

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: April 21, 2014

P. Dawes  
Vodafone Group  
October 18, 2013

Requirements for Marking SIP Messages to be Logged  
draft-dawes-dispatch-logme-reqs-03

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 April 21, 2014.

## Copyright Notice

Copyright (c) 2013 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. 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. Debug server address . . . . .	7
6.2.3. Sending logged information . . . . .	7
7. Potential Solutions . . . . .	7
7.1. Functionality Common to all Solutions . . . . .	7
7.1.1. Starting and Stopping log-me marking . . . . .	7
7.1.1.1. General . . . . .	7
7.1.1.2. Configuration for log-me marking . . . . .	7
7.1.1.3. Maintaining State . . . . .	8
7.1.2. Sending logs to a debug server . . . . .	10
7.1.2.1. Server address . . . . .	10
7.1.2.2. Protecting logs . . . . .	10
7.2. Solution A: LogMe header field . . . . .	10
7.2.1. Introduction . . . . .	10
7.2.2. Server Procedures . . . . .	10
7.2.2.1. General . . . . .	10
7.2.2.2. Sending logged information to a debug server . .	10
7.2.3. Client Procedures . . . . .	10
7.2.3.1. General . . . . .	11
7.2.4. Identifying test cases . . . . .	11
7.2.5. Collecting logged information at a debug server . .	11
7.2.6. Examples . . . . .	11
7.3. Solution B: New Value for purpose header field parameter	

in Call-Info: . . . . .	12
7.4. Solution C: New 'debug' header field parameter to be used in Session-ID header field . . . . .	13
7.5. Comparison of Potential Solutions . . . . .	13
8. References . . . . .	14
8.1. Normative References . . . . .	14
8.2. Informative References . . . . .	14
Appendix A. Additional Stuff . . . . .	15
Author's Address . . . . .	15

## 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.

+-----+ +-----+

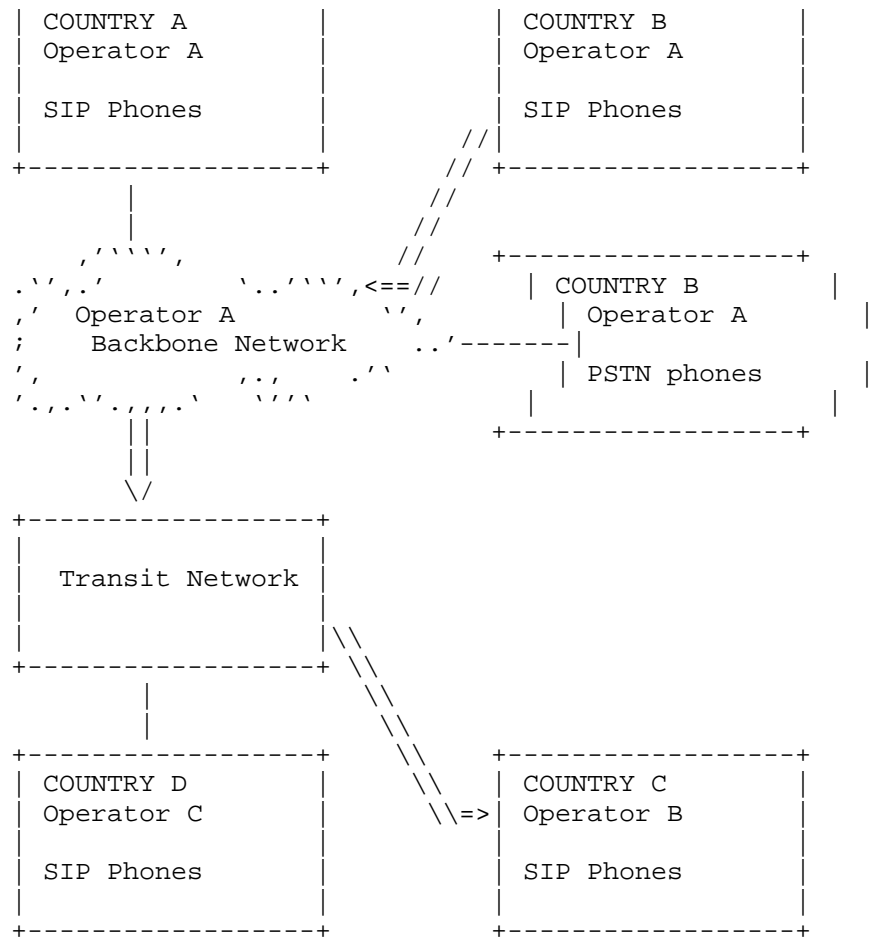


Figure 1: Example signalling path through multiple networks

#### 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.
- o REQ9: A log-me marker may include a locator of the server that collects logs. This locator is known as the diagnostic server identifier and may be an address of a server. A SIP entity can use the diagnostic server identifier to send collected logs to the diagnostic server.

## 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. 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. Potential Solutions

This section describes potential solutions to the logme requirements and functionality that is common to all solutions.

#### 7.1. Functionality Common to all Solutions

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

###### 7.1.1.1. General

A server or client needs to determine when to perform log-me marking. A client or server determines whether to perform log-me marking only by configuration. A regression test might be configured to log-me mark all SIP requests for a given time period whereas a troubleshooting test might be configured to mark sessions based on criteria specific to a reported fault. When configuration has caused a client or server to start log-me marking requests and responses in a dialog, marking continues until the dialog ends or until configuration indicates that marking must stop earlier, for example after certain time period has elapsed.

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

Configuration of a client or server 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.

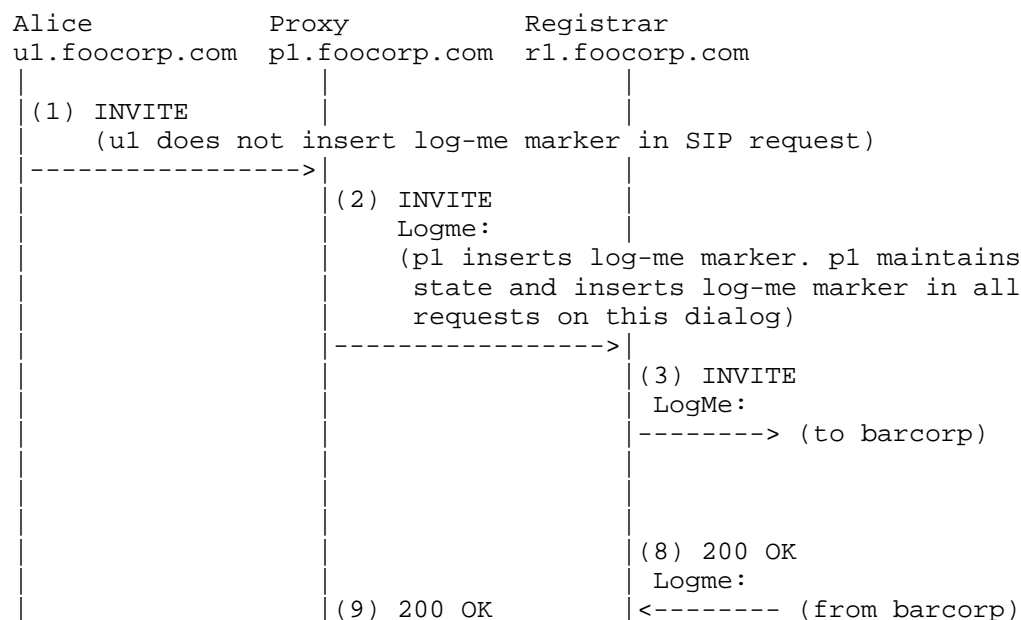
```
<start>09:00:00</start>
<stop>09:10:00</stop>
```

Figure 2: Simple example log-me configuration

#### 7.1.1.3. Maintaining State

An entity that inserts a log-me marker in a SIP request should ensure that a log-me marker is also inserted in responses to that request. An entity 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 it is missing. Entities that perform this log-me marking or checking must maintain a record of which dialogs are of interest to logging.

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 to for logging if end devices do not insert or echo a log-me marker.





