                      End-to-End Session Identification in
                   IP-Based Multimedia Communication Networks

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 the Session
   Initiation Protocol (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 is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   Internet Standards is available in Section 2 of RFC 7841.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc7989.

Copyright Notice

Table of Contents

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 the call flows in
   Section 10. Additionally, this 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
lowercase, 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 Media Access Control (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 the 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 "namespace 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 request 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 of this document).

Since SIP sessions are subject to any number of service interactions, SIP INVITE requests 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 request that gets forked to several different endpoints. A means of identifying each of these separate communication sessions is needed; 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; therefore, it
should 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 a
   session identifier as defined in [RFC7329], the "remote" parameter
   may be absent; 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 request 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 communication 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, a REFER that directs the endpoint to a different peer, or an INVITE request 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 it 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 request, 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 it 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 not to 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 that 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 a 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 initial
INVITE request 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 request 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
request 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, what UUID values the receiving
endpoint should have knowledge of 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 request. 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 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 it 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 that 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, or 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   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, or 6xx) response, the new value MUST NOT be
    used.

o   As stated in Sections 6 and 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 request.
    Upon receiving a CANCEL request, an endpoint or intermediary would
    normally send a 487 Request Terminated response (see
    Section 15.1.2 of [RFC3261]) which, by the rules outlined above,
    would result in the endpoint or intermediary not storing any UUID
    value contained in the CANCEL request. Section 3.8 of [RFC6141]
    specifies conditions where a CANCEL request can result in a 2xx
    response. Because a CANCEL request is not passed end-to-end and
    will always contain the UUID from the original INVITE request,
    retaining a new UUID value received in a CANCEL request 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 request and any subsequent
    messages MUST contain the previously stored UUID value in the

       "remote" parameter". Note that the response to the CANCEL request
       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 request. 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 be utilized for a conference via the "local-uuid" portion of
   the Session-ID header field value in an INVITE request 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 request to the other MCU(s) participating in that
   conference. Also included in this INVITE request is an "isfocus"
   Contact header field value parameter identifying that this INVITE
   request 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 that a single UUID is common across
all participating MCUs of the same conference, but that it 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-diagnostic equipment might assume
that when they see two or more sessions with different session
identifiers but with one UUID in common, 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
exist 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 examples of 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 represent the unique UUID values corresponding to endpoints
or MCUs.

