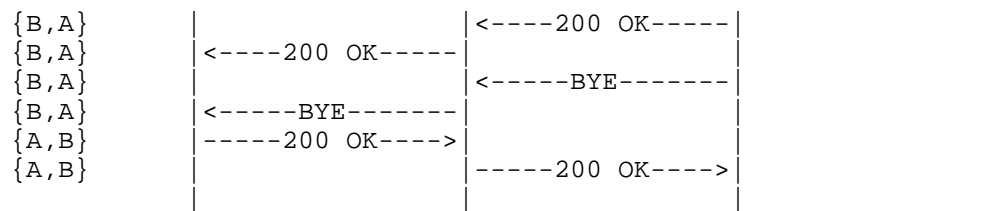


Figure 2 - Call Transfer using REFER

General operation of this example:

Starting from the existing Alice/Bob call described in Figure 1 of this document, which established an existing Session-ID header-value...

- o UA-Bob requests Alice to call Carol, using a REFER transaction, as described in [RFC3515]. UA-Alice is initially put on hold, then told in the REFER who to contact with a new INVITE, in this case UA-Carol. This Alice-to-Carol dialog will have a new Call-ID, therefore it requires a new Session-ID header-value. The wrinkle here is we can, and will, use Alice's UUID from her existing dialog with Bob in the new INVITE to Carol.
- o UA-Alice retains her UUID from the Alice-to-Bob call {A} when requesting a call with UA-Carol. This is placed in the "local-uuid" portion of the Session-ID header-value, at the same time inserting a null "remote-uuid" value (because Carol's UA has not yet received the UUID value). This same UUID traverses the B2BUA unchanged.
- o UA-Carol receives the INVITE with a Session ID UUID {A,N}, replaces the A UUID value into the "remote-uuid" portion of the Session-ID header-value and creates its own UUID {C} and places this value in the "local-uuid" portion of the Session-ID header-value - thereby removing the N (null) value altogether. This combination forms a full Session ID {C,A} in the 200 OK to the INVITE. This Session-ID header-value traverses the B2BUA unchanged towards UA-Alice.
- o UA-Alice receives the 200 OK with the Session ID {C,A} and both responds to UA-Carol with an ACK (just as in Figure 1 - switches places of the two UUID fields), and generates a NOTIFY to Bob with a Session ID {A,B} indicating the call transfer was successful.
- o It does not matter which UA terminates the Alice-to-Bob call; Figure 2 shows UA-Bob doing this transaction.

### 9.3. Basic Call Transfer using reINVITE

[[ Editor's Note: This section needs to be discussed further. Standard SIP signaling does not use an INVITE to perform a call transfer. However, it is common for PBX systems to perform a transfer "behind the scenes" wherein a REFER is not consistently utilized. Do we drop the example or do we need further explanatory text? ]]

From the example built within Section 9.1 (the basic Session ID establishment), we proceed to this 'Basic Call Transfer using reINVITE' example.

Alice is talking to Bob. Bob pushes a button on his phone to transfer Alice to Carol via the B2BUA (using reINVITE).

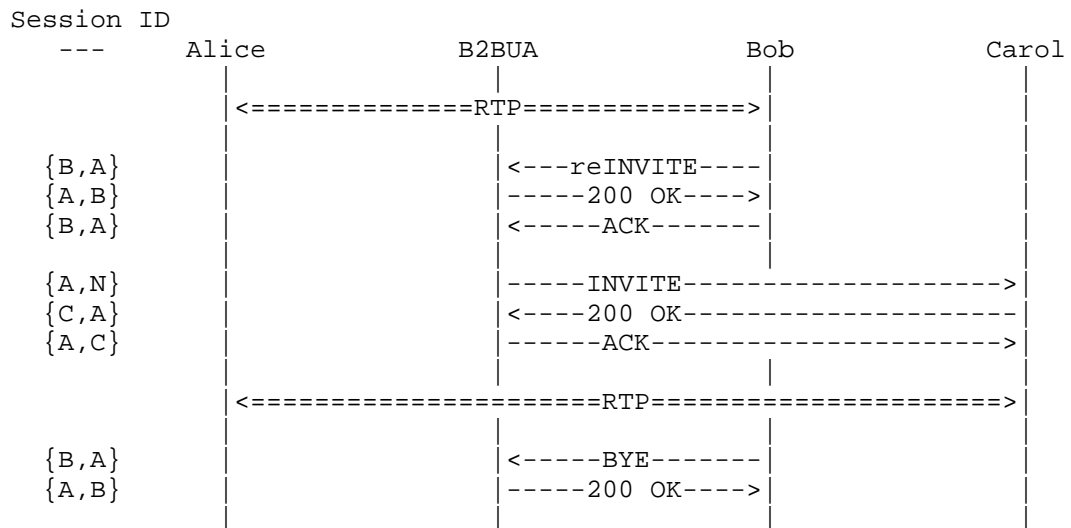


Figure 3 - Call transfer using reINVITE

General operation of this example:

- o We assume the call between Alice and Bob from Section 9.1 is operational with Session ID {A,B}.
- o Bob sends a reINVITE to Alice (with the Session-ID "local-uuid" = Bob's UUID and "remote-uuid" = Alice's UUID), informing her to transfer her existing call to Carol.
- o The B2BUA intercepts this reINVITE and sends a new INVITE with Alice's UUID {"local-uuid" = "A"} to Carol.

- o Carol receives the INVITE and accepts the request and adds her UUID {C} to the Session ID for this session {"local-uuid" = "C", "remote-uuid" = "A"}.
- o Bob terminates the call with a BYE using the Session ID {"local-uuid" = "B", "remote-uuid" = "A"}. The B2BUA intercepts this BYE and responds to Bob since Alice and Carol are now in a new call.

#### 9.4. Single Focus Conferencing

Multiple users call into a conference server (say, an MCU) to attend one of many conferences hosted on or managed by that server. Each user has to identify which conference they want to join, but this information is not necessarily in the SIP messaging. It might be done by having a dedicated address for the conference or via an IVR, as assumed in this example and depicted with the use of M1, M2, and M3. Each user in this example goes through a two-step process of signaling to gain entry onto their conference call, which the conference focus identifies as M'.

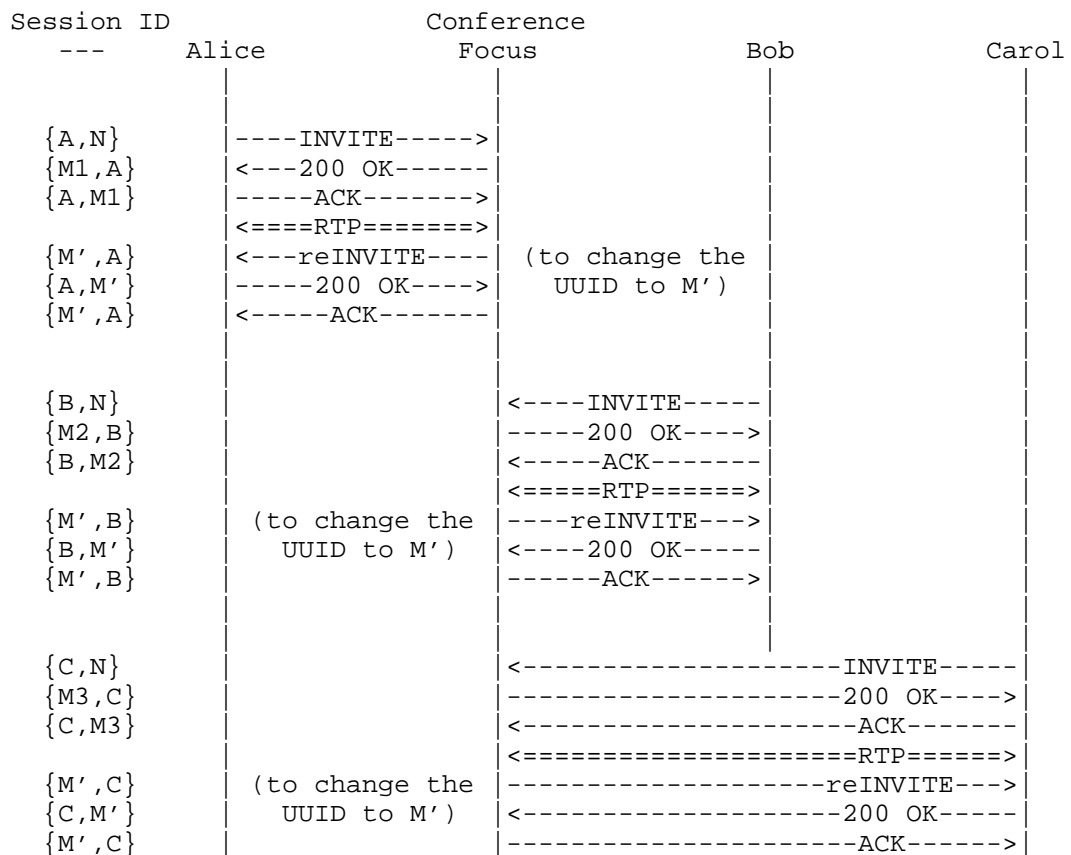


Figure 4 - Single Focus Conference Bridge

General operation of this example:

Alice calls into a conference server to attend a certain conference. This is a two-step operation since Alice cannot include the conference ID at this time and/or any passcode in the INVITE request. The first step is Alice's UA calling another UA to participate in a session. This will appear to be similar as the call-flow in Figure 1 (in section 9.1). What is unique about this call is the second step: the conference server calls back with a reINVITE request with its second UUID, but maintaining the UUID Alice sent in the first INVITE. This subsequent UUID from the conference server will be the same for each UA that calls into this conference server participating in this same conference bridge/call, which is generated once Alice typically authenticates and identifies which bridge she wants to participate on.

- o Alice sends an INVITE to the conference server with her UUID {A} and a "remote" UUID of N.
- o The conference server responds with a 200 OK response which replaces the N UUID with a temporary UUID ("M1") as the "local-uuid" and a "remote-uuid" = "A".

NOTE: this 'temporary' UUID is a real UUID; it is only temporary to the conference server because it knows that it is going to generate another UUID to replace the one just send in the 200 OK.

- o Once Alice, the user, gains access to the IVR for this conference server, she enters a specific conference ID and whatever passcode (if needed) to enter a specific conference call.
- o Once the conference server is satisfied Alice has identified which conference she wants to attend (including any passcode verification), the conference server reINVITES Alice to the specific conference and includes the Session-ID header-value of "local-uuid" = "M'" (and "remote-uuid" = "A") for that conference. All valid participants in the same conference will receive this same UUID for identification purposes and to better enable monitoring, and tracking functions.
- o Bob goes through this two-step process of an INVITE transaction, followed by a reINVITE transaction to get this same UUID ("M'") for that conference.
- o In this example, Carol (and each additional user) goes through the same procedures and steps as Alice and Bob to get on this same conference.



## 9.5. Single Focus Conferencing using WebEx

Alice, Bob and Carol call into same Webex conference.

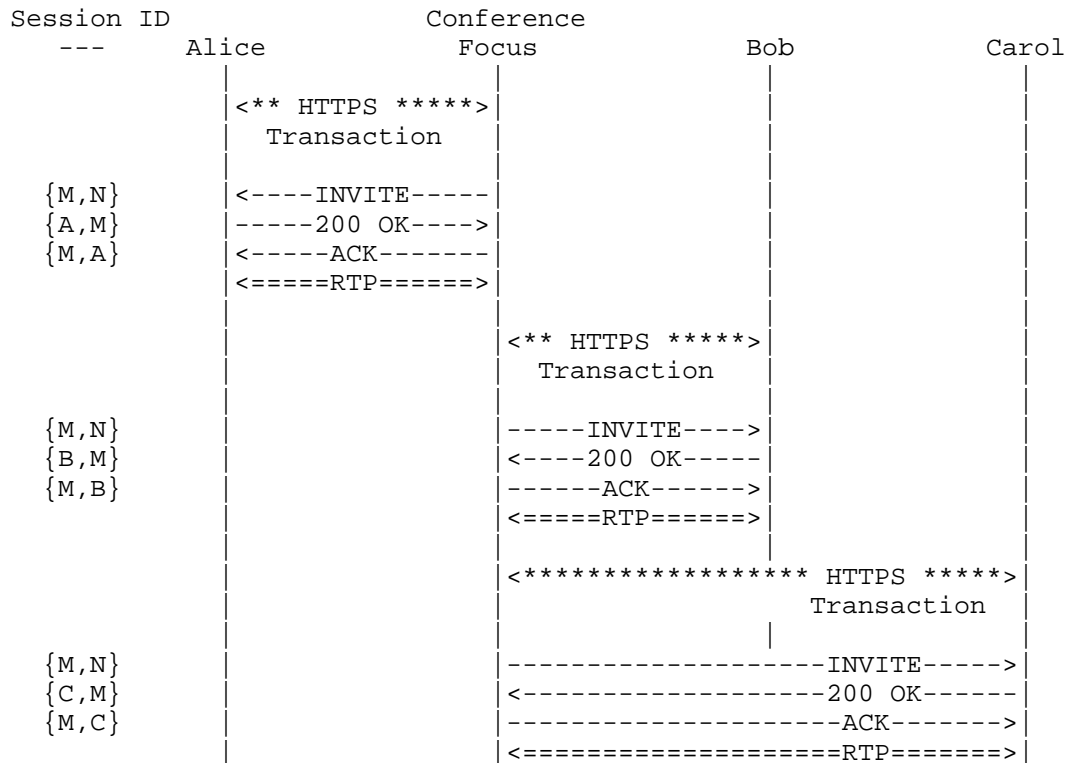


Figure 5 - Single Focus Webex Conference

General operation of this example:

- o Alice communicates with Webex server with desire to join a certain meeting, by meeting number; also includes UA-Alice's contact information (phone number, URI and/or IP address, etc.) for each device she wants for this conference call. For example, the audio and video play-out devices could be separate units.
- o Conference Focus server sends INVITE (Session-ID header-value "local-uuid" = M and a remote UUID of N, where M equals the "local-uuid" for each participant on this conference bridge) to UA-Alice to start session with that server for this A/V conference call.
- o Upon receiving the INVITE request from the conference focus server, Alice responds with a 200 OK. Her UA moves the "local-uuid" unchanged into the "remote-uuid" field, and generates her

own UUID and places that into the "local-uuid" field to complete the Session-ID construction.

- o Bob and Carol perform same function to join this same A/V conference call as Alice.

#### 9.6. Cascading Conference Bridge Support for the Session ID

To expand conferencing capabilities requires cascading conference bridges. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. RFC 4579 [RFC4579] defines the 'isfocus' Contact: header parameter just for this purpose.