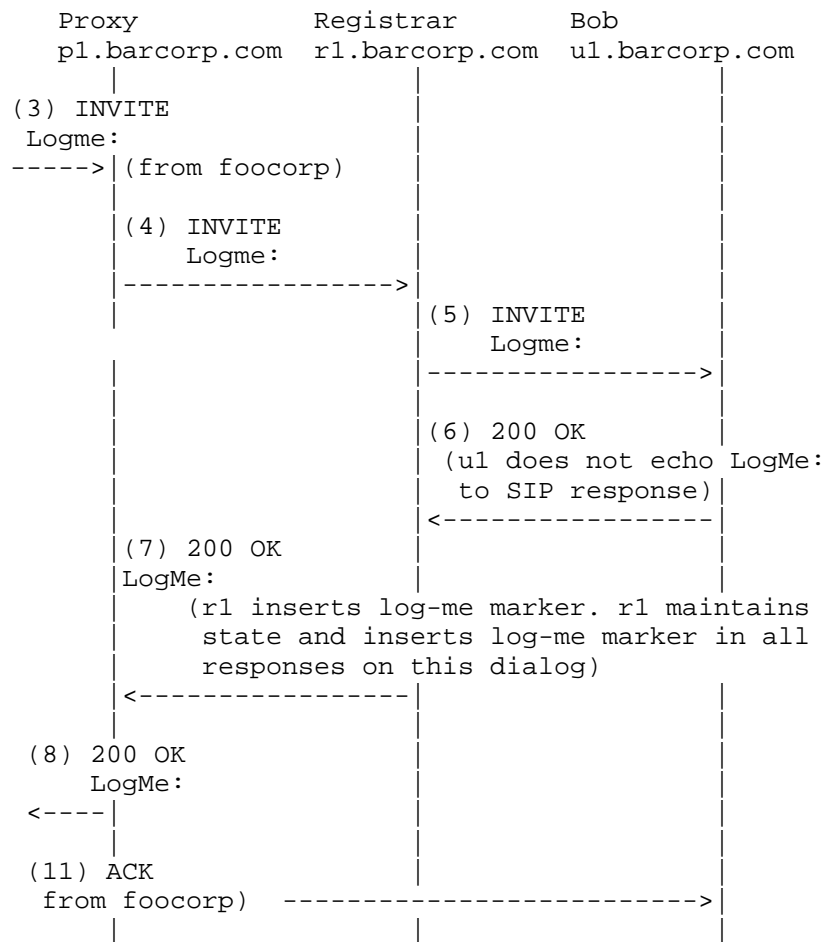
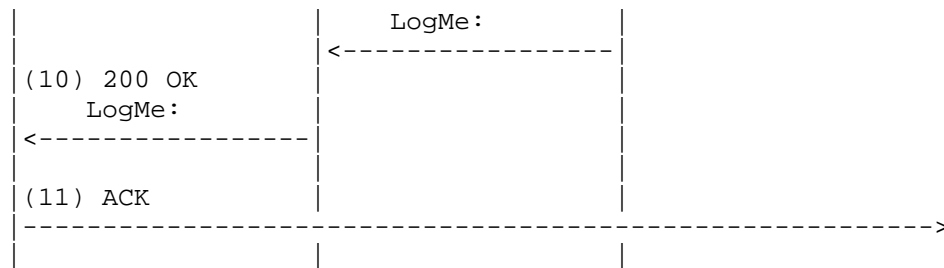


Figure 3: Maintaining state for log-me marking

### 7.1.2. Sending logs to a debug server

#### 7.1.2.1. 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.

#### 7.1.2.2. 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.

### 7.2. Solution A: LogMe header field

#### 7.2.1. Introduction

A new SIP header field, e.g. 'LogMe:', is defined to indicate that a session is of interest to logging. A supporting UA inserts the LogMe header field in an initial SIP request, and subsequent SIP requests belonging to the same dialog if any, and echoed in responses by a supporting UA that receives the SIP request. The LogMe header field has two header field parameters defined, one free-text name of the test case being performed, and one address of a server where collected logging will be sent after logging has terminated.

#### 7.2.2. Server Procedures

##### 7.2.2.1. General

A server may insert a logme marker, start logging, and stop logging.

A server may perform log-me marking. Typically, server marking is useful if a session is of interest to logging and the client did not perform log-me marking. If a server inserts a log-me marker, that server must maintain state in order to echo a log-me marker in SIP responses in a session of interest to logging.

##### 7.2.2.2. Sending logged information to a debug server

A server may send logged information to a debug server when a dialog has ended.

#### 7.2.3. Client Procedures

## 7.2.3.1. General

A client may perform any of the server procedures and may echo a log me marker from a request to a response.

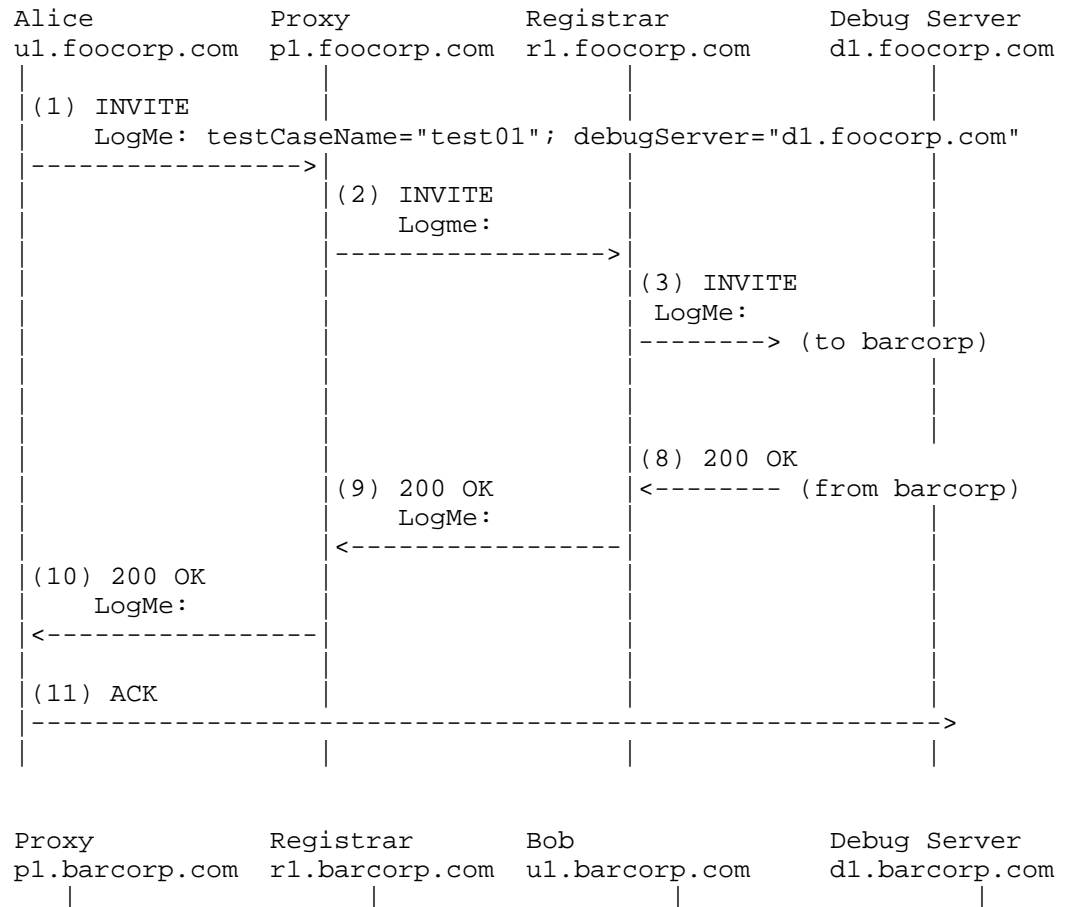
## 7.2.4. Identifying test cases

The Session-ID header field may be used to identify test cases such as particular regression tests.

## 7.2.5. Collecting logged information at a debug server

Clients and servers may send logged information to a debug server.