10.1. Basic Call with Two 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=============>|
```
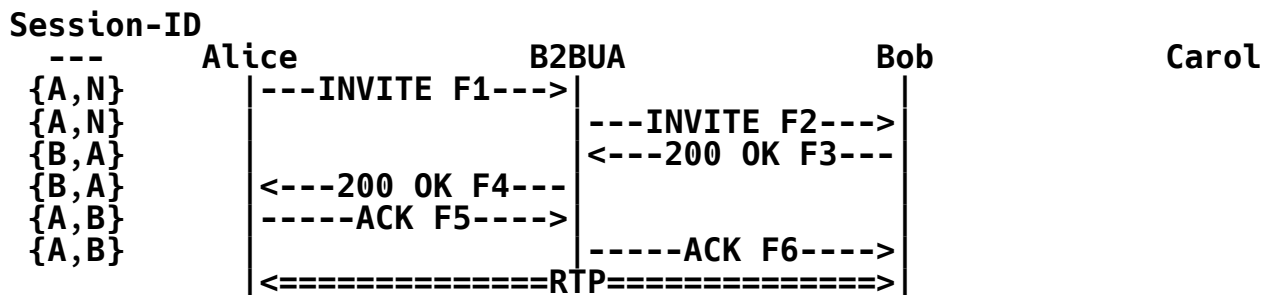
                 Figure 1: Session-ID Creation When Alice Calls Bob

   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 request and populates
      the "remote" parameter with a nil value (32 zeros).

   o The B2BUA receives an INVITE request 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 request
      towards UA-Bob with an unchanged Session-ID header field value.

   o UA-Bob receives the 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 UA-Alice's UUID unchanged to the "remote-uuid"
      portion of the Session-ID header field value in the 200 OK SIP
      response.

   o The B2BUA receives the 200 OK response with a complete Session-ID
      header field value from UA-Bob and transmits the 200 OK response
      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 UA-Alice's UUID is the "local-
      uuid", and UA-Bob's UUID populates the "remote-uuid" portion of
      the header-value.

   o The B2BUA receives the ACK with a complete Session-ID header field
      from UA-Alice and transmits the 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.
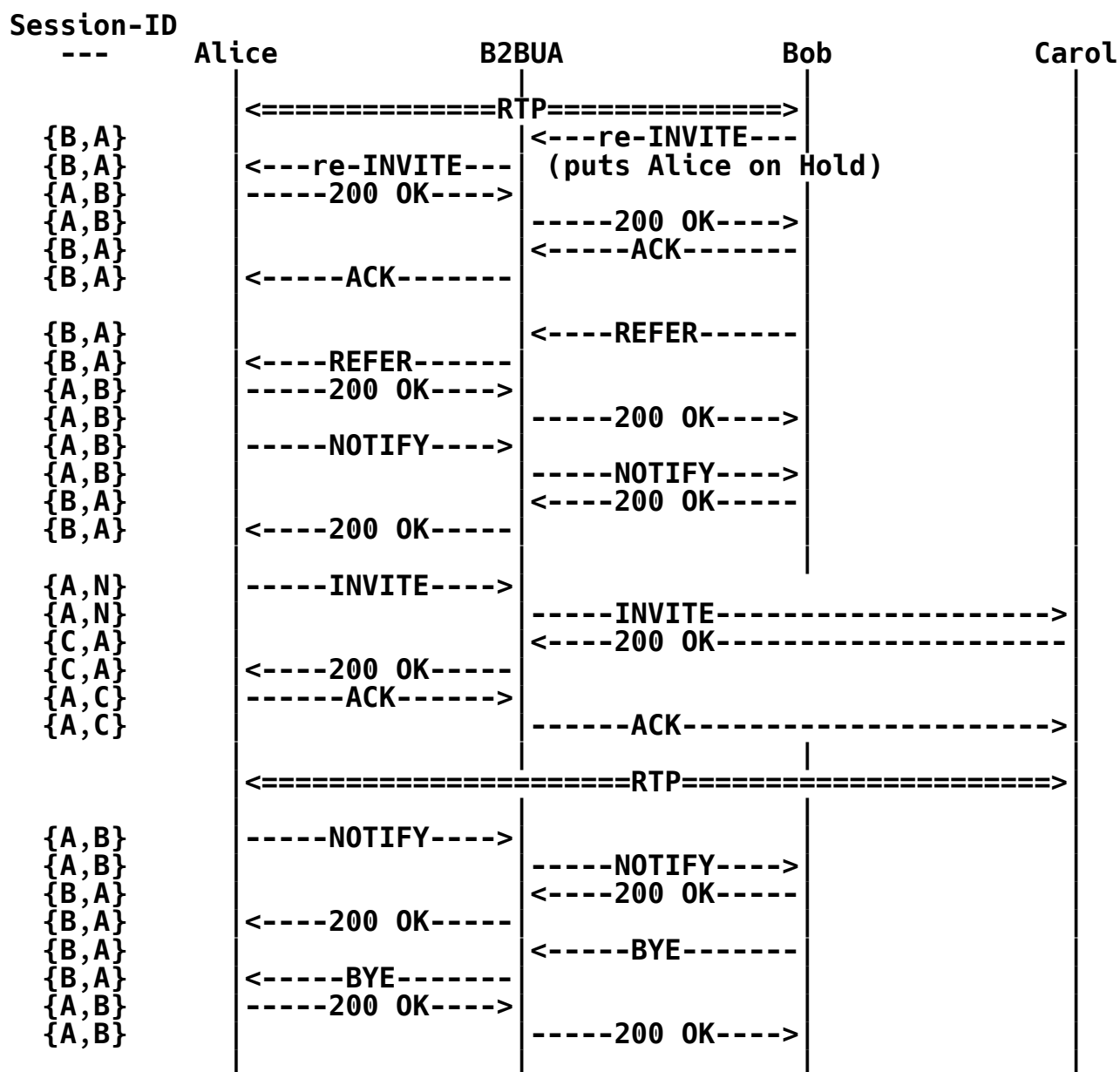
```
        Session-ID
           ---      Alice              B2BUA            Bob           Carol
                    |                  |                |                |
                    |<=============RTP==============>|                |
         {B,A}      |                  |<---re-INVITE---|                |
         {B,A}      |<---re-INVITE---  | (puts Alice on Hold)            |
         {A,B}      |-----200 OK---->  |                |                |
         {A,B}      |                  |-----200 OK---->|                |
         {B,A}      |                  |<-----ACK-------|                |
         {B,A}      |<-----ACK-------  |                |                |
                    |                  |                |                |