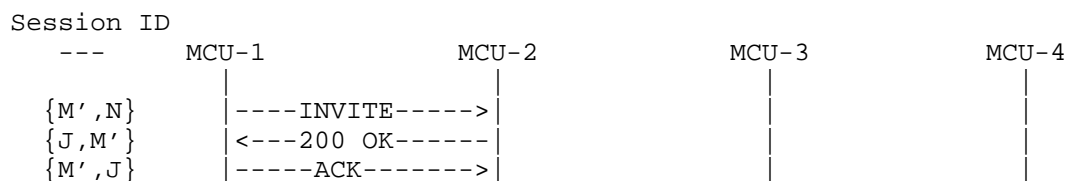
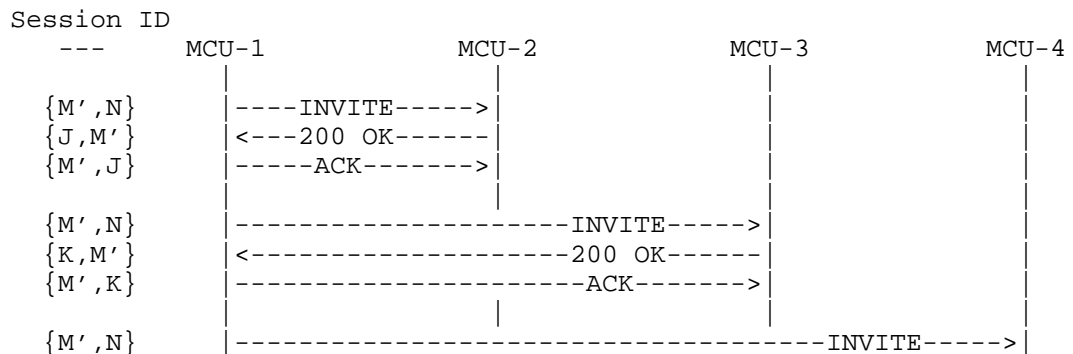


Figure 6 - MCUs Communicating Session ID UUID for Bridge

Regardless of which MCU (1 or 2) a UA contacts for this conference, once the above exchange has been received and acknowledged, the UA will get the same {M',N} UUID pair from the MCU for the complete Session ID.

A more complex form would be a series of MCUs all being informed of the same UUID to use for a specific conference. This series of MCUs can either be informed

- o All by one MCU (that initially generates the UUID for the conference),
- o The one MCU that generates the UUID informs one or several MCUs of this common UUID, and they inform downstream MCUs of this common UUID each will be using for this one conference, or



```

{L,M'}      | <-----200 OK-----|
{M',L}      | -----ACK----->|

```

Figure 7 - MCU Communicating Session ID UUID to More than One

General operation of this example:

- o The MCU generating the Session ID UUID communicates this in a separate INVITE, having a Contact header with the 'isfocus' header parameter. This will identify the MCU as what RFC 4579 conference-aware SIP entity.
- o An MCU that receives this {M',N} UUID pair in an inter-MCU transaction, can communicate the M' UUID in a manner in which it was received (though this time this second MCU would be the UAC MCU), unless local policy dictates otherwise.

#### 9.6.1. Calling into Cascaded Conference Bridge for the Session ID

Here is an example of how a UA, say Robert, calls into a cascaded conference focus. Because MCU-1 has already contacted MCU-3, the MCU where Robert is going to join the conference, MCU-3 already has the Session-ID (M') for this particular conference call.

Session ID	MCU-1	MCU-2	MCU-3	Robert
{M',N}	----	----	----	----
{J,M'}	-----INVITE----->	----	----	----
{M',J}	<---200 OK-----	----	----	----
	-----ACK----->	----	----	----
{M',N}	-----INVITE----->	----	----	----
{K,M'}	<-----200 OK-----	----	----	----
{M',K}	-----ACK----->	----	----	----
{R,N}	----	----	----	----
{M',R}	----	----	<---INVITE-----	----
{R,M'}	----	----	-----200 OK----->	----
	----	----	<---ACK-----	----

Figure 8 - A UA Calling into a Cascaded MCU UUID

General operation of this example:

- o The UA, Robert in this case, INVITEs the MCU to join a particular conference call. Robert's UA does not know anything about whether this is the main MCU of the conference call, or a cascaded MCU. Robert likely does not know MCUs can be cascaded, he just wants to join a particular call. Like as with any standard implementation, he includes a null "remote-uuid".

- o The cascaded MCU, upon receiving this INVITE from Robert, replaces the null UUID with the UUID value communicated from MCU-1 for this conference call as the "local-uuid" in the SIP response. Thus, moving Robert's UUID "R" to the "remote-uuid" value.
- o The ACK has the Session-ID {R,M'}, completing the 3-way handshake for this call establishment. Robert has now joined the conference call originated from MCU-1.

#### 9.7. Basic 3PCC for two UAs

External entity sets up call to both Alice and Bob for them to talk to each other.

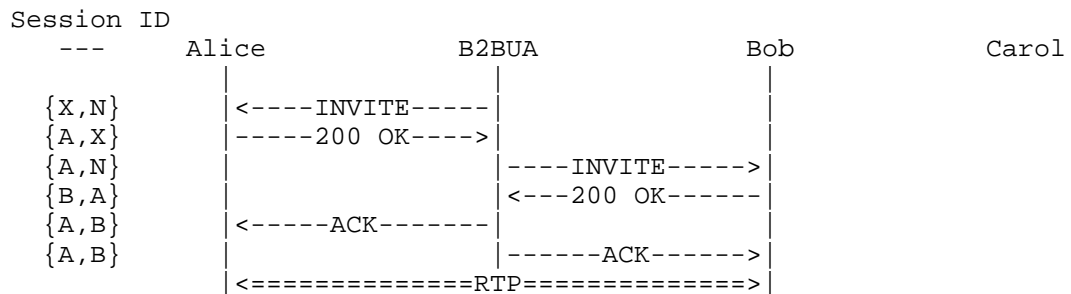


Figure 8 - 3PCC initiated call between Alice and Bob

General operation of this example:

- o Some out of band procedure directs a B2BUA (or other SIP server) to have Alice and Bob talk to each other. In this case, the SIP server MUST be transaction stateful, if not dialog stateful.
- o The SIP server INVITEs Alice to a session and uses a temporary UUID {X} and a null UUID pairing.
- o Alice receives and accepts this call set-up and replaces the null UUID with her UUID {A} in the Session ID, now {A,X}.
- o The transaction stateful SIP server receives Alice's UUID {A} in the local UUID portion and keeps it there, and discards its own UUID {X}, replacing this with a null UUID value in the INVITE to Bob as if this came from Alice originally.
- o Bob receives and accepts this INVITE and adds his own UUID {B} to the Session ID, now {B,A} for the response.
- o And the session is established.

## 9.8. Session ID Handling in 100 Trying SIP Response and CANCEL Request

The following two subsections show examples of the Session ID for a 100 Trying response and a CANCEL request in a single call-flow.

### 9.8.1. Session ID Handling in a 100 Trying SIP Response

The following 100 Trying response is taken from an existing RFC, from [RFC5359] Section 2.9 ("Call Forwarding - No Answer").

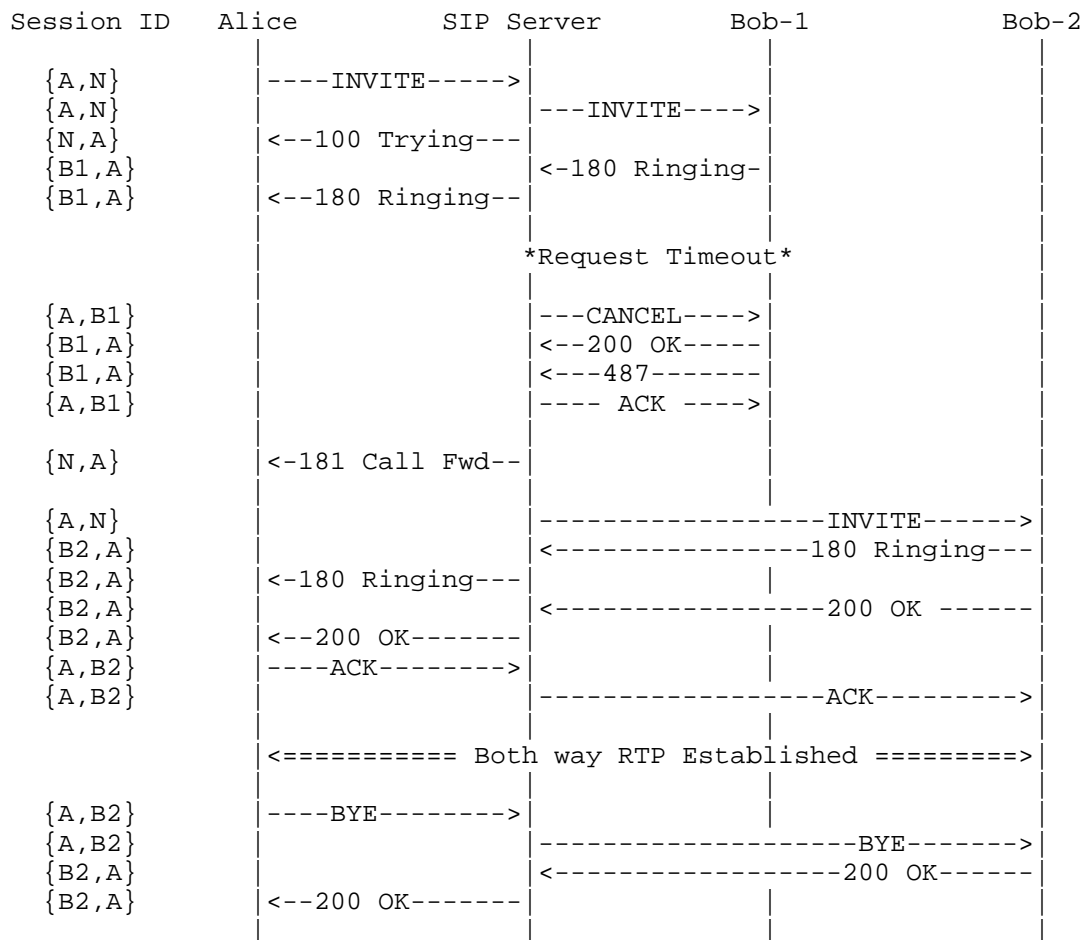


Figure 9 - Session ID in the 100 Trying and CANCEL Messaging

Below is the explanatory text from RFC 5359 Section 2.9 detailing what the desired behavior is in the above call flow (i.e., what the call-flow is attempting to achieve).

"Bob wants calls to B1 forwarded to B2 if B1 is not answered (information is known to the SIP server). Alice calls B1 and no one answers. The SIP server then places the call to B2."

General operation of this example:

- o Alice generates an INVITE request because she wants to invite Bob to join her session. She creates a UUID as described in section 9.1, and places that value in the "local-uuid" field of the Session-ID header-value. Alice also generates a "remote-uuid" of null and sends this along with the "local-uuid".
- o The SIP server (imagine this is a B2BUA), upon receiving Alice's INVITE, and generates the optional provisional response 100 Trying. Since the SIP server has no knowledge Bob's UUID for his part of the Session ID value, it cannot include his "local-uuid". Rather, any 100 Trying response includes Alice's UUID in the "remote-uuid" portion of the Session-ID header-value with a null "local-uuid" value in the response. This is consistent with what Alice's UA expects to receive in any SIP response containing this UUID.

#### 9.8.2. Session ID in a CANCEL SIP Request

In the same call-flow example as the 100 Trying response is a CANCEL request. Please refer to Figure 9 for the CANCEL request example.

General operation of this example:

- o In Figure 9 above, Alice generates an INVITE with her UUID value in the Session-ID header-value.
- o Bob-1 responds to this INVITE with a 180 Ringing. In that response, he includes his UUID in the Session-ID header-value (i.e., {B1,A}); thus completing the Session-ID header-value for this session, even though no final response has been generated by any of Bob's UAs.
- o This means that if the SIP server were to generate a SIP request within this session, in this case a CANCEL request, it would have a complete Session ID to include in that request. In this case, the "local-uuid" = "A", and the "remote-uuid" = "B1".
- o As it happens with this CANCEL, the SIP server intends to invite another UA of Bob (i.e., B2) for Alice to communicate with.
- o In this example call-flow, taken from RFC 5359, Section 2.9, a 181 (Call is being Forwarded) response is sent to Alice. Since the SIP server generated this SIP request, and has no knowledge of Bob-2's UUID value, it cannot include that value in this 181. Thus, and for the exact reasons the 100 Trying including the

Session ID value, only Alice's UUID is included in the remote-uuid field of the Session-ID header-value, with a null UUID present in the "local-uuid" field.

#### 9.9. Session ID in an out-of-dialog REFER Transaction

[[ Editor's Note: In this section, we use {X,Y} as the Session-ID related to the OOD REFER exchange. This ensures that the exchange is treated as a distinct session. Do we want that or do we want to consider such exchanges to be part of the same session and re-use {A,B}? ]]

The following call-flow was extracted from Section 6.1 of [RFC5589] ("Successful Transfer"), with the only changes being the names of the UAs to maintain consistency within this document.

Alice is the transferee  
 Bob is the transferer  
 and Carol is the transfer-target

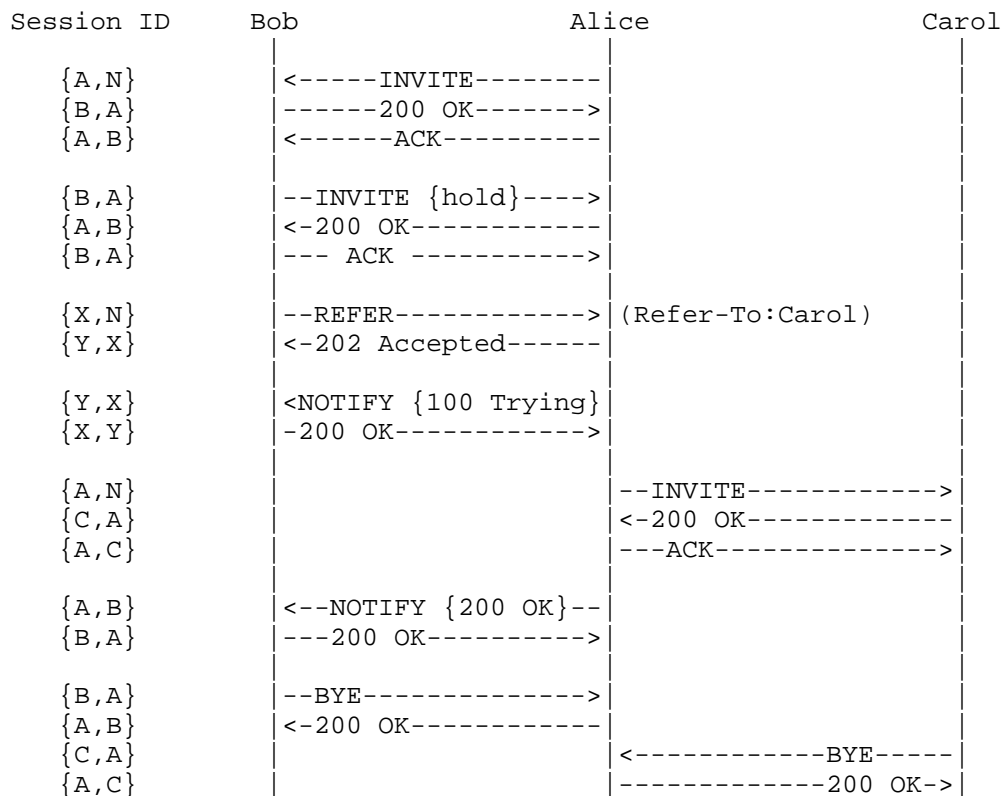


Figure 10: Basic Transfer Call Flow

General operation of this example:

- o Just as in Section 9.2, Figure 2, Alice invites Bob to a session, and Bob eventually transfers Alice to communicate with Carol.
- o What is different about the call-flow in Figure 10 is that Bob's REFER is not in-dialog, meaning it would have the same UUID pair. Rather, in this case, Bob's using an out-of-dialog REFER, meaning Bob generates a new UUID for this SIP request, and Alice, subsequently would also generate a new UUID for the 202 (Accepted) response, replacing the null "remote-uuid" in the REFER.
- o Alice will use her existing UUID {A,N} in the INVITE towards Carol (who generates UUID "C" for this session), thus maintaining the common UUID within the Session ID for this new Alice-to-Carol session.