## 7.2.6. Examples



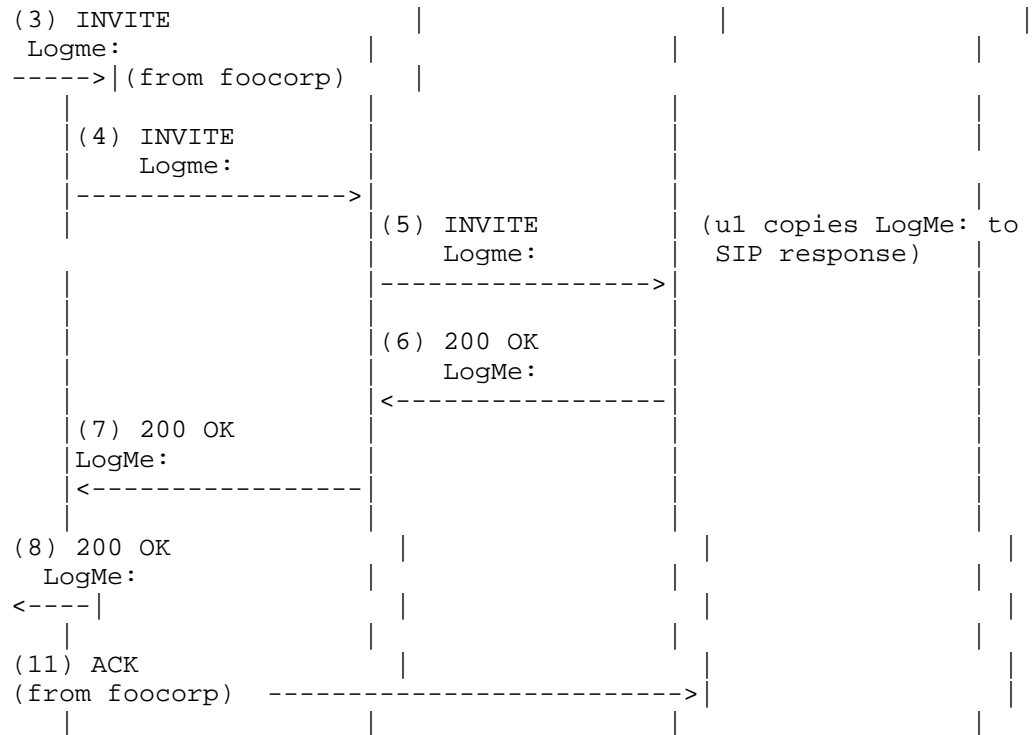


Figure 4: Signalling example for the LogMe header field solution

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

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

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

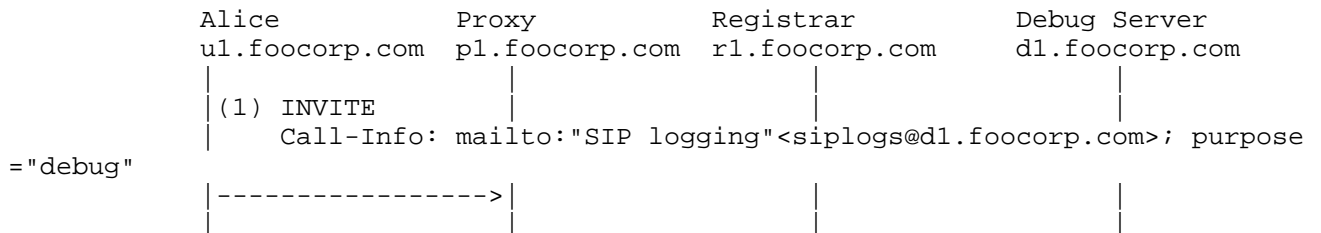


Figure 5: Signalling example for the Call-Info: purpose parameter solution

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.

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.

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

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]).

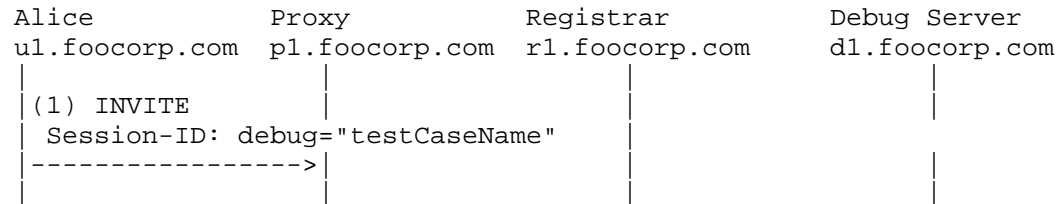


Figure 6: Signalling example for the Session-ID: header field parameter solution

#### 7.5. 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

## 8. References

### 8.1. Normative References

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

### 8.2. 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

Network Working Group  
Internet Draft  
Intended status: Standards Track  
Expires: January 15, 2014

P. Jones  
C. Pearce  
J. Polk  
G. Salgueiro  
Cisco Systems  
July 15, 2013

End-to-End Session Identification in IP-Based Multimedia  
Communication Networks  
draft-ietf-insipid-session-id-02

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.

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 January 15, 2014.

Copyright Notice

Copyright (c) 2013 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. Conventions used in this document.....	3
3. Session Identifier Requirements and Use Cases.....	3
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.....	5
6. Endpoint Behavior.....	6
7. Processing by Intermediaries.....	8
8. Associating Endpoints in a Multipoint Conference.....	9
9. Various Call Flow Operations Utilizing the Session ID.....	9
9.1. Basic Session ID Construction with 2 UUIDs.....	9
9.2. Basic Call Transfer using REFER.....	10
9.3. Basic Call Transfer using reINVITE.....	12
9.4. Single Focus Conferencing.....	13
9.5. Single Focus Conferencing using WebEx.....	15
9.6. Cascading Conference Bridge Support for the Session ID...	16
9.7. Basic 3PCC for two UAs.....	17
9.8. Session ID Handling in 100 Trying SIP Response and CANCEL Request.....	18
9.8.1. Session ID Handling in a 100 Trying SIP Response....	18
9.8.2. Session ID in a CANCEL SIP Request.....	19
9.9. Session ID in an out-of-dialog REFER Transaction.....	20
10. Compatibility with a Previous Implementation.....	21
11. Security Considerations.....	22
12. IANA Considerations.....	23
12.1. Registration of the "Session-ID" Header Field.....	23
12.2. Registration of the "remote" Parameter.....	23
13. Acknowledgments.....	23
14. References.....	23
14.1. Normative References.....	23
14.2. Informative References.....	24
Author's Addresses.....	25

## 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.

Unfortunately, there are a number of factors that contribute to the fact that 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 between the two protocols. This important fact makes it impossible for call identifiers to be exchanged end-to-end when a network utilizes one or more session protocols.

Another reason why the current call identifiers are not suitable to identify the 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 sessions 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 [RFC5234], shown in call flows in section 10. Additionally, this proposal attempts to account for a previous, proprietary version of a SIP Session ID header, proposing a backwards compatibility of sorts, described 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 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 can be found in a separate memo titled "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks" [I-D.ietf-insipid-session-id-reqts].

## 4. Constructing and Conveying the Session Identifier

### 4.1. Constructing the Session Identifier

The Session Identifier is comprised of two RFC 4122 defined 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 always generated with the same values. 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 will create a UUID and transmit that to the ultimate destination UA. Likewise, the responding UA 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 from the intended

recipient of the message, i.e., not an intermediary server along the signaling path. The set {A,B} is equal to the set {B,A}, and thus both represent the same Session Identifier.

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}, where "A" represents the single UUID value transmitted.

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 from 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. This merged call needs to be identified and identification of such sessions is natural and easily traceable when utilizing UUID values assigned by each entity in the communication session.

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. This might allow for correlation among the participants in a single conference, for example.

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 local UA-generated UUID associated with the session. This value MUST remain unchanged throughout the duration of that session.

A SIP UA MUST convey its Session Identifier UUID in all transmitted messages within the same session. To do this, each transmitted message MUST include 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
remote-uuid     = sess-uuid