         {B,A}      |                  |<----REFER------|                |
         {B,A}      |<----REFER------  |                |                |
         {A,B}      |-----200 OK---->  |                |                |
         {A,B}      |                  |-----200 OK---->|                |
         {A,B}      |-----NOTIFY---->  |                |                |
         {A,B}      |                  |-----NOTIFY---->|                |
         {B,A}      |                  |<----200 OK-----|                |
         {B,A}      |<----200 OK-----  |                |                |
                    |                  |                |                |
         {A,N}      |-----INVITE---->  |                |                |
         {A,N}      |                  |-----INVITE-------------------->|
         {C,A}      |                  |<----200 OK---------------------|
         {C,A}      |<----200 OK-----  |                |                |
         {A,C}      |------ACK------>  |                |                |
         {A,C}      |                  |------ACK---------------------->|
                    |                  |                |                |
                    |<===================RTP======================>|
                    |                  |                |                |
         {A,B}      |-----NOTIFY---->  |                |                |
         {A,B}      |                  |-----NOTIFY---->|                |
         {B,A}      |                  |<----200 OK-----|                |
         {B,A}      |<----200 OK-----  |                |                |
         {B,A}      |                  |<-----BYE-------|                |
         {B,A}      |<-----BYE-------  |                |                |
         {A,B}      |-----200 OK---->  |                |                |
         {A,B}      |                  |-----200 OK---->|                |
                    |                  |                |                |