#### 10. Compatibility with a Previous Implementation

There is a much earlier and proprietary document that specifies the use of a Session-ID header [I-D.kaplan-insipid-session-id] that we will herewith attempt to achieve backwards compatibility. Neither Session-ID header has any versioning information, so merely adding that this document describes "version 2" is insufficient. Here are the set of rules for compatibility between the two specifications. For the purposes of this discussion, we will label the proprietary specification of the Session-ID as the "old" version and this specification as the "new" version of the Session-ID.

The previous (i.e., "old") version only has a single value as a Session-ID, but has a generic-parameter value that can be of use.

In order to have an "old" version talk to an "old" version implementation, nothing needs to be done as far as the IETF is concerned.

In order to have a "new" version talk to a "new" version implementation, both implementations need to follow this document (to the letter) and everything should be just fine.

But that is where compatibility is not ensured, given the unknowns related to the behavior of entities implementing the pre-standard implementation. For this "new" implementation to work with the "old" implementation \*and\* any "old" implementation to work with "new" implementations, there needs to be a set of rules for all "new" implementations MUST follow.

- since no option tags or feature tags are to be used for distinguishing versions, the presence and order of any "remote-uuid" value within the Session-ID header value is to be used to distinguish implementation versions.



- if a SIP request has a "remote-uuid" value, this comes from a standard implementation, and not a pre-standard one.
- if a SIP request has no "remote-uuid" value, this comes from a pre-standard implementation, and not a standard one. In this case, one UUID is used to identify this dialog, even if the responder is a standard implementation of this specification.
- if a SIP response has a non-null "local-uuid" that is 32 octets long, this response comes from a standard implementation. There are two exceptions to this rule: a 100 Trying response and a 181 Call Forwarded response will have null "local-uuid" values.
- if a SIP response has a non-null "local-uuid" that is not 32 octets long, this response comes from a misbehaving implementation, and its Session-ID header value MUST be discarded. That said, the response might still be valid according to the rules within SIP [RFC3261], and SHOULD be checked further.
- if a SIP response arrives that has the same value of Session-ID UUIDs in the same order as was sent, this comes from a pre-standard implementation, and MUST NOT be discarded for not altering the null "remote-uuid". In this case, any new transaction within this dialog MUST preserve the order of the two UUIDs within all Session-ID header-values, including the ACK, until this dialog is terminated.
- if a SIP response only contains the "local-uuid" that was sent originally, this comes from a pre-standard implementation and MUST NOT be discarded for removing the null "remote-uuid". In this case, all future transactions within this dialog MUST contain only the UUID received in the first SIP response. Any new transaction starting a new dialog from the standard Session-ID implementation MUST include and "local-uuid" and a null "remote-uuid", even if that new dialog is between the same two UAs.
- Standard implementations SHOULD NOT expect pre-standard implementations to be consistent in their implementation, even within the same dialog. For example, perhaps the first, third and tenth responses contain a "remote-uuid", but all the others do not. This behavior MUST be allowed by implementations of this specification.
- All of this does not apply to other parameters that might be defined in the future, i.e., currently unknown. They are discarded.

## 11. Security Considerations

When creating a UUID value, endpoints SHOULD ensure that there is no user or device-identifying information contained within the UUID. In some environments, though, use of a MAC address, which is one option when constructing a UUID, may be desirable, especially in some

enterprise environments. When communicating over the Internet, though, the UUID value MUST utilize random values.

The Session Identifier might be utilized for logging or troubleshooting, but MUST NOT be used for billing purposes.

[[ Editor's Note: We need to consider privacy-related concerns. Can we enumerate the security and privacy issues that might arise through the use of the Session-ID? ]]

## 12. IANA Considerations

### 12.1. Registration of the "Session-ID" Header Field

The following is the registration for the 'Session-ID' header field to the "Header Name" registry at <http://www.iana.org/assignments/sip-parameters>:

RFC number: RFC XXXX

Header name: 'Session-ID'

Compact form: none

[RFC Editor: Please replace XXXX in this section and the next with the this RFC number of this document.]

### 12.2. Registration of the "remote" Parameter

The following parameter is to be added to the "Header Field Parameters and Parameter Values" section of the SIP parameter registry:

Header Field	Parameter Name	Predefined Values	Reference
Session-ID	remote	No	[RFCXXXX]

## 13. Acknowledgments

The authors would like to thank Robert Sparks, Hadriel Kaplan, Christer Holmberg, Paul Kyzivat, Brett Tate, Keith Drage, Mary Barnes, and Charles Eckel for their invaluable comments during the development of this document.

## 14. References

### 14.1. Normative References

- [RFC3261] Rosenberg, J., et al., "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [H.323] Recommendation ITU-T H.323, "Packet-based multimedia communications systems", December 2009.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4122] Leach, P., Mealling, M., Salz, R., "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.
- [RFC5234] Crocker, D., Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008.
- [RFC4579] Johnston, A., Levin, O., "Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents", RFC 4579, August 2006.
- [RFC3891] Mahy, R., Biggs, B., Dean, R., 'The Session Initiation Protocol (SIP) "Replaces" Header', RFC 3891, September 2004.
- [RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.
- [I-D.kaplan-insipid-session-id]  
Kaplan, H., "A Session Identifier for the Session Initiation Protocol (SIP)", August 2013.

### 14.2. Informative References

- [RFC3550] Schulzrinne, H., et al., "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.
- [I-D.ietf-insipid-session-id-reqts]  
Jones, et al., "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks", draft-ietf-insipid-session-id-reqts-11, February 2014.
- [RFC5359] Johnston, A., et al., "Session Initiation Protocol Service Examples", RFC 5359, October 2008.
- [RFC5589] Sparks, R., Johnston, A., Petrie, D., "Session Initiation Protocol (SIP) Call Control - Transfer", RFC 5359, June 2009.

## Author's Addresses

Paul E. Jones (Ed.)  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA

Phone: +1 919 476 2048  
Email: paulej@packetizer.com  
IM: xmpp:paulej@packetizer.com

Chris Pearce  
Cisco Systems, Inc.  
2300 East President George Bush Highway  
Richardson, TX 75082  
USA

Phone: +1 972 813 5123  
Email: chrep@cisco.com  
IM: xmpp:chrep@cisco.com

James Polk (Ed.)  
Cisco Systems, Inc.  
3913 Treemont Circle  
Colleyville, Texas  
USA

Phone: +1 817 271 3552  
Email: jmpolk@cisco.com  
IM: xmpp:jmpolk@cisco.com

Gonzalo Salgueiro  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA

Phone: +1 919 392 3266  
Email: gsalguei@cisco.com  
IM: xmpp:gsalguei@cisco.com



Network Working Group  
Internet-Draft  
Obsoletes: 7329 (if approved)  
Intended status: Standards Track  
Expires: February 19, 2017

P. Jones  
G. Salgueiro  
C. Pearce  
P. Giralt  
Cisco Systems, Inc.  
August 18, 2016

End-to-End Session Identification in IP-Based Multimedia Communication  
Networks  
draft-ietf-insipid-session-id-27

Abstract

This document describes an end-to-end Session Identifier for use in IP-based multimedia communication systems that enables endpoints, intermediary devices, and management systems to identify a session end-to-end, associate multiple endpoints with a given multipoint conference, track communication sessions when they are redirected, and associate one or more media flows with a given communication session. While the identifier is intended to work across multiple protocols, this document describes its usage in SIP.

This document also describes a backwards compatibility mechanism for an existing session identifier implementation (RFC 7329) that is sufficiently different from the procedures defined in this document.

This document obsoletes RFC 7329.

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 February 19, 2017.

## Copyright Notice

Copyright (c) 2016 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. Conventions used in this document . . . . .	4
3. Session Identifier Definitions, Requirements, and Use Cases .	4
4. Constructing and Conveying the Session Identifier . . . . .	4
4.1. Constructing the Session Identifier . . . . .	4
4.2. Conveying the Session Identifier . . . . .	5
5. The Session-ID Header Field . . . . .	7
6. Endpoint Behavior . . . . .	8
7. Processing by Intermediaries . . . . .	10
8. Handling of Remote UUID Changes . . . . .	13
9. Associating Endpoints in a Multipoint Conference . . . . .	15
10. Examples of Various Call Flow Operations . . . . .	16
10.1. Basic Call with 2 UUIDs . . . . .	16
10.2. Basic Call Transfer using REFER . . . . .	20
10.3. Basic Call Transfer using re-INVITE . . . . .	22
10.4. Single Focus Conferencing . . . . .	24
10.5. Single Focus Conferencing using a web-based conference service . . . . .	26
10.6. Cascading Conference Bridges . . . . .	28
10.6.1. Establishing a Cascaded Conference . . . . .	28
10.6.2. Calling into Cascaded Conference Bridges . . . . .	29
10.7. Basic 3PCC for two UAs . . . . .	30
10.8. Handling in 100 Trying SIP Response and CANCEL Request .	31
10.8.1. Handling in a 100 Trying SIP Response . . . . .	31
10.8.2. Handling a CANCEL SIP Request . . . . .	33
10.9. Out-of-dialog REFER Transaction . . . . .	34
11. Compatibility with a Previous Implementation . . . . .	35
12. Security and Privacy Considerations . . . . .	36
13. IANA Considerations . . . . .	38
13.1. Registration of the "Session-ID" Header Field . . . . .	38
13.2. Registration of the "remote" Parameter . . . . .	38

14. Acknowledgements . . . . .	38
15. Dedication . . . . .	38
16. References . . . . .	39
16.1. Normative References . . . . .	39
16.2. Informative References . . . . .	40
Authors' Addresses . . . . .	41

## 1. Introduction

IP-based multimedia communication systems such as Session Initiation Protocol (SIP) [RFC3261] and [H.323] have the concept of a "call identifier" that is globally unique. The identifier is intended to represent an end-to-end communication session from the originating device to the terminating device. Such an identifier is useful for troubleshooting, session tracking, and so forth.

For several reasons, however, the current call identifiers defined in SIP and H.323 are not suitable for end-to-end session identification. A fundamental issue in protocol interworking is the fact that the syntax for the call identifier in SIP and H.323 is different. Thus, if both protocols are used in a call, it is impossible to exchange the call identifier end-to-end.

Another reason why the current call identifiers are not suitable to identify a session end-to-end is that, in real-world deployments, devices such as session border controllers [RFC7092] often change the session signaling, including the value of the call identifier, as it passes through the device. While this is deliberate and useful, it makes it very difficult to track a session end-to-end.

This document defines a new identifier, referred to as the Session Identifier, that is intended to overcome the issues that exist with the currently defined call identifiers used in SIP and other IP-based communication systems. The identifier defined here has been adopted by the ITU ([H.460.27]) for use in H.323-based systems, allowing for the ability to trace a session end-to-end for sessions traversing both SIP and H.323-based systems. This document defines its use in SIP.

The procedures specified in this document attempt to comply with the requirements specified in [RFC7206]. The procedures also specify capabilities not mentioned in [RFC7206], shown in call flows in section 10. Additionally, the specification attempts to account for a previous, pre-standard version of a SIP Session Identifier header [RFC7329], specifying a backwards compatibility approach in section 11.



## 2. Conventions used in this document

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] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

The term "Session Identifier" refers to the value of the identifier, whereas "Session-ID" refers to the header field used to convey the identifier. The Session Identifier is a set of two Universally Unique Identifiers (UUIDs) and each element of that set is simply referred to herein as a UUID.

Throughout this document, the term "endpoint" refers to a SIP User Agent (UA) that either initiates or terminates a SIP session, such as a user's mobile phone or a conference server, but excludes entities such as Back-to-Back User Agents (B2BUAs) that are generally located along the call signaling path between endpoints. The term "intermediary" refers to any entity along the call signaling path between the aforementioned endpoints, including B2BUAs and SIP proxies. In certain scenarios, intermediaries are allowed to originate and terminate SIP messages without an endpoint being part of the session or transaction. An intermediary may be performing interworking between different protocols (e.g. SIP and H.323) that support the Session Identifier defined in this document.

## 3. Session Identifier Definitions, Requirements, and Use Cases

Requirements and use cases for the end-to-end Session Identifier, along with the definition of "session identifier", "communication session", and "end-to-end" can be found in [RFC7206]. Throughout this document, the term "session" refers to a "communication session" as defined in [RFC7206].

As mentioned in section 6.1 of [RFC7206], the ITU-T undertook a parallel effort to define compatible procedures for an H.323 Session Identifier. They are documented in [H.460.27].

## 4. Constructing and Conveying the Session Identifier

### 4.1. Constructing the Session Identifier

The Session Identifier comprises two UUIDs [RFC4122], with each UUID representing one of the endpoints participating in the session.

The version number in the UUID indicates the manner in which the UUID is generated, such as using random values or using the MAC address of

the endpoint. To satisfy the requirement that no user or device information be conveyed, endpoints MUST generate version 4 (random) or version 5 (SHA-1) UUIDs to address privacy concerns related to use of MAC addresses in UUIDs.

When generating a version 5 UUID, endpoints or intermediaries MUST utilize the procedures defined in Section 4.3 of [RFC4122] and employ the following "name space ID":

```
uuid_t NameSpace_SessionID = {  
    /* a58587da-c93d-11e2-ae90-f4ea67801e29 */  
    0xa58587da,  
    0xc93d,  
    0x11e2,  
    0xae, 0x90, 0xf4, 0xea, 0x67, 0x80, 0x1e, 0x29  
};
```

Further, the "name" to utilize for version 5 UUIDs is the concatenation of the Call-ID header-value and the "tag" parameter that appears on the "From" or "To" line associated with the device for which the UUID is created. Once an endpoint generates a UUID for a session, the UUID never changes, even if values originally used as input into its construction change over time.