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
```

The productions "SEMI", "EQUAL", and "generic-param" are defined in RFC 3261. The production DIGIT is defined in RFC 5234.

The Session-ID header MUST NOT have more than one "remote" parameter.

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 byte of the UUID appearing first.

A UUID having the value of all zeros is a special UUID value. It is used in certain special cases, and hereafter is defined as the "null" UUID value. Either the "local-uuid" field or "remote-uuid" field can have a "null" value.

## 6. Endpoint Behavior

To comply with this specification, 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 with one exception, mentioned below, and the UUID of the peer device, if known, must appear as the "remote" parameter following the transmitter's UUID.

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 when a SIP dialog is replaced via an INVITE with Replaces [RFC3891].

The exception to including the UUID of the transmitting entity mentioned above is in the case of provisional responses that occur before the destination UA has generated its UUID. The 100 (Trying) response and the 181 (Call Forwarding) response are examples of such provisional responses. In these cases, the sending intermediary places the one known UUID in the remote-uuid field, and leaves the "local-uuid" blank. This placement is always where a UA expects to receive its UUID value in SIP responses.

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" parameter that does not match what the UA expected the far end UA's UUID to be. This 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.

For any purpose the UA has for the Session Identifier, it MUST assume that the Session Identifier is {A,B} where "A" is the UUID value of this endpoint (i.e., "local-uuid") and "B" is the UUID value of the peer endpoint (i.e., "remote-uuid"), taken from the most recently received message within this session. Note that when comparing Session Identifiers for equivalence, the identifier {A,B} is equal to the set {B,A}.

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

It is also important to note that if a session is forked by an intermediary in the network, the initiating UA may receive multiple responses back from different endpoints, each of which will contain a different UUID ("local-uuid") value in each response received by this initiating UA. UAs MUST take care to ensure that the correct UUID value is returned in the "remote" parameter when responding to those endpoints.

Cascading MCUs all utilize the same UUID value ("local-uuid" portion of the Session-ID header-value) for all participants of the cascaded conference. An MCU conveys the UUID value to utilize via the "local-uuid" portion of the Session-ID header-value in an INVITE to a second MCU.

## 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 a SIP message having no Session-ID header is received by an intermediary, 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, they **MUST** be consistent with the text in Section 6 that discussed how endpoints are expected to receive a certain UUID value (say, either a "local-uuid" or "remote-uuid", but not both). In provisional responses that have not reached their destination UAs to generate the other UUID value for that endpoint, intermediaries are to place the UAC's UUID value in the "remote-uuid" portion of the Session-ID header-value, and a null "local-uuid" value.

A CANCEL request sent by an intermediary that has received no previous response from the target UA has a Session-ID constructed exactly like the INVITE to that UA, with only a "local-uuid" value in the Session-ID header-value.

## 8. Associating Endpoints in a Multipoint Conference

Multipoint Control Units (MCUs) group two or more sessions into a single multipoint conference. The MCU should utilize the same UUID value for each session that is grouped into the same 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.

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.