                    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 request 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 request 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, this
   switches the places of the two UUID fields), and generates a
   NOTIFY request to Bob with a session identifier {A,B} indicating
   that the call transfer was successful.

o It does not matter which UA terminates the Alice-to-Bob call;
   Figure 2 shows UA-Bob terminating the call.

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).

```
      Session-ID
          ---        Alice              B2BUA              Bob            Carol
                       |                  |                 |                |
                       |<=============RTP==============>|                |
                       |                  |                 |                |
                       |                  | <--- (non-standard signaling)  |
         {A,B}         |                  |---re-INVITE--->|                |
         {B,A}         |                  |<-----200 OK----|                |
         {A,B}         |                  |-----ACK------->|                |
                       |                  |                 |                |
         {A,N}         |                  |-----INVITE--------------------->|
         {C,A}         |                  |<----200 OK----------------------|
         {A,C}         |                  |------ACK----------------------->|
                       |                  |                 |                |
                       |<===================RTP======================>|
                       |                  |                 |                |
         {A,B}         |                  |------BYE------>|                |
         {B,A}         |                  |<----200 OK-----|                |
                       |                  |                 |                |
         {C,A}         |<--re-INVITE----|                 |                |
         {A,C}         |----200 OK----->|                 |                |
         {C,A}         |<-----ACK-------|                 |                |
                       |  (Suppose Alice modifies the session)           |
         {A,C}         |---re-INVITE--->|                 |                |
         {A,C}         |                  |---re-INVITE-------------------->|
         {C,A}         |                  |<---200 OK-----------------------|
         {C,A}         |<---200 OK------|                 |                |
         {A,C}         |------ACK------>|                 |                |
         {A,C}         |                  |------ACK----------------------->|
                       |                  |                 |                |
```
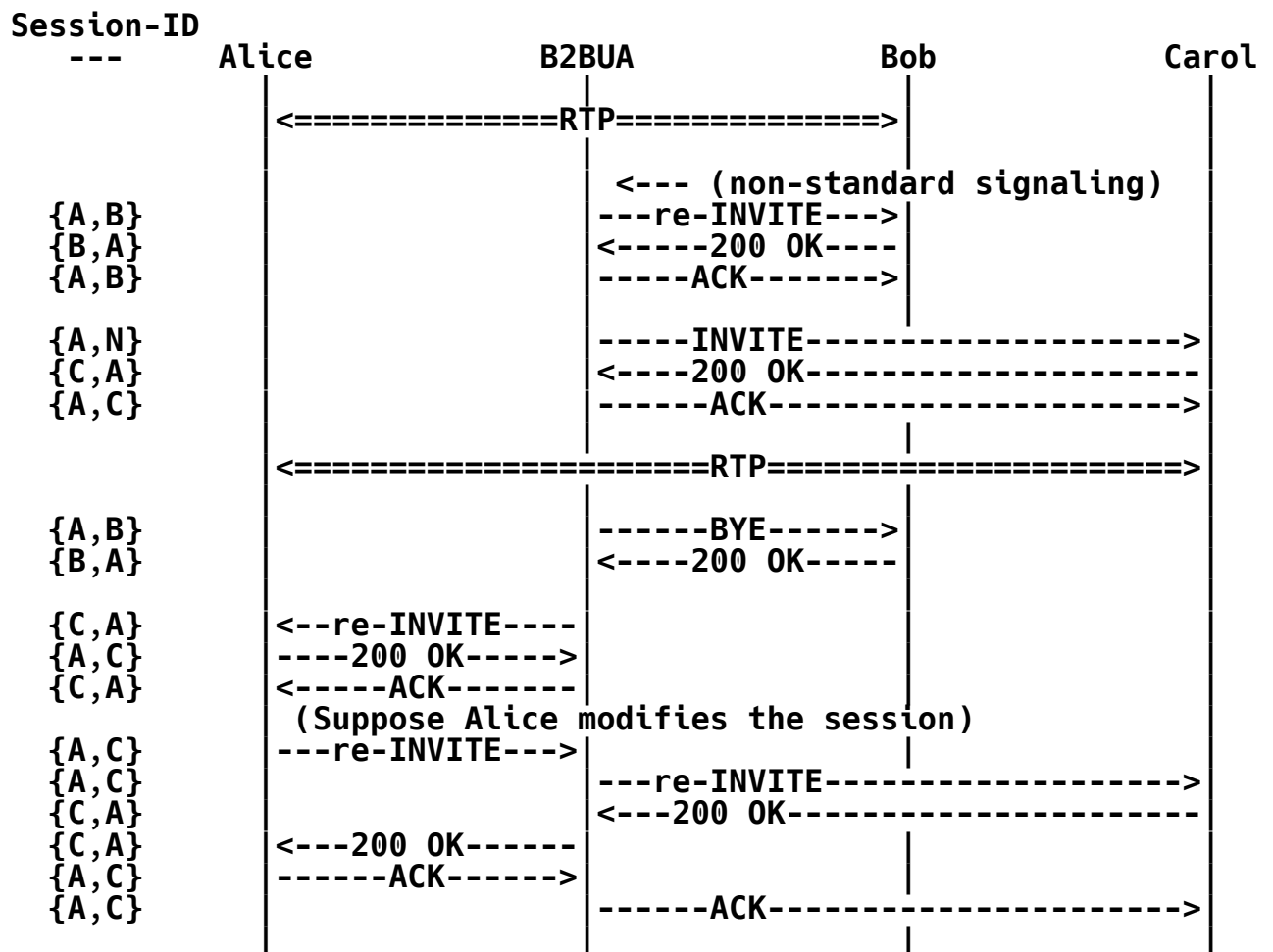
                    Figure 3: Call Transfer Using Re-INVITE