Stateless intermediaries that insert a Session-ID header field into a SIP message on behalf of an endpoint MUST utilize version 5 UUIDs to ensure that UUIDs for the communication session are consistently generated. If a stateless intermediary does not know the tag value for the endpoint (e.g., a new INVITE without a To: tag value or an older SIP implementation [RFC2543] that did not include a tag parameter), the intermediary MUST NOT attempt to generate a UUID for that endpoint. Note that if an intermediary is stateless and the endpoint on one end of the call is replaced with another endpoint due to some service interaction, the values used to create the UUID should change and, if so, the intermediary will compute a different UUID.

#### 4.2. Conveying the Session Identifier

The SIP User Agent (UA) initiating a new session by transmitting a SIP request ("Alice"), i.e., a User Agent Client (UAC), MUST create a new, previously unused, UUID and transmit that to the ultimate destination UA ("Bob"). Likewise, the destination UA ("Bob"), i.e., a User Agent Server (UAS), MUST create a new, previously unused, UUID and transmit that to the first UA ("Alice"). These two distinct UUIDs form what is referred to as the Session Identifier and is represented in this document in set notation of the form {A,B}, where "A" is UUID value created by UA "Alice" and "B" is the UUID value

created by UA "Bob". The Session Identifier {A,B} is equal to the Session Identifier {B,A}. Section 6 describes how the UUIDs selected by the source and destination UAs persist for the duration of the session.

In the case where only one UUID is known, such as when a UA first initiates a potentially dialog-initiating SIP request, the Session Identifier would be {A,N}, where "A" represents the UUID value transmitted by the UA "Alice" and "N" is what is referred to as the nil UUID [RFC4122] (see section 5).

Since SIP sessions are subject to any number of service interactions, SIP INVITE messages might be forked as sessions are established, and since conferences might be established or expanded with endpoints calling in or the conference focus calling out, the construction of the Session Identifier as a set of UUIDs is important.

To understand this better, consider that an endpoint participating in a communication session might be replaced with another, such as the case where two "legs" of a call are joined together by a Private Branch Exchange (PBX). Suppose "Alice" and "Bob" both call UA C ("Carol"). There would be two distinctly identifiable Session Identifiers, namely {A,C} and {B,C}. Then suppose that "Carol" uses a local PBX function to join the call between herself and "Alice" with the call between herself and "Bob", resulting in a single remaining call between "Alice" and "Bob". This merged call can be identified using two UUID values assigned by each entity in the communication session, namely {A,B} in this example.

In the case of forking, "Alice" might send an INVITE that gets forked to several different endpoints. A means of identifying each of these separate communication sessions is needed and, since each of the destination UAs will create its own UUID, each communication session would be uniquely identified by the values {A, B1}, {A, B2}, {A, B3}, and so on, where each of the Bn values refers to the UUID created by the different UAs to which the SIP session is forked.

For conferencing scenarios, it is also useful to have a two-part Session Identifier where the conference focus specifies the same UUID for each conference participant. This allows for correlation among the participants in a single conference. For example, in a conference with three participants, the Session Identifiers might be {A,M}, {B,M}, and {C,M}, where "M" is assigned by the conference focus. Only a conference focus will purposely utilize the same UUID for more than one SIP session and, even then, such reuse MUST be restricted to the participants in the same conference.

How a device acting on Session Identifiers processes or utilizes the Session Identifier is outside the scope of this document, however devices storing a Session Identifier in a log file SHOULD follow the security considerations outlined in [RFC6872]. Note that the primary intent of a Session Identifier is for troubleshooting and should therefore be included in logs at rest that will be used for troubleshooting purposes.

## 5. The Session-ID Header Field

This document replaces the definition of the "Session-ID" token that was added to the definition of the element "message-header" in the SIP message grammar by [RFC7329]. The Session-ID header is a single-instance header.

Each endpoint participating in a communication session has a distinct, preferably locally-generated, UUID associated with it. The endpoint's UUID value remains unchanged throughout the duration of the communication session. Multipoint conferences can bridge sessions from multiple endpoints and impose unique requirements defined in Section 9. An intermediary MAY generate a UUID on behalf of an endpoint that did not include a UUID of its own.

The UUID values for each endpoint are inserted into the "Session-ID" header field of all transmitted SIP messages. The Session-ID header field has the following ABNF [RFC5234] syntax:

```
session-id           = "Session-ID" HCOLON session-id-value
session-id-value     = local-uuid *(SEMI sess-id-param)
local-uuid           = sess-uuid / nil
remote-uuid          = sess-uuid / nil
sess-uuid            = 32(DIGIT / %x61-66) ;32 chars of [0-9a-f]
sess-id-param        = remote-param / generic-param
remote-param         = "remote" EQUAL remote-uuid
nil                  = 32("0")
```

The productions "SEMI", "EQUAL", and "generic-param" are defined in [RFC3261]. The production DIGIT is defined in [RFC5234].

The Session-ID header field MUST NOT have more than one "remote" parameter. In the case where an entity compliant with this

specification is interworking with an entity that implemented [RFC7329], the "remote" parameter may be absent, but otherwise the remote parameter MUST be present. The details under which those conditions apply are described in Section 11. Except for backwards compatibility with [RFC7329], the "remote" parameter MUST be present.

A special nil UUID value composed of 32 zeros is required in certain situations. A nil UUID is expected as the "remote-uuid" of every initial standard SIP request since the initiating endpoint would not initially know the UUID value of the remote endpoint. This nil value will get replaced by the ultimate destination UAS when that UAS generates a response message. One caveat is explained in Section 11 for a possible backwards compatibility case. A nil UUID value is also returned by some intermediary devices that send provisional or other responses as the "local-uuid" component of the Session-ID header field value, as described in Section 7.

The "local-uuid" in the Session-ID header field represents the UUID value of the endpoint transmitting a message and the "remote-uuid" in the Session-ID header field represents the UUID of the endpoint's peer. For example, a Session-ID header field might appear like this:

```
Session-ID: ab30317f1a784dc48ff824d0d3715d86;  
           remote=47755a9de7794ba387653f2099600ef2
```

While this is the general form of the Session-ID header field, exceptions to syntax and procedures are detailed in subsequent sections.

The UUID values are presented as strings of lowercase hexadecimal characters, with the most significant octet of the UUID appearing first.

## 6. Endpoint Behavior

To comply with this specification, endpoints (non-intermediaries) MUST include a Session-ID header field value in all SIP messages transmitted as a part of a communication session. The locally-generated UUID of the transmitter of the message MUST appear in the "local-uuid" portion of the Session-ID header field value. The UUID of the peer device, if known, MUST appear as the "remote" parameter following the transmitter's UUID. The nil UUID value MUST be used if the peer device's UUID is not known.

Once an endpoint allocates a UUID value for a communication session, the endpoint originating the request MUST NOT change that UUID value for the duration of the session, including when

- o communication attempts are retried due to receipt of 4xx messages or request timeouts;
- o the session is redirected in response to a 3xx message;
- o a session is transferred via a REFER message [RFC3515]; or
- o a SIP dialog is replaced via an INVITE with Replaces [RFC3891].

An endpoint that receives a Session-ID header field MUST take note of any non-nil "local-uuid" value that it receives and assume that is the UUID of the peer endpoint within that communications session. Endpoints MUST include this received UUID value as the "remote" parameter when transmitting subsequent messages, making sure not to change this UUID value in the process of moving the value internally from the "local-uuid" field to the "remote-uuid" field.

If an endpoint receives a 3xx message, receives a REFER that directs the endpoint to a different peer, or receives an INVITE with Replaces that also potentially results in communicating with a new peer, the endpoint MUST complete any message exchanges with its current peer using the existing Session Identifier, but MUST NOT use the current peer's UUID value when sending the first message to what it believes may be a new peer endpoint (even if the exchange results in communicating with the same physical or logical entity). The endpoint MUST retain its own UUID value, however, as described above.

It should be noted that messages received by an endpoint might contain a "local-uuid" value that does not match what the endpoint expected its peer's UUID to be. It is also possible for an endpoint to receive a "remote-uuid" value that does not match its generated UUID for the session. Either might happen as a result of service interactions by intermediaries and MUST NOT affect how the endpoint processes the session; however, the endpoint may log this event for troubleshooting purposes.

An endpoint MUST assume that the UUID value of the peer endpoint may change at any time due to service interactions. Section 8 discusses how endpoints must handle remote UUID changes.

It is also important to note that if an intermediary in the network forks a session, the endpoint initiating a session may receive multiple responses back from different endpoints, each of which contains a different UUID ("local-uuid") value. Endpoints MUST ensure that the correct UUID value is returned in the "remote" parameter when interacting with each endpoint. The one exception is when the endpoint sends a CANCEL message, in which case the Session-

ID header field value MUST be identical to the Session-ID header field value sent in the original request.

If an endpoint receives a message that does not contain a Session-ID header field, that message must have no effect on what the endpoint believes is the UUID value of the remote endpoint. That is, the endpoint MUST NOT change the internally maintained "remote-uuid" value for the peer.

If an endpoint receives a SIP response with a non-nil "local-uuid" that is not 32 octets long, this response comes from a misbehaving implementation, and its Session-ID header field MUST be discarded. That said, the response might still be valid according to the rules within SIP [RFC3261], and SHOULD be checked further.

A Multipoint Control Unit (MCU) is a special type of conferencing endpoint and is discussed in Section 9.

## 7. Processing by Intermediaries

The following applies only to an intermediary that wishes to comply with this specification and does not impose a conformance requirement on intermediaries that elect to not provide any special treatment for the Session-ID header field. Intermediaries that do not comply with this specification might pass the header unchanged or drop it entirely.

The Call-ID often reveals personal, device, domain or other sensitive information associated with a user, which is one reason why intermediaries, such as session border controllers, sometimes alter the Call-ID. In order to ensure the integrity of the end-to-end Session Identifier, it is constructed in a way which does not reveal such information, removing the need for intermediaries to alter it.

When an intermediary receives messages from one endpoint in a communication session that causes the transmission of one or more messages toward the second endpoint in a communication session, the intermediary MUST include the Session-ID header field in the transmitted messages with the same UUID values found in the received message, except as outlined in this section and in section 8.

If the intermediary aggregates several responses from different endpoints, as described in Section 16.7 of [RFC3261], the intermediary MUST set the local-uuid field to the nil UUID value when forwarding the aggregated response to the endpoint since the true UUID value of the peer is undetermined at that point. Note that an intermediary that does not implement this specification might forward a non-nil value, resulting in the originating endpoint receiving

different UUID values in the responses. It is possible for this to result in the endpoint temporarily using the wrong remote UUID. Subsequent messages in the dialog should resolve the temporary mismatch as long as the endpoint follows the rules outlined in Section 8 dealing with the handling of remote UUID changes.

Intermediary devices that transfer a call, such as by joining together two different "call legs", MUST properly construct a Session-ID header field that contains the UUID values associated with the endpoints involved in the joined session and correct placement of those values. As described in Section 6, the endpoint receiving a message transmitted by the intermediary will assume that the first UUID value belongs to its peer endpoint.

If an intermediary receives a SIP message without a Session-ID header field or valid header field value from an endpoint for which the intermediary is not storing a "remote-uuid" value, the intermediary MAY assign a "local-uuid" value to represent that endpoint and, having done so, MUST insert that assigned value into all signaling messages on behalf of the endpoint for that dialog. In effect, the intermediary becomes dialog stateful and it MUST follow the endpoint procedures in Section 6 with respect to Session-ID header field value treatment with itself acting as the endpoint (for the purposes of the Session-ID header field) for which it inserted a component into the Session-ID header field value. If the intermediary is aware of the UUID value that identifies the endpoint to which a message is directed, it MUST insert that UUID value into the Session-ID header field value as the "remote-uuid" value. If the intermediary is unaware of the UUID value that identifies the receiving endpoint, it MUST use the nil UUID value as the "remote-uuid" value.

If an intermediary receives a SIP message without a Session-ID header field or valid Session-ID header field value from an endpoint for which the intermediary has previously received a Session-ID and is storing a "remote-uuid" value for that endpoint, the lack of a Session-ID must have no effect on what the intermediary believes is the UUID value of the endpoint. That is, the intermediary MUST NOT change the internally maintained "remote-uuid" value for the peer.

When an intermediary originates a response, such as a provisional response or a response to a CANCEL request, the "remote-uuid" field will contain the UUID value of the receiving endpoint. When the UUID of the peer endpoint is known, the intermediary MUST insert the UUID of the peer endpoint in the "local-uuid" field of the header value. Otherwise, the intermediary MAY set the "local-uuid" field of the header value to the "nil" UUID value.



When an intermediary originates a request message without first having received a SIP message that triggered the transmission of the message (e.g., sending a BYE message to terminate a call for policy reasons), the intermediary **MUST**, if it has knowledge of the UUID values for the two communicating endpoints, insert a Session-ID header field with the "remote-uuid" field of the header value set to the UUID value of the receiving endpoint and the "local-uuid" field of the header value set to the UUID value of the other endpoint. When the intermediary does not have knowledge of the UUID value of an endpoint in the communication session, the intermediary **SHOULD** set the unknown UUID value(s) to the "nil" UUID value. (If both are unknown, the Session-ID header value **SHOULD NOT** be included at all, since it would have no practical value.)

With respect to the previous two paragraphs, note that if an intermediary transmits a "nil" UUID value, the receiving endpoint might use that value in subsequent messages it sends. This effectively violates the requirement of maintaining an end-to-end Session Identifier value for the communication session if a UUID for the peer endpoint had been previously conveyed. Therefore, an intermediary **MUST** only send the "nil" UUID when the intermediary has not communicated with the peer endpoint to learn its UUID. This means that intermediaries **SHOULD** maintain state related to the UUID values for both ends of a communication session if it intends to originate messages (versus merely conveying messages). An intermediary that does not maintain this state and that originates a message as described in the previous two paragraphs **MUST NOT** insert a Session-ID header field in order to avoid unintended, incorrect reassignment of a UUID value.

The Session-ID header field value included in a CANCEL request **MUST** be identical to the Session-ID header field value included in the corresponding request being cancelled.

If a SIP intermediary initiates a dialog between two endpoints in a third-party call control (3PCC [RFC3725]) scenario, the SIP request in the initial INVITE will have a non-nil, locally-fabricated "local-uuid" value; call this temporary UUID X. The request will still have a nil "remote-uuid" value; call this value N. The SIP server **MUST** be transaction stateful. The UUID pair in the INVITE will be {X,N}. A 1xx or 2xx response will have a UUID pair {A,X}. This transaction stateful, dialog initiating SIP server **MUST** replace its own UUID, i.e., X, with a nil UUID (i.e., {A,N}) in the INVITE sent towards the other UAS as expected (see Section 10.7 for an example).

Intermediaries that manipulate messages containing a Session-ID header field **SHOULD** be aware of what UUID values it last sent towards an endpoint and, following any kind of service interaction initiated

or affected by the intermediary, of what UUID values the receiving endpoint should have knowledge to ensure that both endpoints in the session have the correct and same UUID values. If an intermediary can determine that an endpoint might not have received a current, correct Session-ID field, the intermediary SHOULD attempt to provide the correct Session-ID header field to the endpoint such as by sending a re-INVITE message. Failure to take such measures may make troubleshooting more difficult because of the mismatched identifiers, therefore it is strongly advised that intermediaries attempt to provide the correct Session Identifier if it able to do so.