Note, however, that this assumption of being part of the same conference is not always true. For example, in a SIP forking scenario, there might also be what appears to be multiple sessions with a shared UUID value. This is actually desirable. What is desired is to allow for the association of related sessions. Whether sessions are related because of forking or because endpoints are communicating as a part of a conference does not matter. They are nonetheless related.

## 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].

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

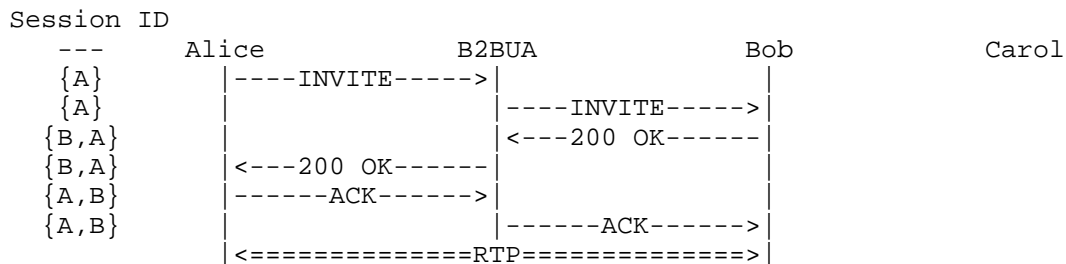


Figure 1 - Session ID Creation when Alice calls Bob

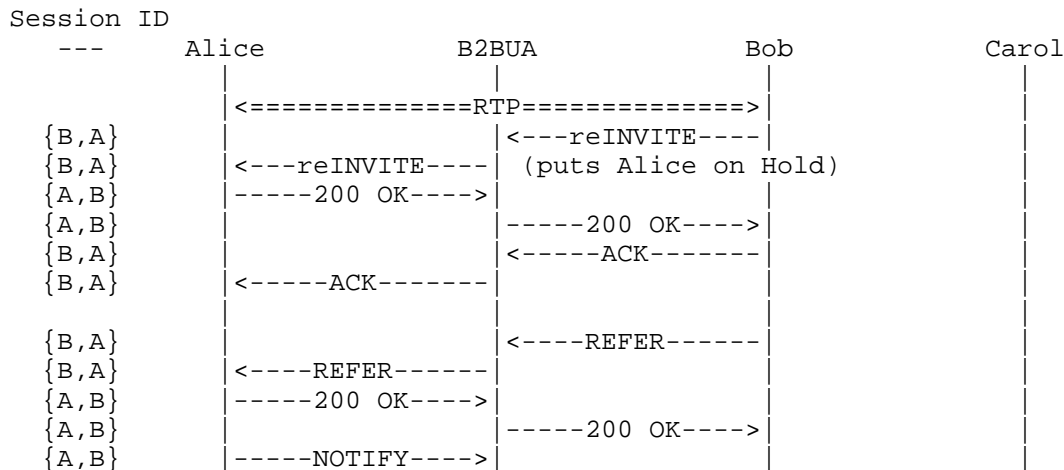
General operation of this example:



- o The originating transmitter of a SIP message populates the "local-uuid" portion of the Session-ID header-value.
- o UA-Alice sends its UUID in the SIP INVITE.
- o B2BUA receives an INVITE with a "local-uuid" portion of the Session-ID header-value from UA-Alice, and transmits INVITE towards UA-Bob with an unchanged Session-ID header-value.
- o UA-Bob receives Session-ID and adds 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.



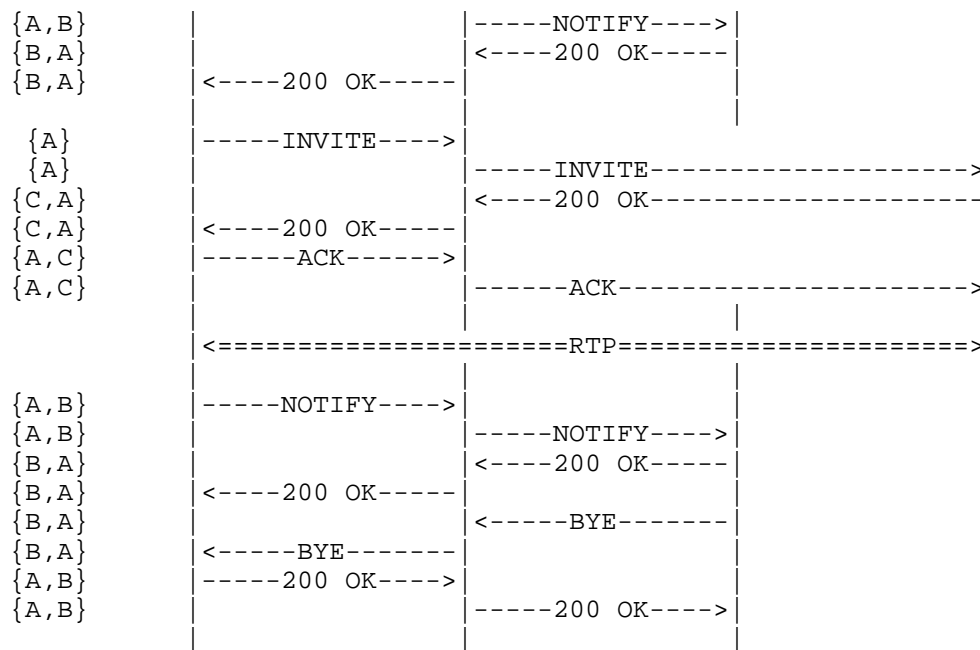


Figure 2 - Call Transfer using REFER

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 reINVITES 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, with no "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}, moves this 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. 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

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).

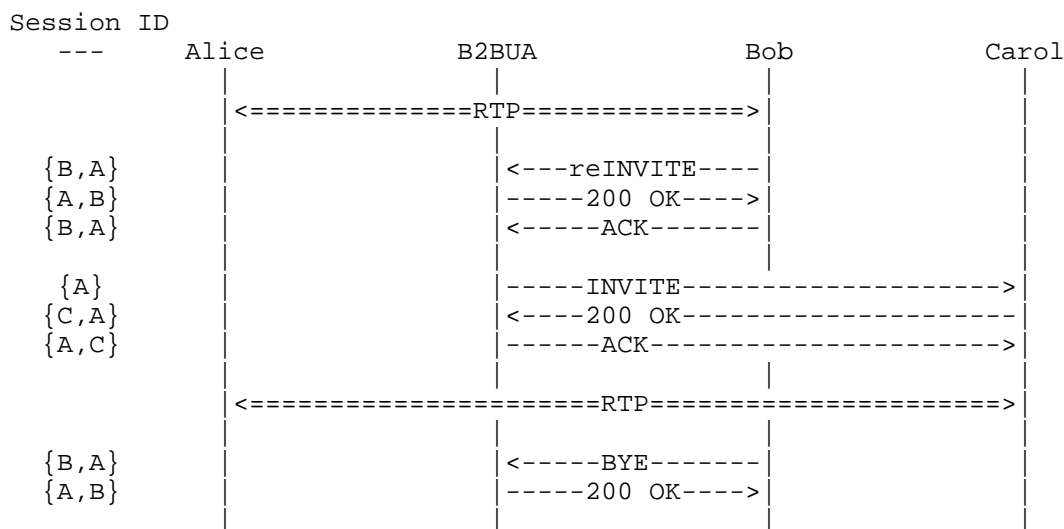


Figure 3 - Call transfer using reINVITE

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 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. Each user in this example goes through a two-step process of signaling to gain entry onto their conference call.

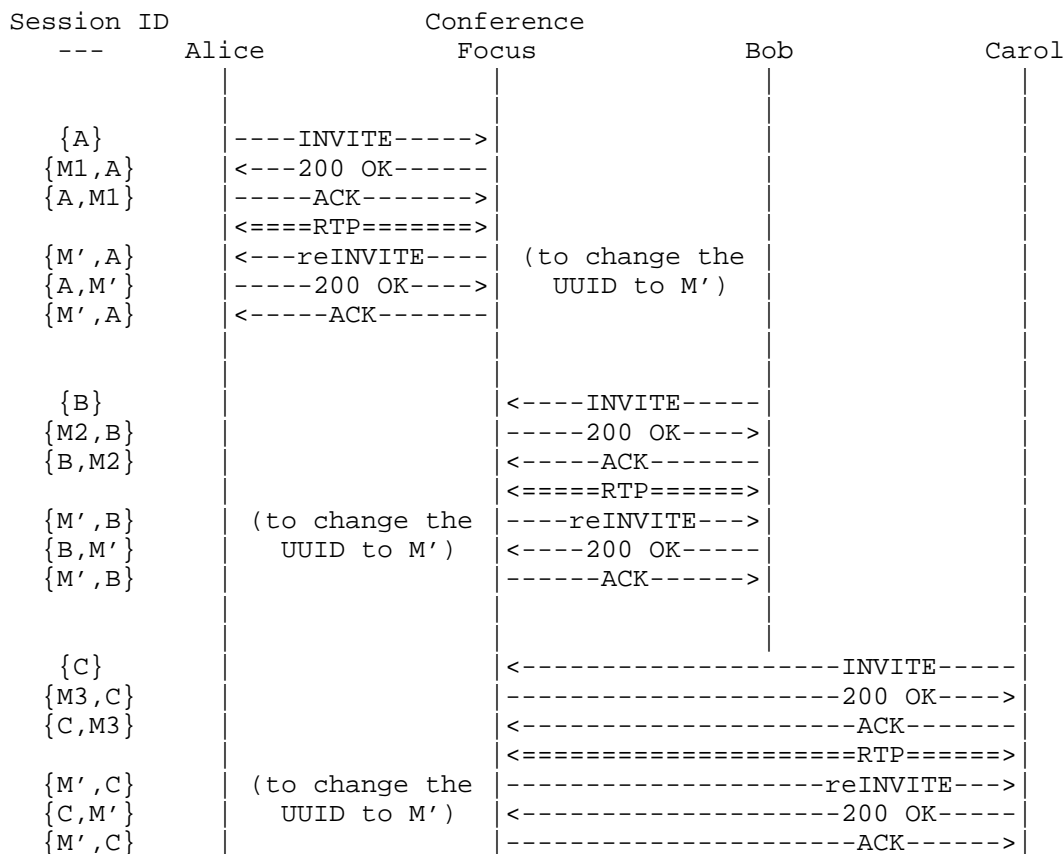


Figure 4 - Single Focus Conference Bridge

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}.
- o The conference server responds with a 200 OK response which includes 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.

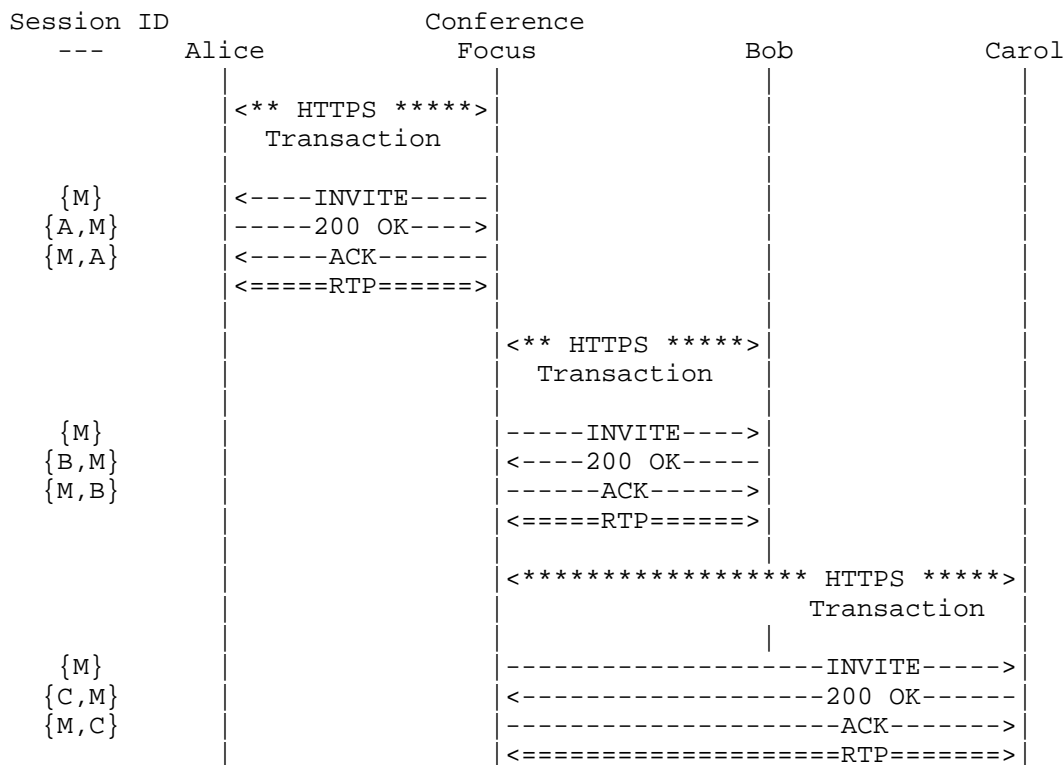


Figure 5 - Single Focus Webex Conference

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, where M equals the "local-uuid" for each participant on this conference bridge) to UA-Alice to start session with the of 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.

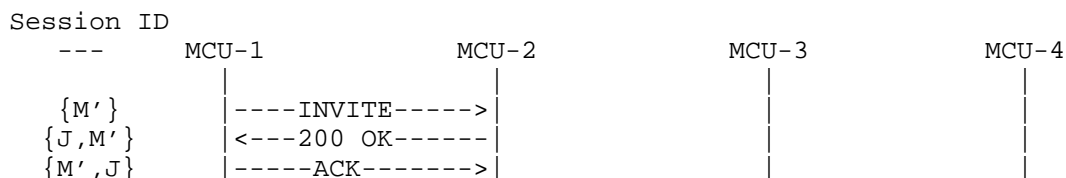


Figure 6 - MCUs Communicating Session ID UUID for Bridge

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' UUID 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

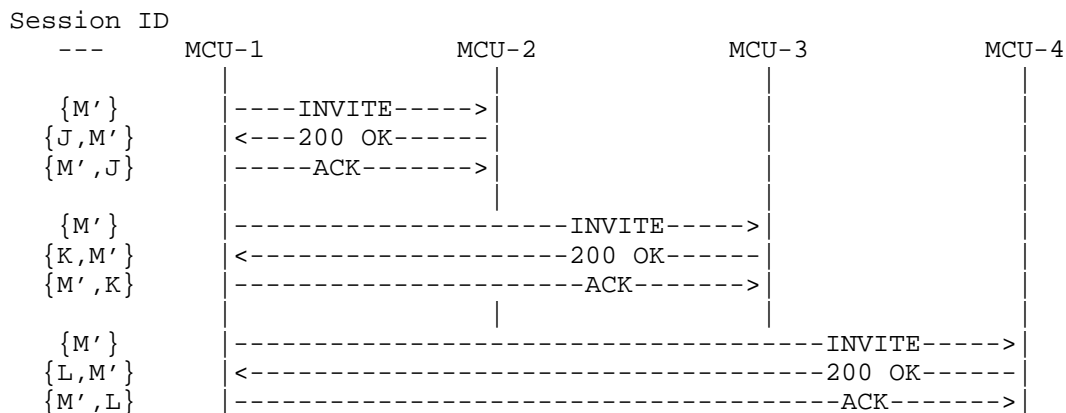


Figure 7 - MCU Communicating Session ID UUID to More than One

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 The MCU that is contacted, i.e., the UAS MCU, does not populate or complete the Session-ID header value. The UAS MCU transmits a 200 OK response acknowledging it is to respond with this M' UUID to all requests for the designated conference.
- o An MCU that receives this M' UUID 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.

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

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

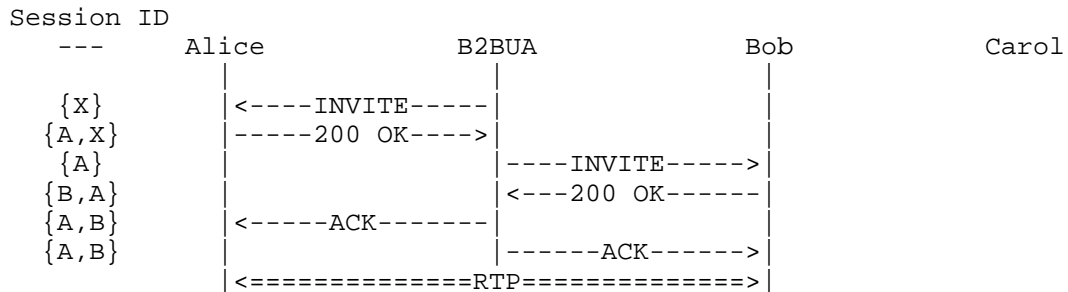


Figure 8 - 3PCC initiated call between Alice and Bob

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.
- o The SIP server INVITEs Alice to a session and uses a temporary UUID {X}.
- o Alice receives and accepts this call set-up and includes her UUID {A} in the Session ID, now {A,X}.
- o The SIP server uses Alice's UUID {A}, and discards its own {X} to INVITE Bob to the session 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").

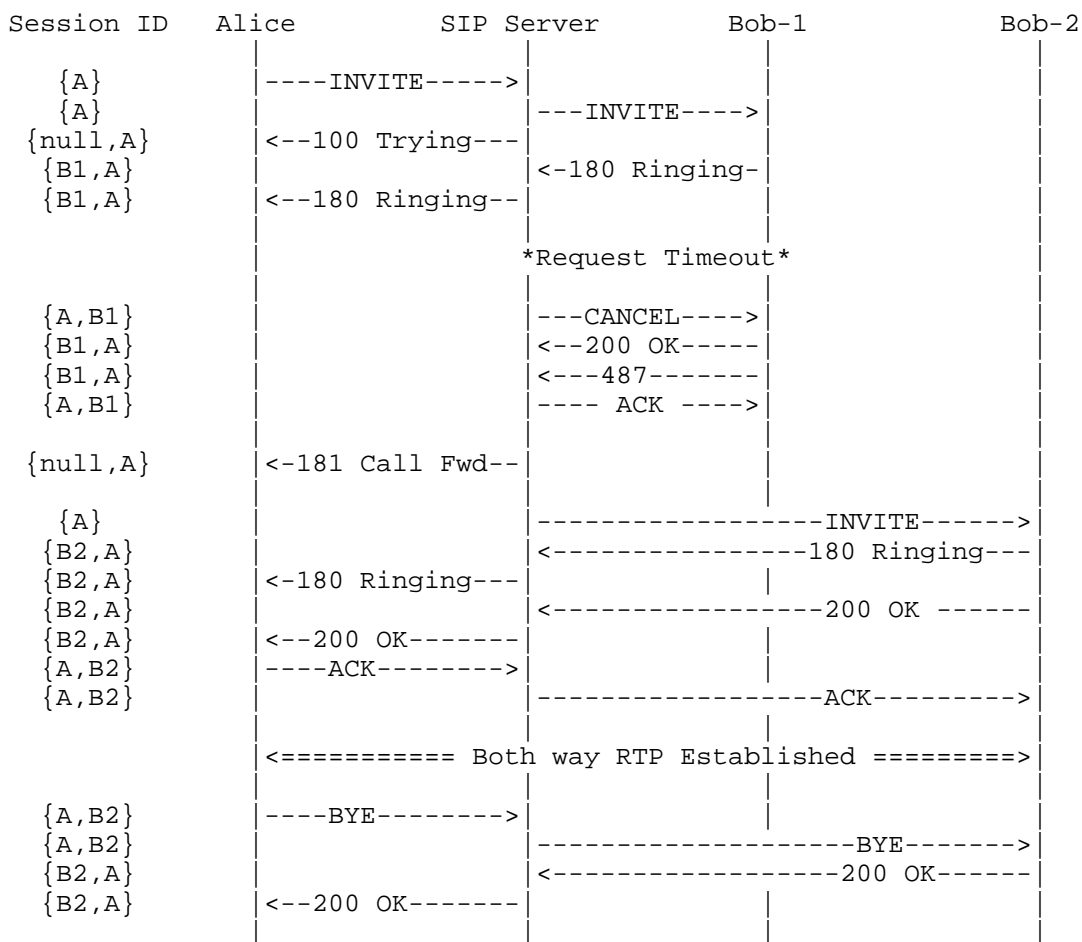


Figure 9 - Session ID in the 100 Trying and CANCEL Messaging

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.
- 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 UUID. Rather, the 100 Trying response only 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; 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 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

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