   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 request with the session identifier
      {"local-uuid" = "A", "remote-uuid" = "B"} to renegotiate the
      session with Bob.

   o The B2BUA sends a new INVITE request with Alice's UUID {"local-
      uuid" = "A"} to Carol.

   o Carol receives the INVITE request 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 request 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 request 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 request (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 (for example, 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
   Interactive Voice Response (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".

```
           Session-ID              Conference
              ---        Alice       Focus           Bob          Carol
                          |             |             |             |
           {A,N}          |----INVITE----->             |             |
           {M1,A}         |<---200 OK------             |             |
           {A,M1}         |-----ACK------->             |             |
                          |<====RTP=======>             |             |
           {M',A}         |<---re-INVITE---             |             |
           {A,M'}         |-----200 OK---->             |             |
           {M',A}         |<-----ACK-------             |             |
                          |             |             |             |
                          |             |             |             |
           {B,N}          |             |<----INVITE-----             |
           {M2,B}         |             |-----200 OK---->             |
           {B,M2}         |             |<-----ACK-------             |
                          |             |<=====RTP======>             |
           {M',B}         |             |---re-INVITE--->             |
           {B,M'}         |             |<----200 OK-----             |
           {M',B}         |             |------ACK------>             |
                          |             |             |             |
                          |             |             |             |
           {C,N}          |             |<--------------------INVITE-----
           {M3,C}         |             |---------------------200 OK---->
           {C,M3}         |             |<--------------------ACK-------
                          |             |<===================RTP======>
           {M',C}         |             |-------------------re-INVITE--->
           {C,M'}         |             |<--------------------200 OK-----
           {M',C}         |             |---------------------ACK------>|
```
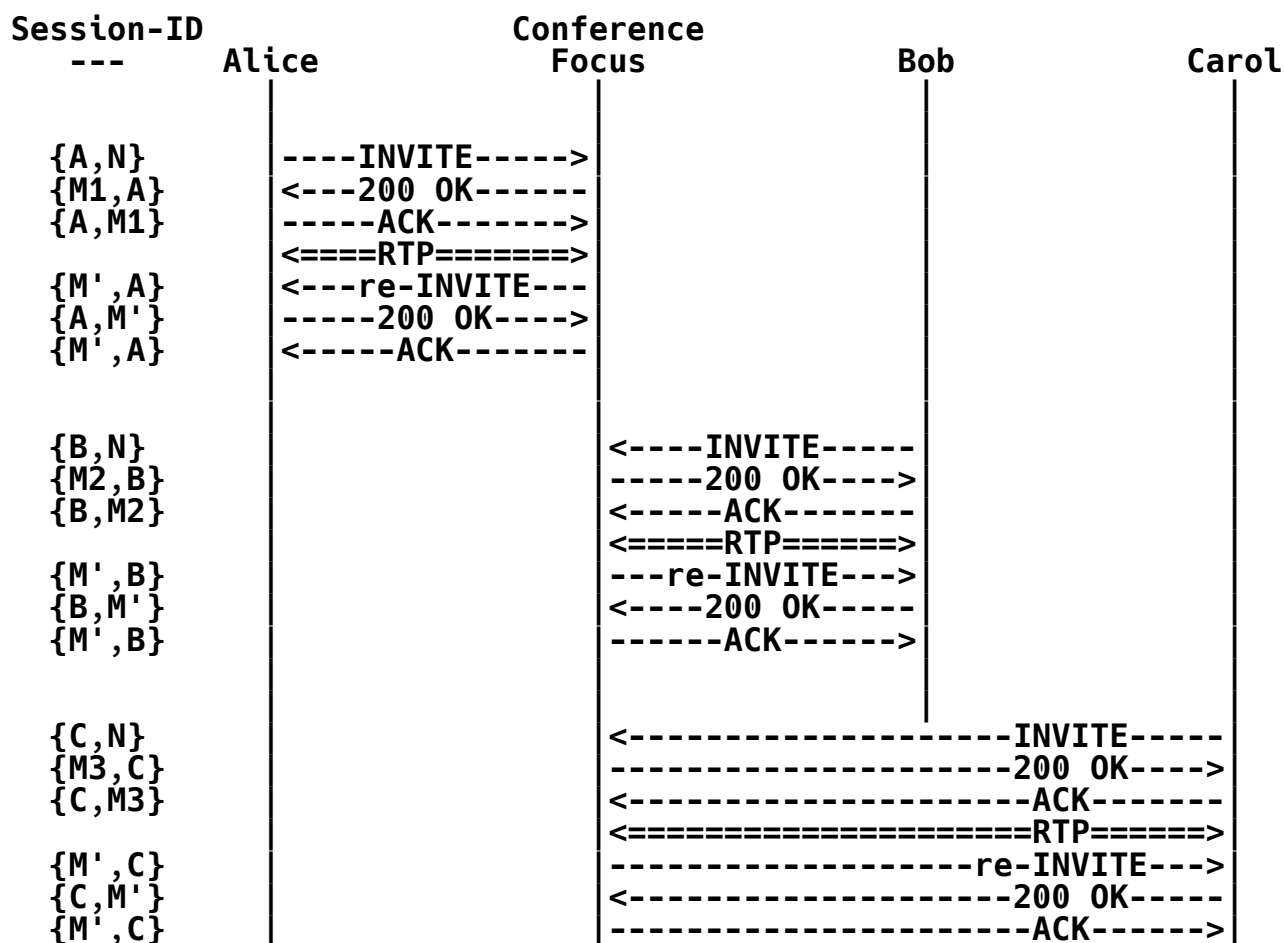
                  Figure 4: Single Focus Conference Bridge

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 request 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 sent in the 200 OK response.

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 the conference.