If an intermediary receives a SIP response with a non-nil "local-uuid" that is not 32 octets long, this response comes from a misbehaving implementation, and its Session-ID header field MUST be discarded. That said, the response might still be valid according to the rules within SIP [RFC3261], and SHOULD be checked further.

An intermediary MUST assume that the UUID value of session peers may change at any time due to service interactions and MAY itself change UUID values for sessions under its control to ensure end to end session identifiers are consistent for all participants in a session. Section 8 discusses how intermediaries must handle remote UUID changes if they maintain state of the session identifier.

An intermediary may perform protocol interworking between different IP-based communications systems, e.g. interworking between H.323 and SIP. If the intermediary supports the Session Identifier for both protocols for which it is interworking, it SHOULD pass the identifier between the two call legs to maintain an end-to-end identifier regardless of protocol.

## 8. Handling of Remote UUID Changes

It is desirable to have all endpoints and intermediaries involved in a session agree upon the current session identifier when these changes occur. Due to race conditions or certain interworking scenarios, it is not always possible to guarantee session identifier consistency; however, in an attempt to ensure the highest likelihood of consistency, all endpoints and intermediaries involved in a session MUST accept a peer's new UUID under the following conditions:

- o When an endpoint or intermediary receives a mid-dialog request containing a new UUID from a peer, all responses to that request MUST contain the new UUID value as the "remote" parameter unless a subsequent successful transaction (for example, an UPDATE) contains a different UUID, in which case the newest UUID MUST be used.

- o If an endpoint or intermediary sends a successful (2xx) or redirection (3xx) response to the request containing the new UUID value, the endpoint or intermediary MUST accept the peer's UUID and include this new UUID as the "remote" parameter for any subsequent messages unless the UUID from a subsequent transaction has already been accepted. The one exception is a CANCEL request as outlined below.
- o If the endpoint or intermediary sends a failure (4xx, 5xx, 6xx) response, it MUST NOT accept the new UUID value and any subsequent messages MUST contain the previously stored UUID value in the "remote" parameter for any subsequent message. Note that the failure response itself will contain the new UUID value from the request in the "remote" parameter.
- o The ACK method is a special case as there is no response. When an endpoint or intermediary receives an ACK for a successful (2xx) or redirection (3xx) response with a new UUID value, it MUST accept the peer's new UUID value and include this new UUID as the "remote" parameter for any subsequent messages. If the ACK is for a failure (4xx, 5xx, 6xx) response, the new value MUST NOT be used.
- o As stated in Section 6 and Section 7, the Session-ID header field value included in a CANCEL request MUST be identical to the Session-ID header field value included in the corresponding INVITE. Upon receiving a CANCEL request, an endpoint or intermediary would normally send a Request Terminated (487 - see Section 15.1.2 of [RFC3261]) response which, by the rules outlined above, would result in the endpoint or intermediary not storing any UUID value contained in the CANCEL. Section 3.8 of [RFC6141] specifies conditions where a CANCEL can result in 2xx response. Because CANCEL is not passed end-to-end and will always contain the UUID from the original INVITE, retaining a new UUID value received in a CANCEL may result in inconsistency with the Session-ID value stored on the endpoints and intermediaries involved in the session. To avoid this situation, an endpoint or intermediary MUST NOT accept the new UUID value received in a CANCEL and any subsequent messages MUST contain the previously stored UUID value in the "remote" parameter". Note that the response to the CANCEL will contain the UUID value from the CANCEL request in the "remote" parameter.
- o When an endpoint or intermediary receives a response containing a new UUID from a peer, the endpoint or intermediary MUST accept the new UUID as the peer's UUID and include this new UUID as the "remote" parameter for any subsequent messages.

When an intermediary accepts a new UUID from a peer, the intermediary SHOULD attempt to provide the correct Session-ID header field to other endpoints involved in the session, for example, by sending a re-INVITE message. If an intermediary receives a message with a "remote" parameter in the session identifier that does not match the updated UUID, the intermediary MUST update the "remote" parameter with the latest stored UUID.

If an intermediary is performing interworking between two different protocols that both support the Session Identifier defined in this document (e.g. SIP to H.323), UUID changes SHOULD be communicated between protocols to maintain the end-to-end session identifier.

#### 9. Associating Endpoints in a Multipoint Conference

Multipoint Control Units (MCUs) group two or more sessions into a single multipoint conference and have a conference Focus responsible for maintaining the dialogs connected to it [RFC4353]. MCUs, including cascaded MCUs, MUST utilize the same UUID value ("local-uuid" portion of the Session-ID header field value) with all participants in the conference. In so doing, each individual session in the conference will have a unique Session Identifier (since each endpoint will create a unique UUID of its own), but will also have one UUID in common with all other participants in the conference.

When creating a cascaded conference, an MCU MUST convey the UUID value to utilize for a conference via the "local-uuid" portion of the Session-ID header field value in an INVITE to a second MCU when using SIP to establish the cascaded conference. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. [RFC4579] defines the "isfocus" Contact header field value parameter just for this purpose. The initial MCU MUST include the UUID of that particular conference in the "local-uuid" of an INVITE to the other MCU(s) participating in that conference. Also included in this INVITE is an "isfocus" Contact header field value parameter identifying that this INVITE is coming from an MCU and that this UUID is to be given out in all responses from endpoints into those MCUs participating in this same conference. This ensures a single UUID is common across all participating MCUs of the same conference, but is unique between different conferences.

In the case where two existing conferences are joined, there should be a session between the two MCUs where the Session Identifier is comprised of the UUID values of the two conferences. This Session Identifier can be used to correlate the sessions between participants in the joined conference. This specification does not impose any additional requirements when two existing conferences are joined.

Intermediary devices or network diagnostics equipment might assume that when they see two or more sessions with different Session Identifiers, but with one UUID in common, that the sessions are part of the same conference. However, the assumption that two sessions having one common UUID being part of the same conference is not always correct. In a SIP forking scenario, for example, there might also be what appears to be multiple sessions with a shared UUID value; this is intended. The desire is to allow for the association of related sessions, regardless of whether a session is forked or part of a conference.

## 10. Examples of Various Call Flow Operations

Seeing something frequently makes understanding easier. With that in mind, this section includes several call flow examples with the initial UUID and the complete Session Identifier indicated per message, as well as when the Session Identifier changes according to the rules within this document during certain operations/functions.

This section is for illustrative purposes only and is non-normative. In the following flows, RTP refers to the Real-time Transport Protocol [RFC3550].

In the examples in this section, "N" represents a nil UUID and other letters represents the unique UUID values corresponding to endpoints or MCUs.

### 10.1. Basic Call with 2 UUIDs

Session-ID	Alice	B2BUA	Bob	Carol
{A,N}		---	INVITE F1---	
{A,N}			---	INVITE F2---
{B,A}			<---	200 OK F3---
{B,A}		<---	200 OK F4---	
{A,B}		-----	ACK F5-----	
{A,B}			-----	ACK F6-----
		<=====RTP=====		

Figure 1: Session-ID Creation when Alice calls Bob

General operation of this example:

- o UA-Alice populates the "local-uuid" portion of the Session-ID header field value.
- o UA-Alice sends its UUID in the SIP INVITE, and populates the "remote" parameter with a nil value (32 zeros).

- o B2BUA receives an INVITE with both a "local-uuid" portion of the Session-ID header field value from UA-Alice as well as the nil "remote-uuid" value, and transmits the INVITE towards UA-Bob with an unchanged Session-ID header field value.
- o UA-Bob receives Session-ID and generates its "local-uuid" portion of the Session-ID header field value UUID to construct the whole/complete Session-ID header field value, at the same time transferring Alice's UUID unchanged to the "remote-uuid" portion of the Session-ID header field value in the 200 OK SIP response.
- o B2BUA receives the 200 OK response with a complete Session-ID header field value from UA-Bob, and transmits 200 OK towards UA-Alice with an unchanged Session-ID header field value.
- o UA-Alice, upon reception of the 200 OK from the B2BUA, transmits the ACK towards the B2BUA. The construction of the Session-ID header field in this ACK is that of Alice's UUID is the "local-uuid", and Bob's UUID populates the "remote-uuid" portion of the header-value.
- o B2BUA receives the ACK with a complete Session-ID header field from UA-Alice, and transmits ACK towards UA-Bob with an unchanged Session-ID header field value.

Below is a SIP message exchange illustrating proper use of the Session-ID header field. For the sake of brevity, non-essential headers and message bodies are omitted.

F1 INVITE Alice -> B2BUA

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.example.com
;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.example.com>
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: ab30317f1a784dc48ff824d0d3715d86
;remote=00000000000000000000000000000000
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.example.com>
Content-Type: application/sdp
Content-Length: 142
```

(Alice's SDP not shown)

F2 INVITE B2BUA -> Bob

```
INVITE sip:bob@192.168.10.20 SIP/2.0
Via: SIP/2.0/UDP server10.biloxi.example.com
    ;branch=z9hG4bK4b43c2ff8.1
Via: SIP/2.0/UDP pc33.atlanta.example.com
    ;branch=z9hG4bK776asdhds;received=10.1.3.33
Max-Forwards: 69
To: Bob <sip:bob@biloxi.example.com>
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: ab30317f1a784dc48ff824d0d3715d86
    ;remote=00000000000000000000000000000000
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.example.com>
Record-Route: <sip:server10.biloxi.example.com;lr>
Content-Type: application/sdp
Content-Length: 142
```

(Alice's SDP not shown)

F3 200 OK Bob -> B2BUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.example.com
    ;branch=z9hG4bK4b43c2ff8.1;received=192.168.10.1
Via: SIP/2.0/UDP pc33.atlanta.example.com
    ;branch=z9hG4bK776asdhds;received=10.1.3.33
To: Bob <sip:bob@biloxi.example.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: 47755a9de7794ba387653f2099600ef2
    ;remote=ab30317f1a784dc48ff824d0d3715d86
CSeq: 314159 INVITE
Contact: <sip:bob@192.168.10.20>
Record-Route: <sip:server10.biloxi.example.com;lr>
Content-Type: application/sdp
Content-Length: 131
```

(Bob's SDP not shown)