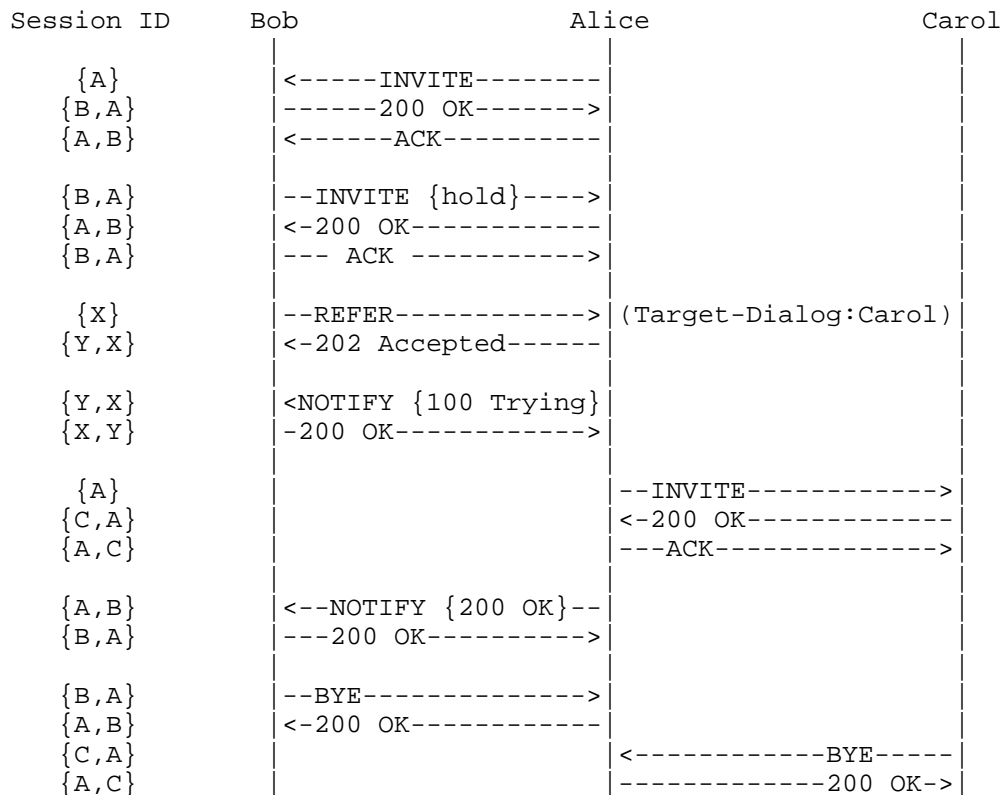


Figure 10: Basic Transfer Call Flow

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.
- o Alice will use her existing UUID "A" 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 that we will herewith attempt to achieve backwards compatibility. Neither Session-ID 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 following this document (to the letter) and everything should be just fine.

In order to have an "old" version talk to a "new" version implementation, several aspects need to be looked at. They are:

- o The "old" version UA will include a single UUID as its Session-ID.
- o The "new" version UA will respond by including a complete Session-ID with two UUIDs, with the "new" version's UUID listed first (because it cannot know it is talking with an "old" version implementation at this point).

- o The "old" version UA will have to ignore the first UUID, and consider its singular "old" UUID as valid, as long as the value does not change.
- o During subsequent transactions within this session, the "new" version may receive SIP requests without its UUID, but with the "old" version's UUID. The "new" version UA MUST add its UUID to the received Session-ID. The "old" version implementation will merely disregard it each time it receives this "new" version UUID (if it was not the first UUID).

In order to have a "new" version talk to an "old" Version implementation, several aspects need to be looked at. They are:

- o The "new" version UA will include a single UUID as its initial Session-ID header always, not knowing which version of UA it is communicating with.
- o The "old" version UA will respond by seeing the UUID as a valid and complete Session-ID and not include another UUID or generic-param. Thus, the 200 OK will not include any Session-ID part of its own from the "old" version implementation.

Rule: implementation supporting a "new" version of the Session-ID MUST NOT error or otherwise reject receiving only its own UUID back in any transaction. It MUST interpret this response to mean that it is communicating with an "old" Session-ID implementation.

- o Open question - how do we want all intermediaries and/or monitoring systems to interpret this single UUID complete Session-ID?

## 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.

Other considerations???

## 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, 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.

#### 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-07, June 2013.
- [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  
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  
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 WG  
Internet-Draft  
Intended status: PS  
Expires: April 21, 2014

James Polk  
Paul Jones  
Cisco Systems  
Oct 21, 2013

A Proposal for Backwards Compatibility of the INSIPID Session-ID  
draft-polk-insipid-bkwsd-compatibility-proposal-03.txt

Abstract

This is a proposal for backwards compatibility for the INSIPID Session-ID SIP header.

This draft is never intended to reach RFC status, and is written merely to make a concrete proposal that everyone can view - and if agreed upon, it will be woven into the INSIPID solution draft.

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 April 21, 2014.

Copyright Notice

Copyright (c) 2013 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.

## 1. Introduction

This is a proposal for backwards compatibility for the INSIPID Session-ID SIP header.

We, in INSIPID, have the 'kaplan draft' specifying a legacy solution [kaplan]. We have the 'jones draft' proposing the current solution [Insipid], which addresses a wider set of requirements than the kaplan draft. Let us call this the 'polk proposal' for backwards compatibility, given that this topic has become such an issue - let's give it a name.

The current way in which the jones draft specifies the Session-ID message (or call) flow is articulated here:

Alice invites Bob to a call. She includes her UUID, but not his, into the Session-ID header-value. Bob's UUID is inserted by his UA (or call server acting on behalf of his UA).