o   In this example, Carol (and each additional user) goes through the
    same procedures 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 the same web-based conference. Note
   that this is one of many ways of implementing this functionality, and
   it should not be construed as the preferred way of establishing a
   web-based conference.

```
     Session-ID             Conference
        ---        Alice       Focus           Bob          Carol
                      |          |              |              |
                      |<** HTTPS *****>|        |              |
                      | Transaction    |        |              |
                      |          |              |              |
       {M,N}          |<----INVITE-----|        |              |
       {A,M}          |-----200 OK---->|        |              |
       {M,A}          |<-----ACK-------|        |              |
                      |<=====RTP======>|        |              |
                      |          |              |              |
                      |          |<** HTTPS *****>|             |
                      |          | Transaction    |             |
                      |          |              |              |
       {M,N}          |          |-----INVITE---->|            |
       {B,M}          |          |<----200 OK-----|            |
       {M,B}          |          |------ACK------>|            |
                      |          |<=====RTP======>|            |
                      |          |              |              |
                      |          |<***************** HTTPS *****>|
                      |          |              | Transaction    |
                      |          |              |              |
       {M,N}          |          |--------------------INVITE----->|
       {C,M}          |          |<-------------------200 OK------|
       {M,C}          |          |--------------------ACK------->|
                      |          |<===================RTP======>|
```
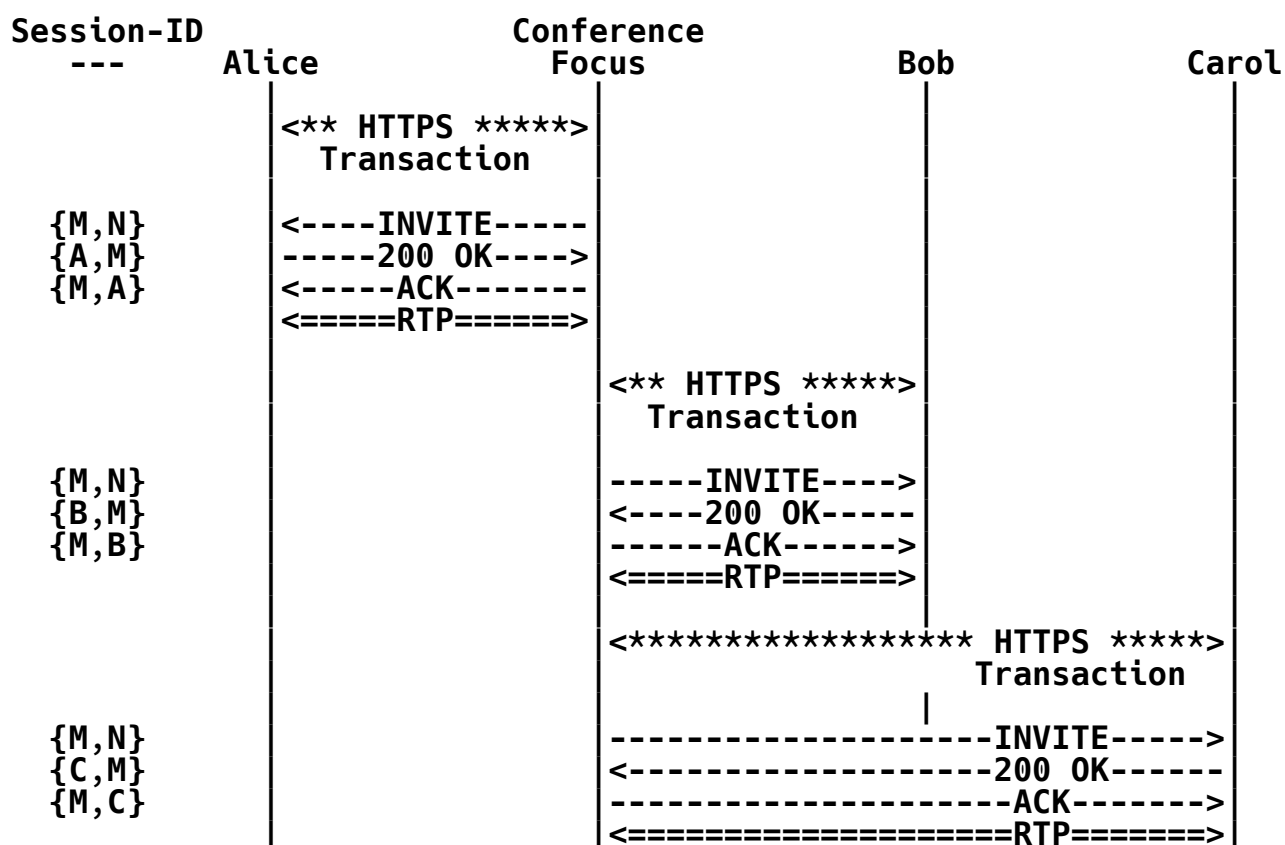
                  Figure 5: Single Focus Web-Based Conference

   General operation of this example:

   o Alice communicates with the web server that she wants to join a
      certain meeting by using a meeting number and including 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 (A/V) play-out devices could be separate
      units.

   o The Conference Focus server sends the INVITE request (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 a 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, 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

   Expanding 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 field value parameter just for this purpose.


```
     Session-ID
        ---       MCU-1             MCU-2             MCU-3             MCU-4
                    |                 |                 |                 |
     {M',N}         |----INVITE----->|                 |                 |
     {J,M'}         |<---200 OK------|                 |                 |
     {M',J}         |-----ACK------->|                 |                 |
```
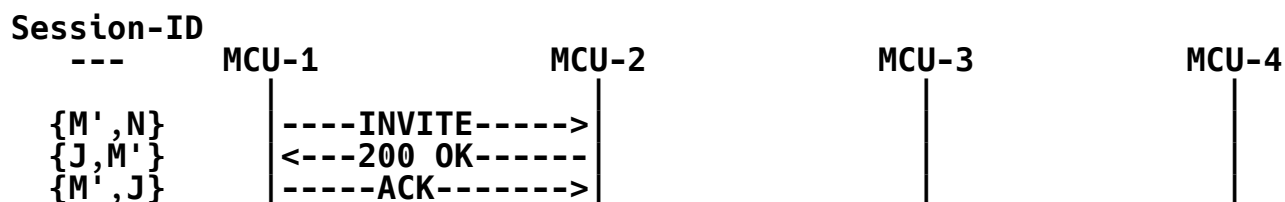
       Figure 6: MCUs Communicating Session Identifier UUID for Bridge