F4 200 OK B2BUA -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.example.com
    ;branch=z9hG4bK776asdhds;received=10.1.3.33
```

To: Bob <sip:bob@biloxi.example.com>;tag=a6c85cf  
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774  
Call-ID: a84b4c76e66710@pc33.atlanta.example.com  
Session-ID: 47755a9de7794ba387653f2099600ef2  
;remote=ab30317f1a784dc48ff824d0d3715d86  
CSeq: 314159 INVITE  
Contact: <sip:bob@192.168.10.20>  
Record-Route: <sip:server10.biloxi.example.com;lr>  
Content-Type: application/sdp  
Content-Length: 131

(Bob's SDP not shown)

F5 ACK Alice -> B2BUA

ACK sip:bob@192.168.10.20 SIP/2.0  
Via: SIP/2.0/UDP pc33.atlanta.example.com  
;branch=z9hG4bKnashds8  
Route: <sip:server10.biloxi.example.com;lr>  
Max-Forwards: 70  
To: Bob <sip:bob@biloxi.example.com>;tag=a6c85cf  
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774  
Call-ID: a84b4c76e66710@pc33.atlanta.example.com  
Session-ID: ab30317f1a784dc48ff824d0d3715d86  
;remote=47755a9de7794ba387653f2099600ef2  
CSeq: 314159 ACK  
Content-Length: 0

F6 ACK B2BUA -> Bob

ACK sip:bob@192.168.10.20 SIP/2.0  
Via: SIP/2.0/UDP server10.biloxi.example.com  
;branch=z9hG4bK4b43c2ff8.2  
Via: SIP/2.0/UDP pc33.atlanta.example.com  
;branch=z9hG4bKnashds8;received=10.1.3.33  
Max-Forwards: 70  
To: Bob <sip:bob@biloxi.example.com>;tag=a6c85cf  
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774  
Call-ID: a84b4c76e66710@pc33.atlanta.example.com  
Session-ID: ab30317f1a784dc48ff824d0d3715d86  
;remote=47755a9de7794ba387653f2099600ef2  
CSeq: 314159 ACK  
Content-Length: 0



The remaining examples in this Section do not display the complete SIP message exchange. Instead, they simply use the set notation described in Section 4.2 to show the Session Identifier exchange throughout the particular call flow being illustrated.

#### 10.2. Basic Call Transfer using REFER

From the example built within Section 10.1, we proceed to this 'Basic Call Transfer using REFER' example. Note that this is a mid-dialog REFER in contrast with the out-of-dialog REFER in Section 10.9.

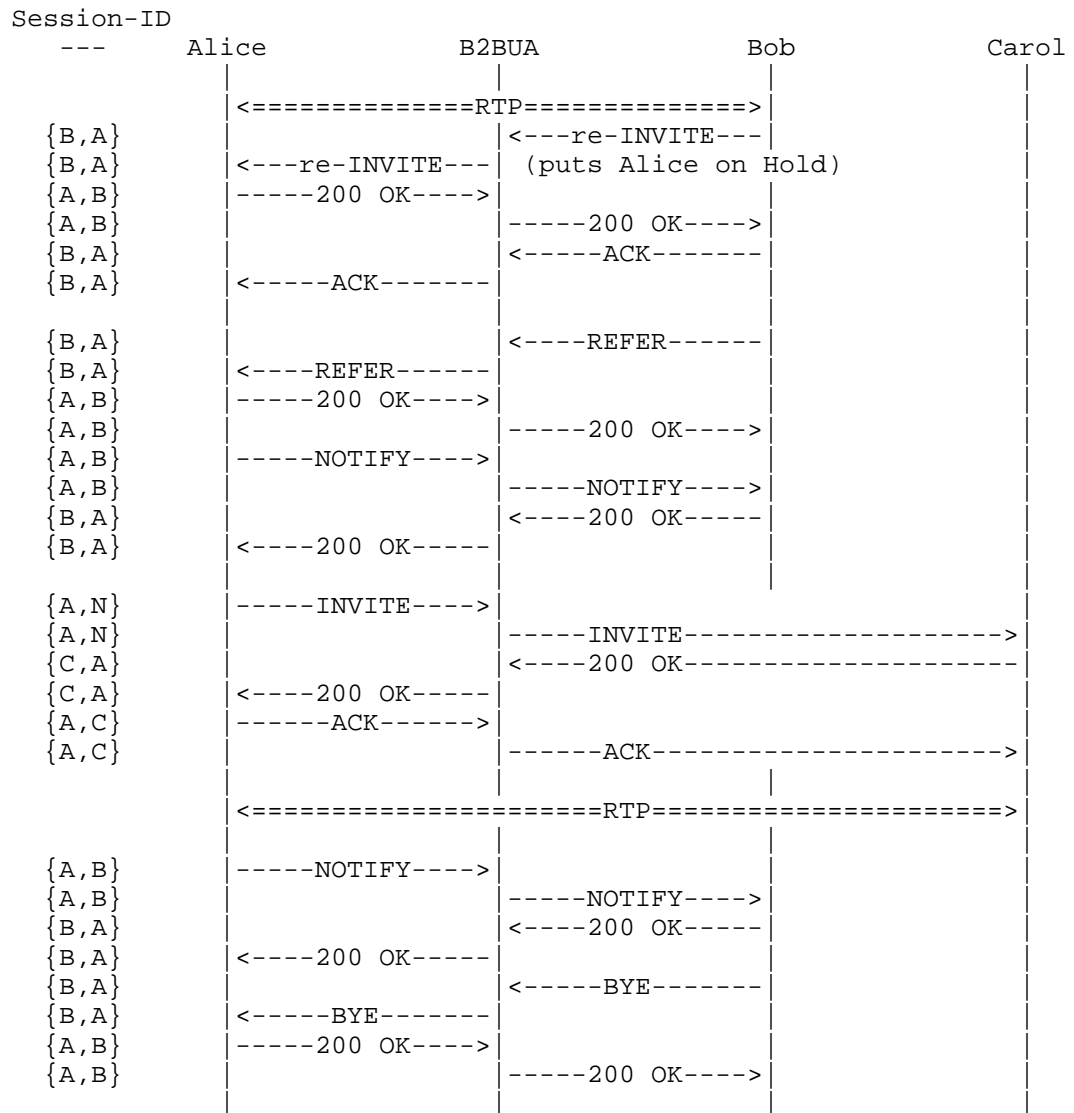


Figure 2: Call Transfer using REFER

General operation of this example:

Starting from the existing Alice/Bob call described in Figure 1 of this document, which established an existing Session-ID header field value:

- o UA-Bob requests Alice to call Carol, using a REFER transaction, as described in [RFC3515]. UA-Alice is initially put on hold, then told in the REFER who to contact with a new INVITE, in this case UA-Carol. This Alice-to-Carol dialog will have a new Call-ID, therefore it requires a new Session-ID header field value. The wrinkle here is we can, and will, use Alice's UUID from her existing dialog with Bob in the new INVITE to Carol.
- o UA-Alice retains her UUID from the Alice-to-Bob call {A} when requesting a call with UA-Carol. This is placed in the "local-uuid" portion of the Session-ID header field value, at the same time inserting a nil "remote-uuid" value (because Carol's UA has not yet received the UUID value). This same UUID traverses the B2BUA unchanged.
- o UA-Carol receives the INVITE with a Session Identifier UUID {A,N}, replaces the A UUID value into the "remote-uuid" portion of the Session-ID header field value and creates its own UUID {C} and places this value in the "local-uuid" portion of the Session-ID header field value, thereby removing the N(nil) value altogether. This combination forms a full Session Identifier {C,A} in the 200 OK to the INVITE. This Session-ID header field traverses the B2BUA unchanged towards UA-Alice.
- o UA-Alice receives the 200 OK with the Session Identifier {C,A} and responds to UA-Carol with an ACK (just as in Figure 1 - switches places of the two UUID fields), and generates a NOTIFY to Bob with a Session Identifier {A,B} indicating the call transfer was successful.
- o It does not matter which UA terminates the Alice-to-Bob call; Figure 2 shows UA-Bob doing this transaction.

### 10.3. Basic Call Transfer using re-INVITE

From the example built within Section 10.1, we proceed to this 'Basic Call Transfer using re-INVITE' example.

Alice is talking to Bob. Bob pushes a button on his phone to transfer Alice to Carol via the B2BUA (using re-INVITE).

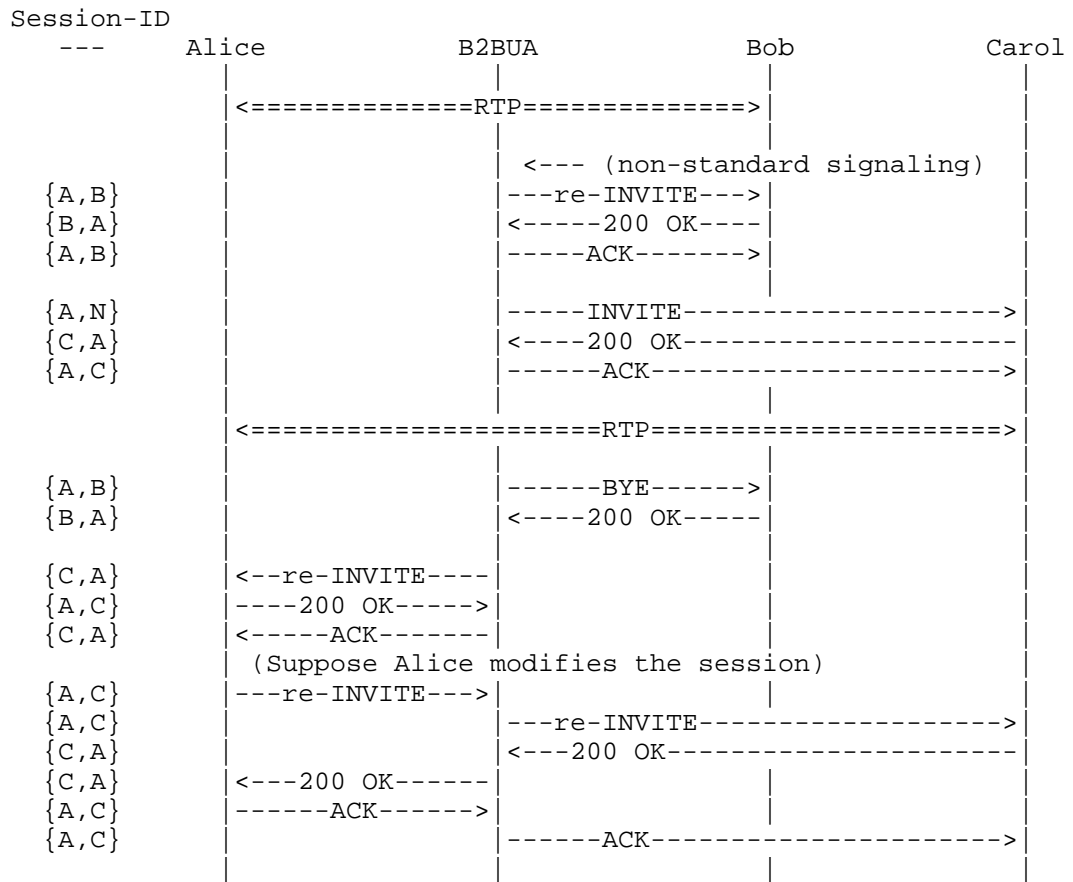


Figure 3: Call transfer using re-INVITE

General operation of this example:

- o We assume the call between Alice and Bob from Section 10.1 is operational with Session Identifier {A,B}.
- o Bob uses non-standard signaling to the B2BUA to initiate a call transfer from Alice to Carol. This could also be initiated via a REFER message from Bob, but the signaling that follows might still be similar to the above flow. In either case, Alice is completely unaware of the call transfer until a future point in time when Alice receives a message from Carol.
- o The B2BUA sends a re-INVITE with the Session Identifier {"local-uuid" = "A", "remote-uuid" = "B"} to re-negotiate the session with Bob.

- o The B2BUA sends a new INVITE with Alice's UUID {"local-uuid" = "A"} to Carol.
- o Carol receives the INVITE and accepts the request and adds her UUID {C} to the Session Identifier for this session {"local-uuid" = "C", "remote-uuid" = "A"}.
- o The B2BUA then terminates the call to Bob with a BYE using the Session Identifier {"local-uuid" = "A", "remote-uuid" = "B"}.
- o The B2BUA sends a re-INVITE to Alice to update Alice's view of the Session Identifier.
- o When Alice later attempts to modify the session with a re-INVITE, Alice will send "remote-uuid" = "C" toward Carol because it had previously received the updated UUID in the re-INVITE from the B2BUA. The B2BUA maintains the Session Identifier {"local-uuid" = "A", "remote-uuid" = "C"}. Carol replies with the "local-uuid" = "C", "remote-uuid" = "A" to reflect what was received in the INVITE (which Carol already knew from previous exchanges with the B2BUA). Alice then includes "remote-uuid" = "C" in the subsequent ACK message.

#### 10.4. Single Focus Conferencing

Multiple users call into a conference server (say, an MCU) to attend one of many conferences hosted on or managed by that server. Each user has to identify which conference they want to join, but this information is not necessarily in the SIP messaging. It might be done by having a dedicated address for the conference or via an IVR, as assumed in this example and depicted with the use of M1, M2, and M3. Each user in this example goes through a two-step process of signaling to gain entry onto their conference call, which the conference focus identifies as M'.

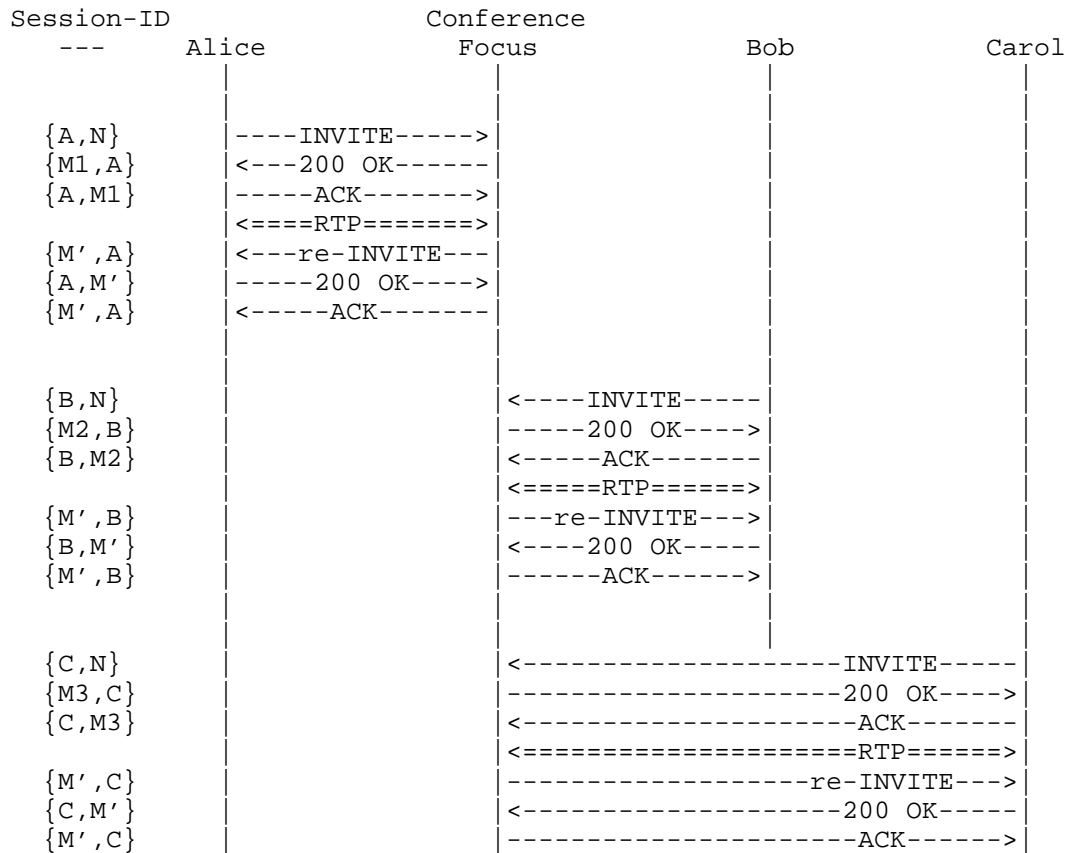


Figure 4: Single Focus Conference Bridge

General operation of this example:

Alice calls into a conference server to attend a certain conference. This is a two-step operation since Alice cannot include the conference ID at this time and/or any passcode in the INVITE request. The first step is Alice's UA calling another UA to participate in a session. This will appear to be similar as the call-flow in Figure 1 (in section 10.1). What is unique about this call is the second step: the conference server sends a re-INVITE request with its second UUID, but maintaining the UUID Alice sent in the first INVITE. This subsequent UUID from the conference server will be the same for each UA that calls into this conference server participating in this same conference bridge/call, which is generated once Alice typically authenticates and identifies which bridge she wants to participate on.

- o Alice sends an INVITE to the conference server with her UUID {A} and a "remote-uuid" = N.
- o The conference server responds with a 200 OK response which replaces the N UUID with a temporary UUID ("M1") as the "local-uuid" and a "remote-uuid" = "A".

NOTE: this 'temporary' UUID is a real UUID; it is only temporary to the conference server because it knows that it is going to generate another UUID to replace the one just send in the 200 OK.

- o Once Alice, the user, gains access to the IVR for this conference server, she enters a specific conference ID and whatever passcode (if needed) to enter a specific conference call.
- o Once the conference server is satisfied Alice has identified which conference she wants to attend (including any passcode verification), the conference server re-INVITES Alice to the specific conference and includes the Session-ID header field value component "local-uuid" = "M'" (and "remote-uuid" = "A") for that conference. All valid participants in the same conference will receive this same UUID for identification purposes and to better enable monitoring, and tracking functions.
- o Bob goes through this two-step process of an INVITE transaction, followed by a re-INVITE transaction to get this same UUID ("M'") for that conference.
- o In this example, Carol (and each additional user) goes through the same procedures and steps as Alice and Bob to get on this same conference.

#### 10.5. Single Focus Conferencing using a web-based conference service

Alice, Bob and Carol call into same web-based conference. Note this this is one of many ways of implementing this functionality and should not be construed as the preferred way of establishing a web-based conference.

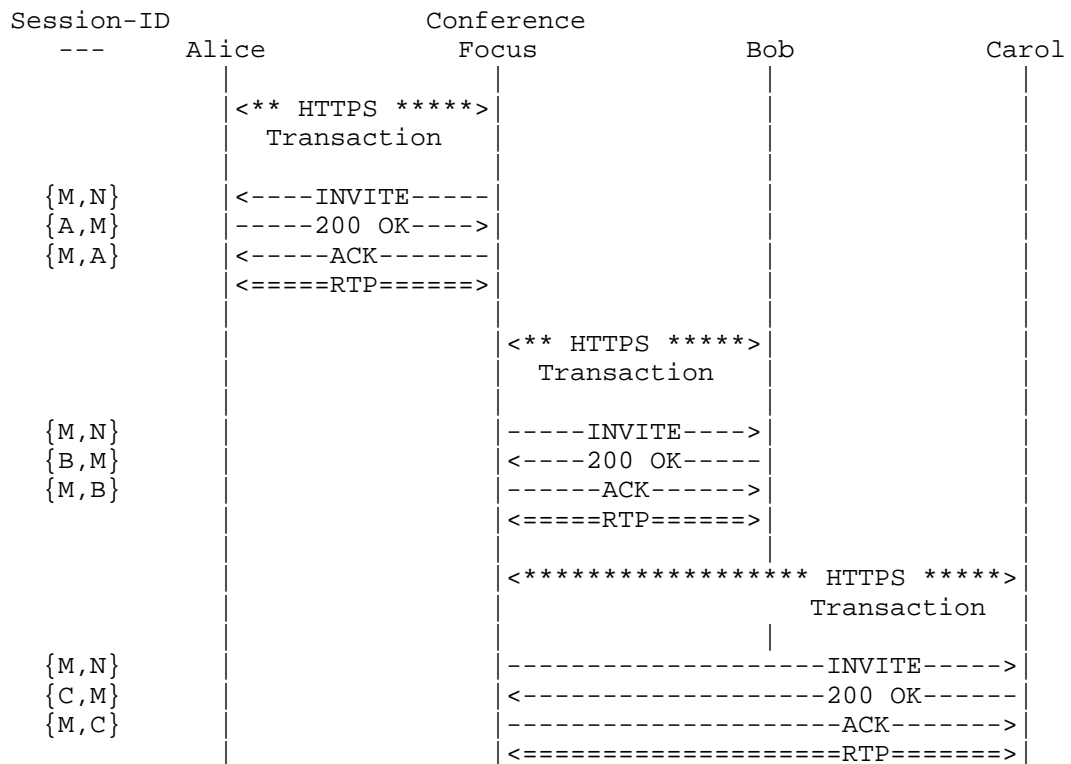


Figure 5: Single Focus Web-based Conference

General operation of this example:

- o Alice communicates with web server with desire to join a certain meeting, by meeting number; also includes UA-Alice's contact information (phone number, URI and/or IP address, etc.) for each device she wants for this conference call. For example, the audio and video play-out devices could be separate units.
- o Conference Focus server sends INVITE (Session-ID header field value components "local-uuid" = M and a remote UUID of N, where M equals the "local-uuid" for each participant on this conference bridge) to UA-Alice to start session with that server for this A/V conference call.
- o Upon receiving the INVITE request from the conference focus server, Alice responds with a 200 OK. Her UA moves the "local-uuid" unchanged into the "remote-uuid" field, and generates her own UUID and places that into the "local-uuid" field to complete the Session-ID construction.



- o Bob and Carol perform same function to join this same A/V conference call as Alice.