```
INVITE
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823;
```

Bob receives Alice's INVITE and responses (in a perfect world right away because everything in Alice's INVITE is perfectly acceptable to Bob) with a 200 OK response. This response contains Bob's UUID. Alice's UUID has been copied and moved to the 'remote=' portion of the Session-ID header-value.

```
200 OK
from Bob to Alice
Session-ID: bellaafc8b22911df86c412313a006823;
          remote=aeffa652b22911dfa81f12313a006823
```

Once Alice receives this 200 OK from Bob, she transmits the appropriate ACK. The only difference in this Session-ID is the order of the UUIDs.

```
ACK
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823;
          remote=bellaafc8b22911df86c412313a006823
```

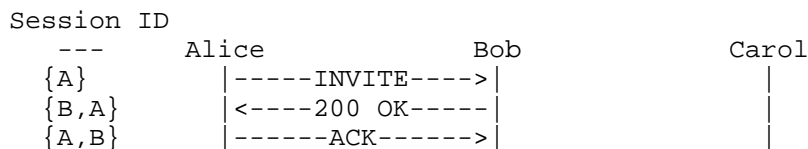


Figure 1. Basic Call Flow Without Proposed Session-ID Modification

This is all fine and good when conversing between jones draft implementations. In fact, all is fine and good between kaplan draft implementations (that all have a single UUID in the Session-ID header-value. However, problems start happening if either Alice or Bob is a jones implementation and the other UA is a kaplan implementation, and it does not matter which is which.

This draft proposes how we can make a single-UUID implementation interoperate with a two-UUID implementation with the single-UUID implementation requiring zero new code.

This version includes a couple of suggested alternative means for indicating which implementation an endpoint is compatible with. These suggestions are each outside of the Session-ID header, and each have their own failings.

This draft is never intended to reach RFC status, and is written merely to make a concrete proposal that everyone can view and reference in discussions.

## 2. Terminology

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

## 3. Session-ID Backwards Compatibility Proposal

This proposal involves having all jones implementations send two UUIDs in all their messages, including the initial INVITE. Alice will generate her UUID value the same as she does in the current jones draft. However, she will include a second temporary 'null' UUID in the remote= part of the Session-ID header-value, just as the following example shows:

```
INVITE
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823;
           remote=00000000000000000000000000000000
```

A 'null' UUID is 32 zeros, and MUST NOT be anything other than 32 zeros in length, i.e., not 'remote=' or 'remote=0'. This has huge benefits for 2 different implementations (a jones and a kaplan implementation). Not only is Alice indicating that she is a jones compatible version to all entities along the signaling path, but if she gets back just her UUID in Bob's 200 OK response or her UUID plus the temporary null value in the 'remote=' in the same order she sent it in, she knows that Bob is a kaplan implementation - because kaplan implementations will reflect the initial UUID as the complete

Session-ID and remove any other values in the header.

```
200 OK
from Bob to Alice
Session-ID: aeffa652b22911dfa81f12313a006823
```

Or

```
200 OK
from Bob to Alice
Session-ID: aeffa652b22911dfa81f12313a006823;
remote=00000000000000000000000000000000
```

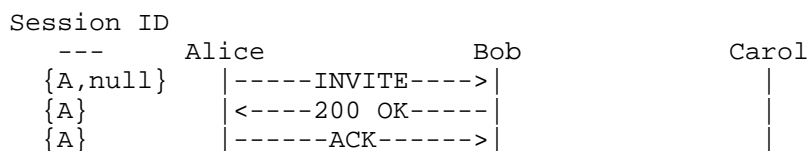


Figure 2a. Basic Call Flow With Proposed Session-ID Modification

Or

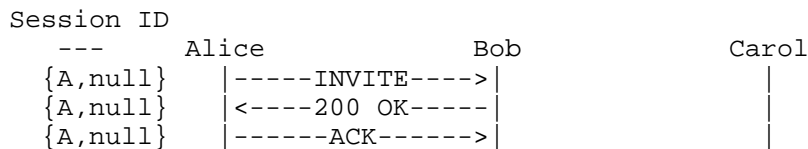


Figure 2a. Basic Call Flow With Proposed Session-ID Modification

If Alice receives two UUIDs back in Bob's response, such as this

```
200 OK
from Bob to Alice
Session-ID: bellaafc8b22911df86c412313a006823;
remote=aeffa652b22911dfa81f12313a006823
```

Alice knows Bob has a jones implementation.

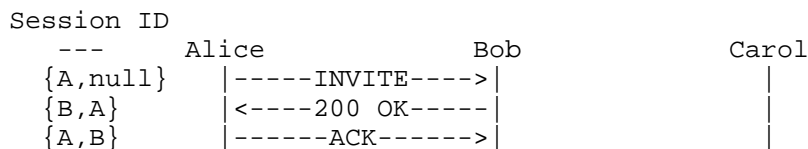


Figure 3. Basic Call Flow With Proposed Session-ID Modification

In either case, she sends two UUIDs back in her response, to ensure she is always broadcasting which implementation she is compatible with.

Either this when Bob is a kaplan implementation - learned by Bob not sending a second UUID in his response.

```
ACK
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823
```

or this, when Bob included his UUID in the response.

```
ACK
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823;
           remote=bellaafc8b22911df86c412313a006823
```

Alternatively, we could modify this proposal to have Alice (a jones implementation), and the (any) initiator of the SIP request, always send

```
Session-ID: aeffa652b22911dfa81f12313a006823;
           remote=00000000000000000000000000000000
```

because this appears to do no harm to a kaplan implementation

because, first, the UA will ignore the second UUID, and second, it will not include this second UUID in any response. It is just a question of how closely kaplan implementations follow the kaplan draft, which states that all values in the Session-ID header-value are merely copied into the response. We have heard about inconsistencies as to how this is in fact implemented.

Session ID	Alice	Bob	Carol
{A,null}	-----INVITE----->		
{A}	<-----200 OK-----		
{A,null}	-----ACK----->		

Figure 4. Basic Call Flow With Mod. to Proposed Session-ID Modification

This will affect all subsequent call flows, and I'm not opposed to doing this if the WG wants to go in this direction instead.

### 3.1 When Alice is the kaplan implementation and she initiates

If Alice is the kaplan implementation and she contacts a jones implementation, the same will be true as above, but let me walk through that example. Let us say Alice (a kaplan) invites Bob (a jones) to a session. It would look like this:

```

INVITE
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823

```

Alice would generate 'the' UUID for this session with Bob.

Bob's response (in a perfect world) would be this:

```

200 OK
from Bob to Alice
Session-ID: aeffa652b22911dfa81f12313a006823

```

Bob has learned Alice is a kaplan implementation, so he merely reflects Alice's UUID back to her in his response.

Below is Alice's ACK