   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 be informed in one of two ways:

   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 then they inform downstream MCUs of this
      common UUID that each will be using for this one conference.

```
     Session-ID
       ---        MCU-1            MCU-2            MCU-3            MCU-4
      {M',N}      |----INVITE----->|                |                |
      {J,M'}      |<---200 OK------|                |                |
      {M',J}      |-----ACK------->|                |                |
                  |                |                |                |
      {M',N}      |---------------------------------INVITE----->|    |
      {K,M'}      |<--------------------------------200 OK------|    |
      {M',K}      |---------------------------------ACK------->|     |
                  |                |                |                |
      {M',N}      |----------------------------------------------------INVITE----->|
      {L,M'}      |<---------------------------------------------------200 OK------|
      {M',L}      |----------------------------------------------------ACK------->|
```
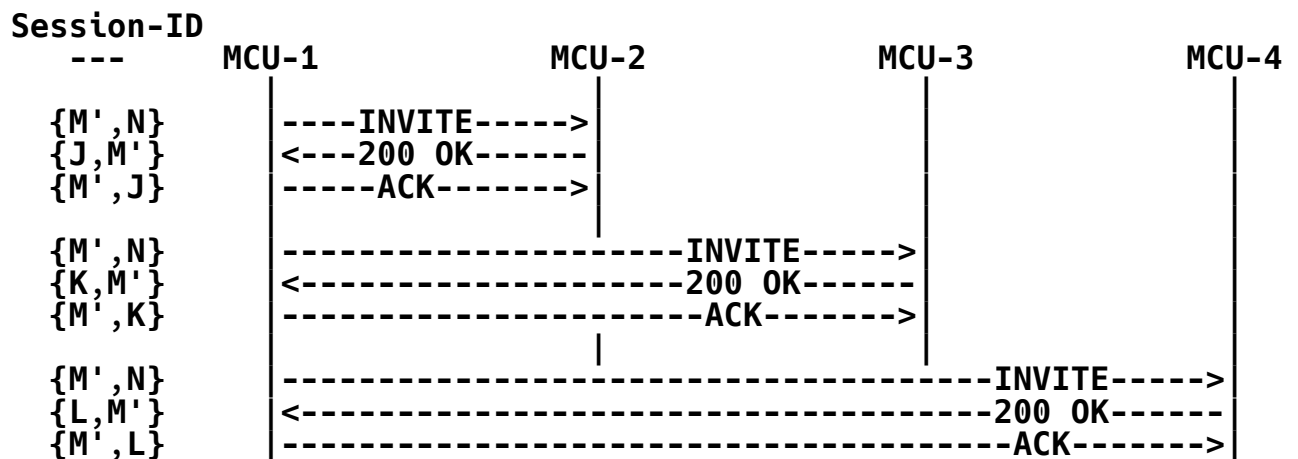
<p align="center">Figure 7: MCU Communicating
Session Identifier UUID to More Than One MCU</p>

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"
   Contact header field value 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, Robert for example, 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}      |----INVITE----->|               |               |
        {J,M'}      |<---200 OK------|               |               |
        {M',J}      |-----ACK------->|               |               |

        {M',N}      |------------------------INVITE----->|            |
        {K,M'}      |<-----------------------200 OK------|            |
        {M',K}      |------------------------ACK------->|             |