## 10.6. Cascading Conference Bridges

### 10.6.1. Establishing a Cascaded Conference

To expand conferencing capabilities requires cascading conference bridges. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. [RFC4579] defines the 'isfocus' Contact: header parameter just for this purpose.

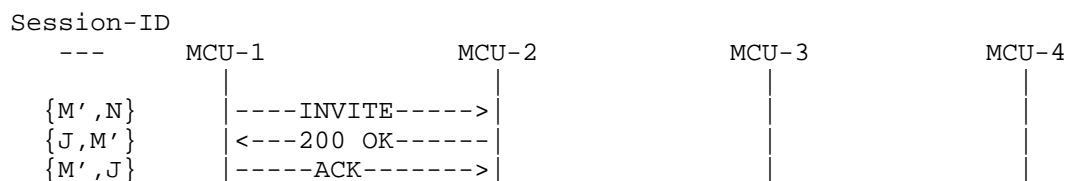


Figure 6: MCUs Communicating Session Identifier UUID for Bridge

Regardless of which MCU (1 or 2) a UA contacts for this conference, once the above exchange has been received and acknowledged, the UA will get the same {M',N} UUID pair from the MCU for the complete Session Identifier.

A more complex form would be a series of MCUs all being informed of the same UUID to use for a specific conference. This series of MCUs can either be informed

- o All by one MCU (that initially generates the UUID for the conference).
- o The MCU that generates the UUID informs one or several MCUs of this common UUID, and they inform downstream MCUs of this common UUID that each will be using for this one conference.

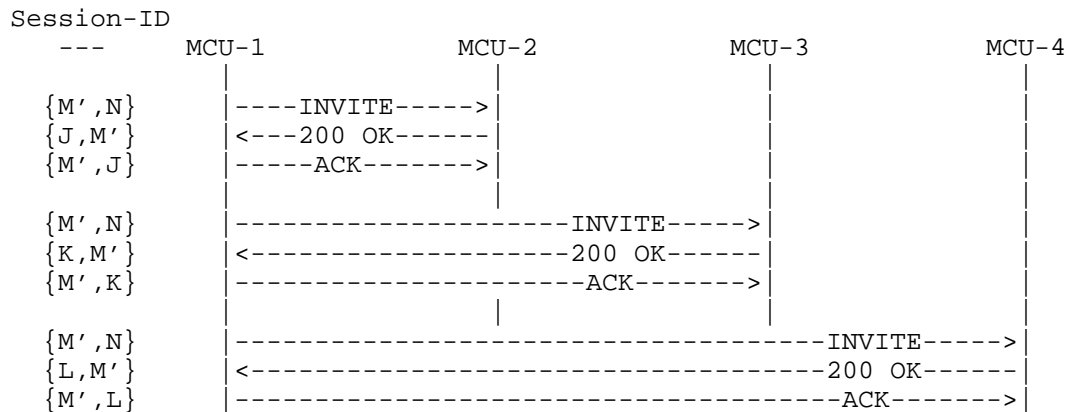


Figure 7: MCU Communicating Session Identifier UUID to More than One MCU

General operation of this example:

- o The MCU generating the Session Identifier UUID communicates this in a separate INVITE, having a Contact header with the 'isfocus' header parameter. This will identify the MCU as what [RFC4579] calls a conference-aware SIP entity.
- o An MCU that receives this {M',N} UUID pair in an inter-MCU transaction can communicate the M' UUID in a manner in which it was received to construct a hierarchical cascade (though this time this second MCU would be the UAC MCU).
- o Once the conference is terminated, the cascaded MCUs will receive a BYE message to terminate the cascade.

#### 10.6.2. Calling into Cascaded Conference Bridges

Here is an example of how a UA, say Robert, calls into a cascaded conference focus. Because MCU-1 has already contacted MCU-3, the MCU where Robert is going to join the conference, MCU-3 already has the Session-ID (M') for this particular conference call.

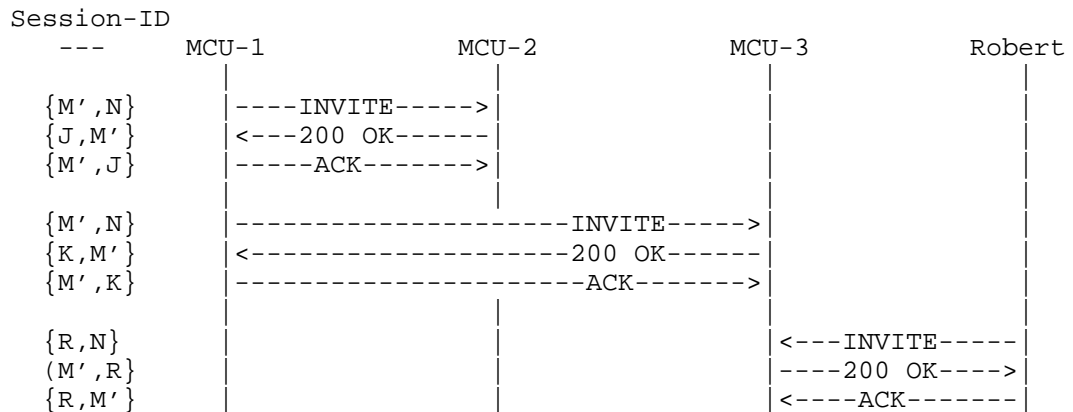


Figure 8: A UA Calling into a Cascaded MCU UUID

General operation of this example:

- o The UA, Robert in this case, INVITEs the MCU to join a particular conference call. Robert's UA does not know anything about whether this is the main MCU of the conference call, or a cascaded MCU. Robert likely does not know MCUs can be cascaded, he just wants to join a particular call. Like as with any standard implementation, he includes a nil "remote-uuid".
- o The cascaded MCU, upon receiving this INVITE from Robert, replaces the nil UUID with the UUID value communicated from MCU-1 for this conference call as the "local-uuid" in the SIP response. Thus, moving Robert's UUID "R" to the "remote-uuid" value.
- o The ACK has the Session-ID {R, M'}, completing the 3-way handshake for this call establishment. Robert has now joined the conference call originated from MCU-1.
- o Once the conference is terminated, the cascaded MCUs will receive a BYE message to terminate the cascade.

#### 10.7. Basic 3PCC for two UAs

An external entity sets up calls to both Alice and Bob for them to talk to each other.

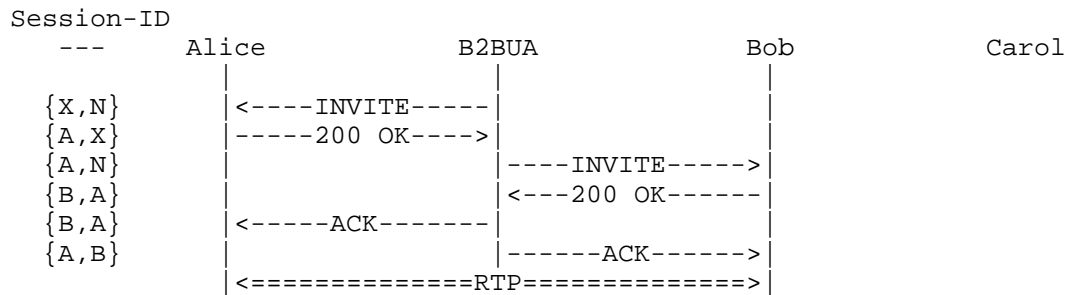


Figure 9: 3PCC initiated call between Alice and Bob

General operation of this example:

- o Some out of band procedure directs a B2BUA (or other SIP server) to have Alice and Bob talk to each other. In this case, the SIP server has to be transaction stateful, if not dialog stateful.
- o The SIP server INVITEs Alice to a session and uses a temporary UUID {X} and a nil UUID pairing.
- o Alice receives and accepts this call set-up and replaces the nil UUID with her UUID {A} in the Session Identifier, now {A,X}.
- o The transaction stateful SIP server receives Alice's UUID {A} in the local UUID portion and keeps it there, and discards its own UUID {X}, replacing this with a nil UUID value in the INVITE to Bob as if this came from Alice originally.
- o Bob receives and accepts this INVITE and adds his own UUID {B} to the Session Identifier, now {B,A} for the response.
- o The session is established.

#### 10.8. Handling in 100 Trying SIP Response and CANCEL Request

The following two subsections show examples of the Session Identifier for a 100 Trying response and a CANCEL request in a single call-flow.

##### 10.8.1. Handling in a 100 Trying SIP Response

The following 100 Trying response is taken from an existing RFC, from [RFC5359] Section 2.9 ("Call Forwarding - No Answer").

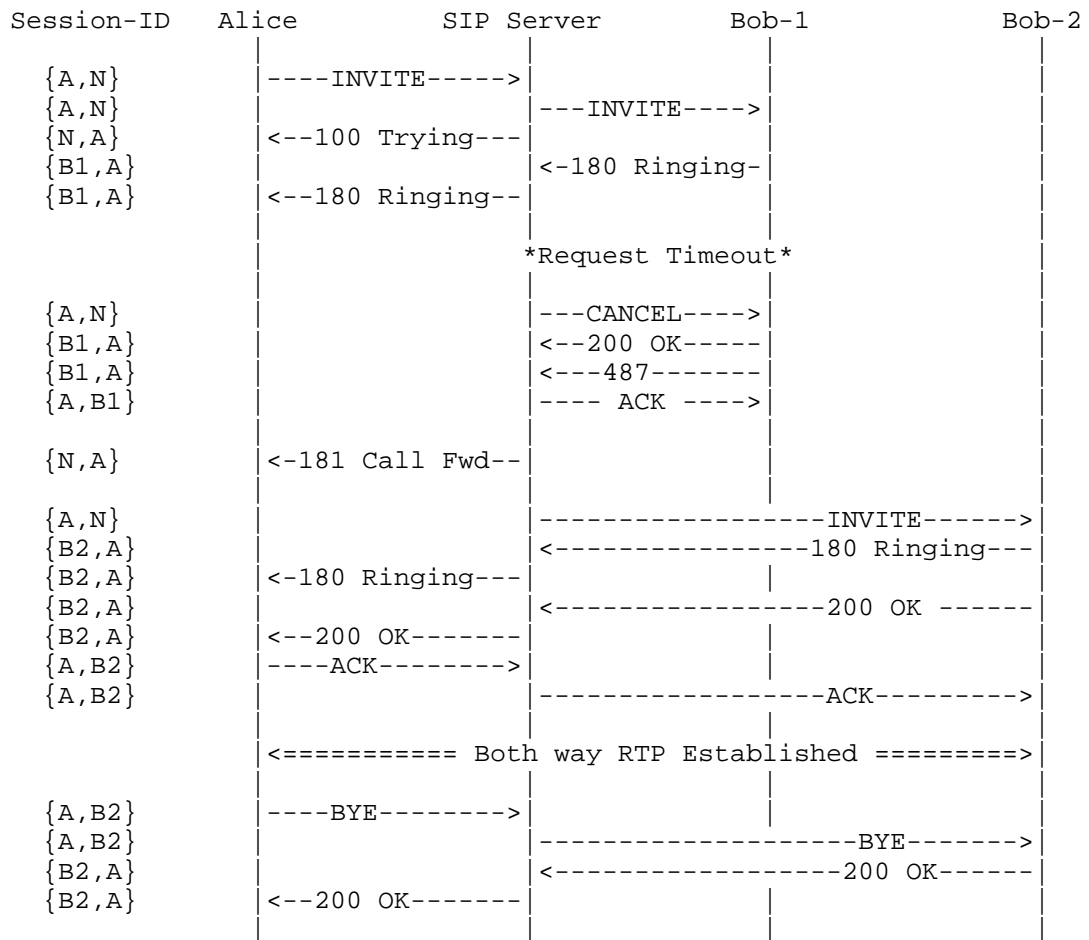


Figure 10: Session Identifier in the 100 Trying and CANCEL Messaging

Below is the explanatory text from RFC 5359 Section 2.9 detailing what the desired behavior is in the above call flow (i.e., what the call-flow is attempting to achieve).

"Bob wants calls to B1 forwarded to B2 if B1 is not answered (information is known to the SIP server). Alice calls B1 and no one answers. The SIP server then places the call to B2."

General operation of this example:

- o Alice generates an INVITE request because she wants to invite Bob to join her session. She creates a UUID as described in section 10.1, and places that value in the "local-uuid" field of the

Session-ID header field value. Alice also generates a "remote-uuid" of nil and sends this along with the "local-uuid".

- o The SIP server (imagine this is a B2BUA), upon receiving Alice's INVITE, generates the optional provisional response 100 Trying. Since the SIP server has no knowledge Bob's UUID for his part of the Session Identifier value, it cannot include his "local-uuid". Rather, any 100 Trying response includes Alice's UUID in the "remote-uuid" portion of the Session-ID header-value with a nil "local-uuid" value in the response. This is consistent with what Alice's UA expects to receive in any SIP response containing this UUID.

#### 10.8.2. Handling a CANCEL SIP Request

In the same call-flow example as the 100 Trying response is a CANCEL request. Please refer to Figure 10 for the CANCEL request example.

General operation of this example:

- o In Figure 10 above, Alice generates an INVITE with her UUID value in the Session-ID header field.
- o Bob-1 responds to this INVITE with a 180 Ringing. In that response, he includes his UUID in the Session-ID header field value (i.e., {B1,A}); thus completing the Session-ID header field for this session, even though no final response has been generated by any of Bob's UAs.
- o While this means that if the SIP server were to generate a SIP request within this session it could include the complete SessionID, the server sends a CANCEL and a CANCEL always uses the same Session-ID header field as the original INVITE. Thus, the CANCEL would have a Session Identifier with the "local-uuid" = "A", and the "remote-uuid" = "N".
- o As it happens with this CANCEL, the SIP server intends to invite another UA of Bob (i.e., B2) for Alice to communicate with.
- o In this example call-flow, taken from RFC 5359, Section 2.9, a 181 (Call is being Forwarded) response is sent to Alice. Since the SIP server generated this SIP request, and has no knowledge of Bob-2's UUID value, it cannot include that value in this 181. Thus, and for the exact reasons the 100 Trying including the Session Identifier value, only Alice's UUID is included in the remote-uuid component of the Session-ID header field value, with a nil UUID present in the "local-uuid" component.

## 10.9. Out-of-dialog REFER Transaction

The following call-flow was extracted from Section 6.1 of [RFC5589] ("Successful Transfer"), with the only changes being the names of the UAs to maintain consistency within this document.

Alice is the transferee  
 Bob is the transferer  
 and Carol is the transfer-target

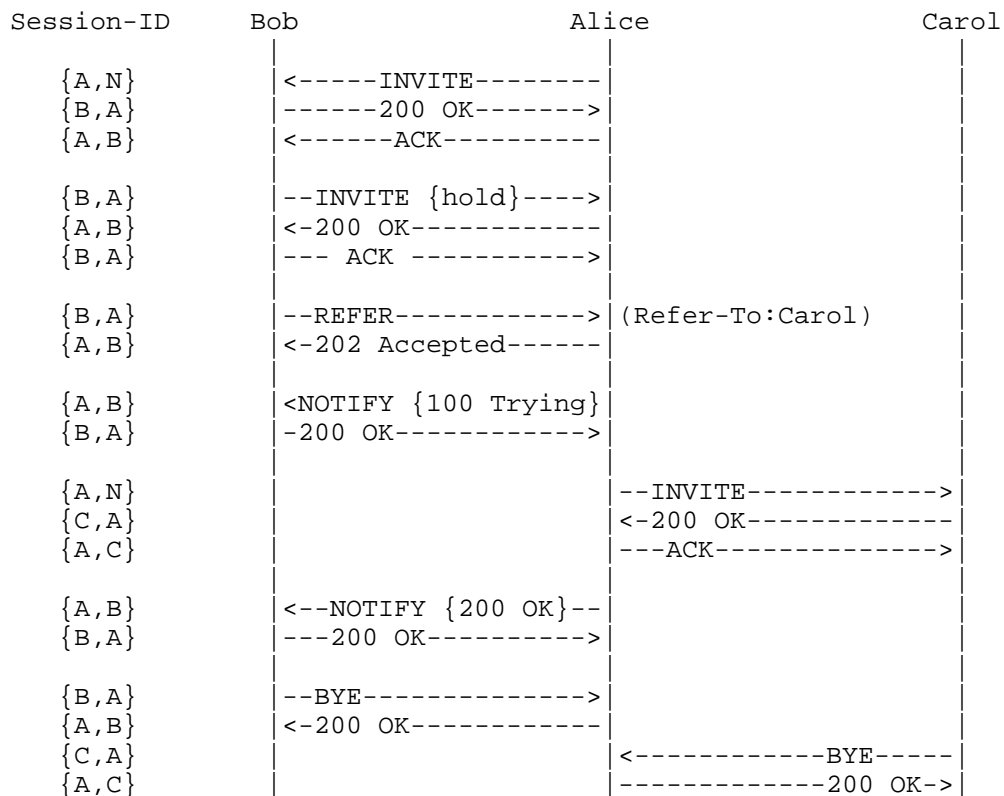


Figure 11: Out-Of-Dialog Call Transfer

General operation of this example:

- o Just as in Section 10.2, Figure 2, Alice invites Bob to a session, and Bob eventually transfers Alice to communicate with Carol.
- o What is different about the call-flow in Figure 11 is that Bob's REFER is not in-dialog. Even so, this is treated as part of the

same communication session and, thus, the Session Identifier in those messages is {A,B}.

- o Alice will use her existing UUID and the nil UUID ({A,N}) in the INVITE towards Carol (who generates UUID "C" for this session), thus maintaining the common UUID within the Session Identifier for this new Alice-to-Carol session.

## 11. Compatibility with a Previous Implementation

There is a much earlier document that specifies the use of a Session-ID header field (namely, [RFC7329]) that we will herewith attempt to achieve backwards compatibility. Neither Session-ID header field has any versioning information, so merely adding that this document describes "version 2" is insufficient. Here are the set of rules for compatibility between the two specifications. Although the previous version was never standardized, it has been heavily implemented and adopted by other standards development organizations. For the purposes of this discussion, we will label the pre-standard specification of the Session-ID as the "old" version and this specification as the "new" version of the Session-ID.

The previous (i.e., "old") version only has a single UUID value as a Session-ID header field value, but has a generic-parameter value that can be of use.

In order to have an "old" version talk to an "old" version implementation, nothing needs to be done as far as the IETF is concerned.

In order to have a "new" version talk to a "new" version implementation, both implementations need to follow this document (to the letter) and everything should be just fine.

For this "new" implementation to work with the "old" implementation and an "old" implementation to work with "new" implementations, there needs to be a set of rules that all "new" implementations MUST follow if the "new" implementation will be communicating with devices that have implemented the "old" implementation.

- o Since no option tags or feature tags are to be used for distinguishing versions, the presence and order of any "remote-uuid" value within the Session-ID header field value is to be used to distinguish implementation versions.
- o If a SIP request has a "remote-uuid" value, this comes from a standard implementation, and not a pre-standard one.