```

ACK
from Alice to Bob
Session-ID: aeffa652b22911dfa81f12313a006823

```

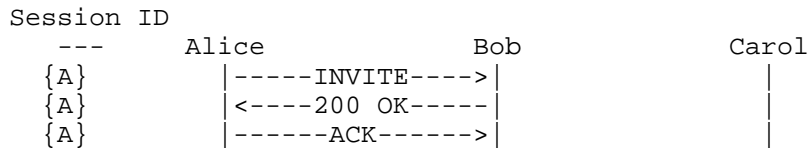


Figure 5. Basic Call Flow With Alice being a Kaplan Implementation And Bob being a Jones Implementation

### 3.2 Call Transfer

For the case where both Alice and Bob are jones implementations, the initial INVITE would have the sender's UUID and the 'remote=null' (32 zeros) Session-ID. The same is true for the INVITE request as a result of the REFER message - to Carol in most examples, including in the INSIPID solution draft [1].

In the next two sub-sections, Alice is a kaplan implementation, and Bob is a jones implementation.

#### 3.2.1 Transferee is a kaplan implementation

In this case, Alice and Bob are in a session, and Bob transfers Alice to Carol (a jones implementation) using a REFER.

Alice is the kaplan implementation and Bob is the jones implementation in this scenario. Since Alice only provides a single UUID in the initial INVITE, she is indicating to all signaling entities in the signaling path that she is a kaplan implementation.

Therefore there is only every one UUID, regardless of whether Carol is a kaplan implementation or a jones implementation.

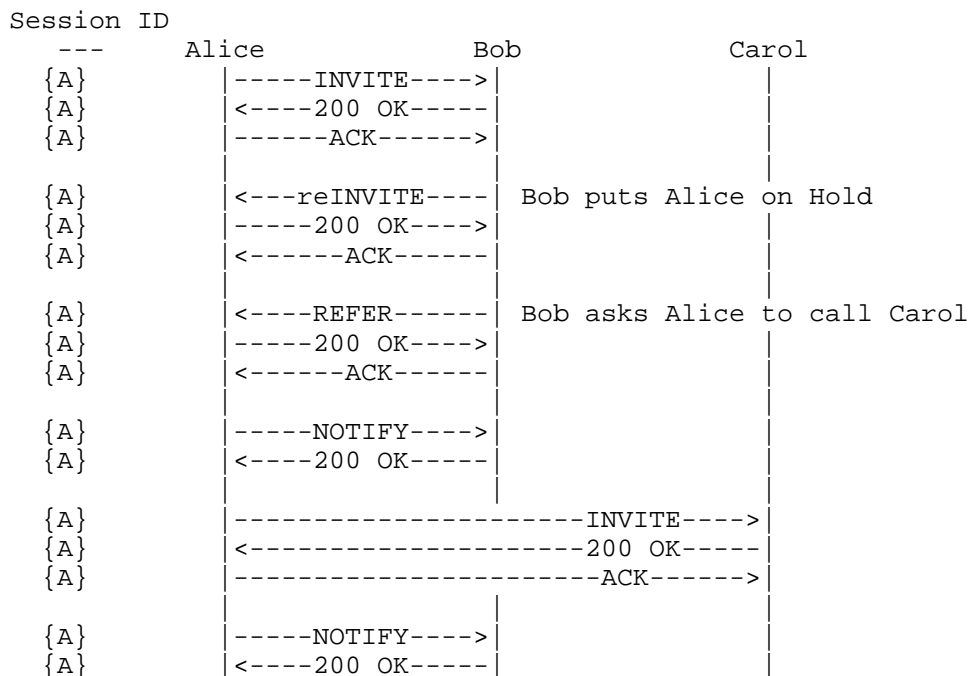


Figure 6. Call Transfer where Transferee is a kaplan implementation

### 3.2.2 Transferee is a jones implementation

In this case, Alice and Bob are in a session, and Alice transfers Bob to Carol (a jones implementation) using a REFER.

Alice is the kaplan implementation and Bob is the jones implementation in this scenario. Since Alice only provides a single UUID in the initial INVITE, she is indicating to all signaling entities in the signaling path that she is a kaplan implementation. Therefore there is only every one UUID, *\*except\** when Bob, a jones implementation, initiates a new SIP request to any entity other than Alice. We want any jones implementation to constantly be looking for other jones implementations so we have the exact same message flow *\*until\** Bob sends his INVITE to Carol.

Bob, a jones implementation, told to contact Carol - who he wants to determine if she is another jones implementation, includes a 'remote=null' value as the second UUID in the Session-ID header-value.

INVITE  
from Bob to Carol



```
Session-ID: aeffa652b22911dfa81f12313a006823;
remote=00000000000000000000000000000000
```

Carol sees this INVITE with 2 UUIDs where the second is a null value (in the Session-ID header-value), therefore she understands Bob to be a jones implementation, so she responds with this 200 OK, indicating that she, too, is a jones implementation. Bob's UUID has been copied and moved to the 'remote=' portion of the Session-ID header-value.

```
200 OK
from Carol to Bob
Session-ID: cf26afc8b22911df86c412313a006823;
remote=aeffa652b22911dfa81f12313a006823
```

Once Bob receives this 200 OK from Carol, he transmits the appropriate ACK. The only difference in this Session-ID is the order of the UUIDs.

```
ACK
from Bob to Carol
Session-ID: aeffa652b22911dfa81f12313a006823;
remote=cf26afc8b22911df86c412313a006823
```

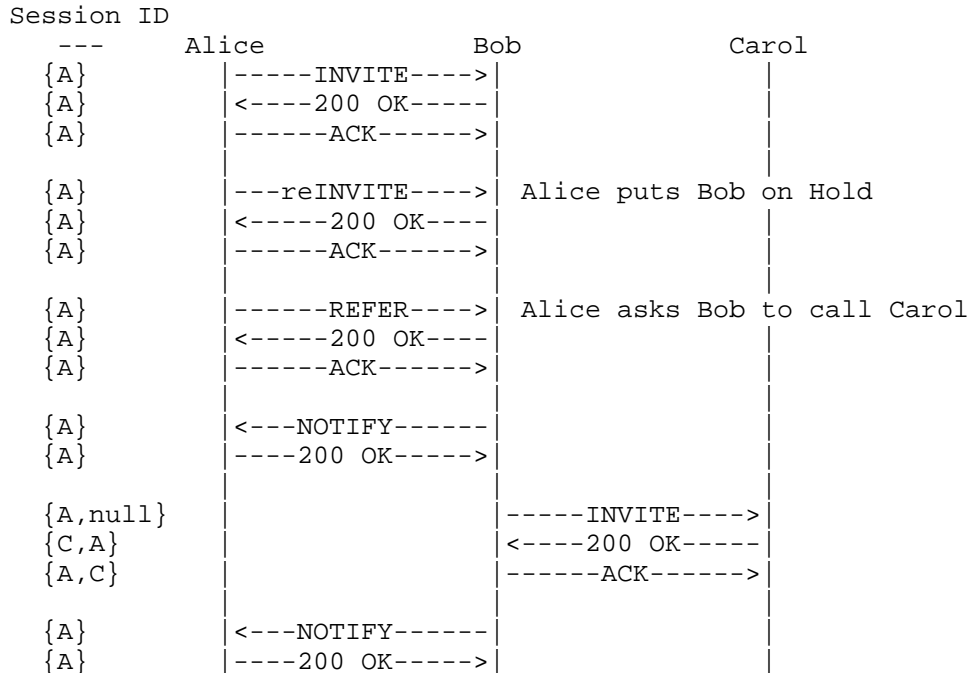


Figure 7. Call Transfer where Transferee is a jones implementation

#### 4. Any More Call Flows Necessary?

I hope readers familiar with the INSIPID work understand this proposal by now, but I will do more call flows if needed.

#### 5. Flags Outside of the Session-ID to Indicate Compliance with a Particular Implementation

There have been to date two alternatives discussed on the mailing list regarding a flag to identify which implementation an endpoint is; namely a new option tag and a Contact header feature tag. Both of these will not work IMO for the following reasons:

A new Option tag

- Option tags are indications of support (or not) for one or more particular SIP extensions. Kaplan draft endpoints support the Session-ID extension. Jones implementations will support the Session-ID extension. However, what could be a proper indication for 'which' Session-ID implementation an endpoint is compliant with that does not involve writing new code in kaplan endpoints, which is not a desired outcome (to force all kaplan implementations to write new code)? What does an endpoint indicate if it supports both kaplan and jones implementations?

A Contact header feature tag

- the Contact header is not part of a SIP MESSAGE method request, and I find no reason to limit this particular new capability to "not to be used with MESSAGE requests" (ever).

6.

#### 7. Acknowledgements

To Christer Holmberg, Keith Drage, Paul Kyzivat and Hadriel Kaplan for the helpful comments.

#### 8. IANA Considerations

null

#### 9. Security Considerations

null

## 10. References

### 10.1 Normative References

- [RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997
- [Inspid] P. Jones, C. Pearce, J. Polk, G. Salgueiro, "End-to-End Session Identification in IP-Based Multimedia Communication Networks", "work in progress",

### 10.2 Informative References

- [kaplan] H. Kaplan "A Session Identifier for the Session Initiation Protocol (SIP)", "work in progress", August 2013

## Author's Addresses

James Polk  
Cisco Systems, Inc.  
8017 Hallmark Dr.  
North Richland Hills, TX 76182  
USA

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

Paul E. Jones  
7025 Kit Creek Rd.  
Research Triangle Park, NC, USA  
+1 919 476 2048

mailto: paulej@packetizer.com