- o If a SIP request has no "remote-uuid" value, this comes from a pre-standard implementation, and not a standard one. In this case, one UUID is used to identify this dialog, even if the responder is a standard implementation of this specification.
- o If a SIP response has a non-nil "local-uuid" that is 32 octets long and differs from the endpoint's own UUID value, this response comes from a standard implementation.
- o If a SIP response arrives that has the same value of Session-ID UUIDs in the same order as was sent, this comes from a pre-standard implementation, and MUST NOT be discarded even though the "remote-uuid" may be nil. In this case, any new transaction within this dialog MUST preserve the order of the two UUIDs within all Session-ID header fields, including the ACK, until this dialog is terminated.
- o If a SIP response only contains the "local-uuid" that was sent originally, this comes from a pre-standard implementation and MUST NOT be discarded for removing the nil "remote-uuid". In this case, all future transactions within this dialog MUST contain only the UUID received in the first SIP response. Any new transaction starting a new dialog from the standard Session-ID implementation MUST include a "local-uuid" and a nil "remote-uuid", even if that new dialog is between the same two UAs.
- o Standard implementations should not expect pre-standard implementations to be consistent in their implementation, even within the same dialog. For example, perhaps the first, third and tenth responses contain a "remote-uuid", but all the others do not. This behavior MUST be allowed by implementations of this specification.
- o The foregoing does not apply to other, presently unknown parameters that might be defined in the future. They are ignored for the purposes of interoperability with previous implementations.

## 12. Security and Privacy Considerations

The Session Identifier MUST be constructed in such a way that does not convey any user or device information as outlined in Section 4.1. This ensures that the data contained in the Session Identifier itself does not convey user or device information, however the Session Identifier may reveal relationships between endpoints that might not be revealed by messages without a Session Identifier.

Section 4.2 requires that a UA always generate a new, previously unused, UUID when transmitting a request to initiate a new session. This ensures that two unrelated sessions originating from the same UA will never have the same UUID value, thereby removing the ability for an attacker to use the Session Identifier to identify the two unrelated sessions as being associated with the same user.

Because of the inherent property that Session Identifiers are conveyed end-to-end and remain unchanged by a UA for the duration of a session, the Session Identifier could be misused to discover relationships between two or more parties when multiple parties are involved in the same session such as the case of a redirect, transfer, or conference. For example, suppose that Alice calls Bob and Bob, via his PBX (acting as a B2BUA), forwards or transfers the call to Carol. Without use of the Session Identifier, an unauthorized third party that is observing the communications between Alice and Bob might not know that Alice is actually communicating with Carol. If Alice, Bob, and Carol include the Session Identifier as a part of the signaling messages, it is possible for the third party to observe that the UA associated with Bob changed to some other UA. If the third party also has access to signaling messages between Bob and Carol, the third party can then discover that Alice is communicating with Carol. This would be true even if all other information relating to the session is changed by the PBX, including both signaling information and media address information. That said, the Session Identifier would not reveal the identity of Alice, Bob, or Carol. It would only reveal the fact that those endpoints were in associated with the same session.

This document allows for additional parameters (generic-param) to be included in the Session-ID header. This is done to allow for future extensions while preserving backward compatibility with this document. To protect privacy, the data for any generic-param included in the Session-ID header value MUST NOT include any user or device information. Additionally, any information conveyed through an additional parameter MUST NOT persist beyond the current session and therefore MUST NOT be reused between unrelated sessions. Additional parameters MAY be used by future extensions of this document to correlate related communication sessions that cannot already be correlated by the procedures described in this document as long as the requirements regarding privacy and persistence defined above are followed."

An intermediary implementing a privacy service that provides user privacy as per Section 5.3 of [RFC3323] MAY choose to consider the Session-ID header as being a non-essential informational header with the understanding that doing so will impair the ability to use the Session Identifier for troubleshooting purposes.

### 13. IANA Considerations

#### 13.1. Registration of the "Session-ID" Header Field

The following is the registration for the 'Session-ID' header field to the "Header Name" registry at

<http://www.iana.org/assignments/sip-parameters>:

RFC number: RFC XXXX

Header name: 'Session-ID'

Compact form: none

Note: This document replaces the "Session-ID" header originally registered via [RFC7329].

[RFC Editor: Please replace XXXX in this section and the next with the this RFC number of this document.]

#### 13.2. Registration of the "remote" Parameter

The following parameter is to be added to the "Header Field Parameters and Parameter Values" section of the SIP parameter registry:

Header Field	Parameter Name	Predefined Values	Reference
Session-ID	remote	No	[RFCXXXX]

### 14. Acknowledgements

The authors would like to thank Robert Sparks, Hadriel Kaplan, Christer Holmberg, Paul Kyzivat, Brett Tate, Keith Drage, Mary Barnes, Charles Eckel, Peter Dawes, Andrew Hutton, Arun Arunachalam, Adam Gensler, Roland Jesske, and Faisal Siyavudeen for their invaluable comments during the development of this document.

### 15. Dedication

This document is dedicated to the memory of James Polk, a long-time friend and colleague. James made important contributions to this specification, including being one of its primary editors. The IETF global community mourns his loss and he will be missed dearly.

## 16. References

### 16.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<http://www.rfc-editor.org/info/rfc3261>>.
- [RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, DOI 10.17487/RFC3515, April 2003, <<http://www.rfc-editor.org/info/rfc3515>>.
- [RFC3891] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, DOI 10.17487/RFC3891, September 2004, <<http://www.rfc-editor.org/info/rfc3891>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<http://www.rfc-editor.org/info/rfc4122>>.
- [RFC4579] Johnston, A. and O. Levin, "Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents", BCP 119, RFC 4579, DOI 10.17487/RFC4579, August 2006, <<http://www.rfc-editor.org/info/rfc4579>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC7206] Jones, P., Salgueiro, G., Polk, J., Liess, L., and H. Kaplan, "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks", RFC 7206, DOI 10.17487/RFC7206, May 2014, <<http://www.rfc-editor.org/info/rfc7206>>.

## 16.2. Informative References

- [H.323] International Telecommunications Union, "Recommendation ITU-T H.323, Packet-based multimedia communications systems", December 2009.
- [H.460.27] International Telecommunications Union, "Recommendation ITU-T H.460.27, End-to-End Session Identifier for H.323 Systems", November 2015.
- [RFC2543] Handley, M., Schulzrinne, H., Schooler, E., and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, DOI 10.17487/RFC2543, March 1999, <<http://www.rfc-editor.org/info/rfc2543>>.
- [RFC3323] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, DOI 10.17487/RFC3323, November 2002, <<http://www.rfc-editor.org/info/rfc3323>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<http://www.rfc-editor.org/info/rfc3550>>.
- [RFC3725] Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)", BCP 85, RFC 3725, DOI 10.17487/RFC3725, April 2004, <<http://www.rfc-editor.org/info/rfc3725>>.
- [RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, DOI 10.17487/RFC4353, February 2006, <<http://www.rfc-editor.org/info/rfc4353>>.
- [RFC5359] Johnston, A., Ed., Sparks, R., Cunningham, C., Donovan, S., and K. Summers, "Session Initiation Protocol Service Examples", BCP 144, RFC 5359, DOI 10.17487/RFC5359, October 2008, <<http://www.rfc-editor.org/info/rfc5359>>.
- [RFC5589] Sparks, R., Johnston, A., Ed., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, DOI 10.17487/RFC5589, June 2009, <<http://www.rfc-editor.org/info/rfc5589>>.

- [RFC6141] Camarillo, G., Ed., Holmberg, C., and Y. Gao, "Re-INVITE and Target-Refresh Request Handling in the Session Initiation Protocol (SIP)", RFC 6141, DOI 10.17487/RFC6141, March 2011, <<http://www.rfc-editor.org/info/rfc6141>>.
- [RFC6872] Gurbani, V., Ed., Burger, E., Ed., Anjali, T., Abdelnur, H., and O. Festor, "The Common Log Format (CLF) for the Session Initiation Protocol (SIP): Framework and Information Model", RFC 6872, DOI 10.17487/RFC6872, February 2013, <<http://www.rfc-editor.org/info/rfc6872>>.
- [RFC7092] Kaplan, H. and V. Pascual, "A Taxonomy of Session Initiation Protocol (SIP) Back-to-Back User Agents", RFC 7092, DOI 10.17487/RFC7092, December 2013, <<http://www.rfc-editor.org/info/rfc7092>>.
- [RFC7329] Kaplan, H., "A Session Identifier for the Session Initiation Protocol (SIP)", RFC 7329, DOI 10.17487/RFC7329, August 2014, <<http://www.rfc-editor.org/info/rfc7329>>.

## Authors' Addresses

Paul E. Jones  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA

Phone: +1 919 476 2048  
Email: [paulej@packetizer.com](mailto:paulej@packetizer.com)

Gonzalo Salgueiro  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA

Phone: +1 919 392 3266  
Email: [gsalguei@cisco.com](mailto:gsalguei@cisco.com)

Chris Pearce  
Cisco Systems, Inc.  
2300 East President George Bush Highway  
Richardson, TX 75082  
USA

Phone: +1 972 813 5123  
Email: chrep@cisco.com

Paul Giralt  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA

Phone: +1 919 991 5644  
Email: pgiralt@cisco.com