        {R,N}       |                |               |<---INVITE-----|
        (M',R}      |                |               |----200 OK---->|
        {R,M'}      |                |               |<----ACK-------|
```
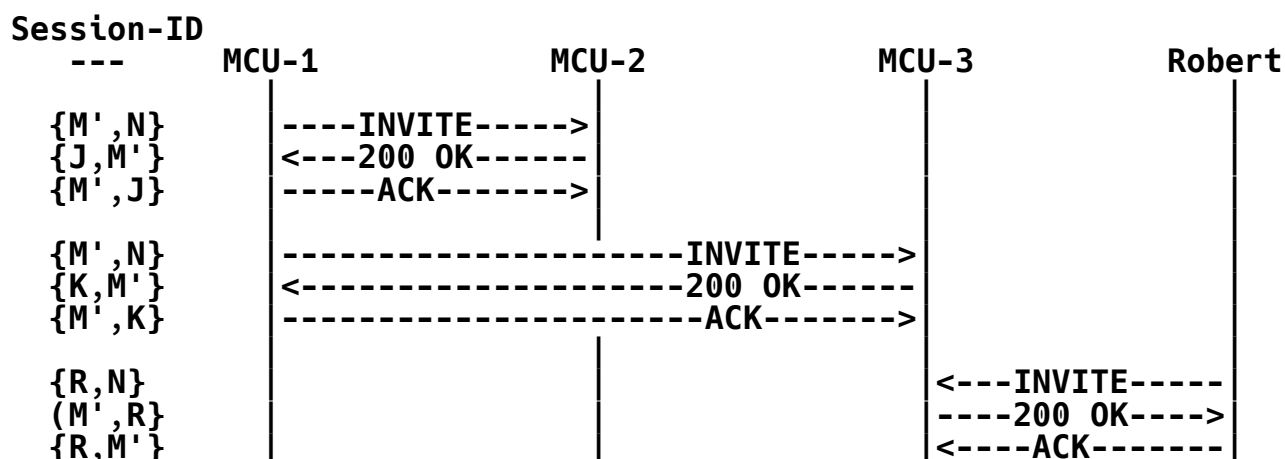
              Figure 8: A UA Calling Into a Cascaded MCU UUID

   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. As is the case with any standard
      implementation, he includes a nil "remote-uuid".

   o The cascaded MCU, upon receiving this INVITE request 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 three-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.
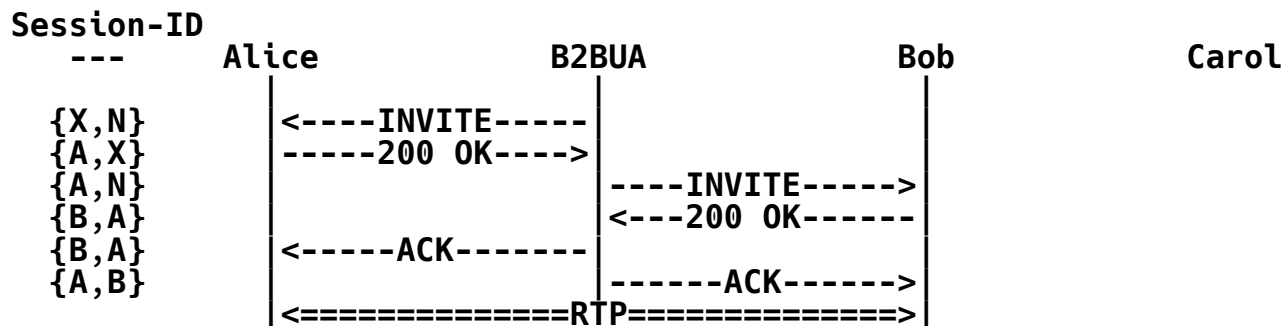
```
     Session-ID
        ---      Alice              B2BUA               Bob              Carol
                   |                  |                  |                  |
      {X,N}        |<----INVITE-----  |                  |                  |
      {A,X}        |-----200 OK---->  |                  |                  |
      {A,N}        |                  |----INVITE----->  |                  |
      {B,A}        |                  |<---200 OK------  |                  |
      {B,A}        |<-----ACK-------  |                  |                  |
      {A,B}        |                  |------ACK------>  |                  |
                   |<=============RTP=============>|                  |
```

            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 setup 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 it discards its own
   UUID {X}, replacing this with a nil UUID value in the INVITE
   request to Bob as if this came from Alice originally.

o Bob receives and accepts this INVITE request 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 [RFC5359],
   Section 2.9 ("Call Forwarding - No Answer").

```
     Session-ID    Alice          SIP Server        Bob-1              Bob-2
                     |                |                |                  |
       {A,N}         |----INVITE----->|                |                  |
       {A,N}         |                |---INVITE---->  |                  |
       {N,A}         |<--100 Trying---|                |                  |
       {B1,A}        |                |<-180 Ringing-  |                  |
       {B1,A}        |<--180 Ringing--|                |                  |
                     |                |                |                  |
                     |                |*Request Timeout*                  |
                     |                |                |                  |
       {A,N}         |                |---CANCEL---->  |                  |
       {B1,A}        |                |<--200 OK-----  |                  |
       {B1,A}        |                |<---487-------  |                  |
       {A,B1}        |                |---- ACK ---->  |                  |
                     |                |                |                  |
       {N,A}         |<-181 Call Fwd--|                |                  |
                     |                |                |                  |
       {A,N}         |                |------------------INVITE------>    |
       {B2,A}        |                |<---------------180 Ringing---     |
       {B2,A}        |<-180 Ringing---|                |                  |
       {B2,A}        |                |<---------------200 OK ------      |
       {B2,A}        |<--200 OK-------|                |                  |
       {A,B2}        |----ACK-------->|                |                  |
       {A,B2}        |                |-----------------ACK-------->      |
                     |                |                |                  |
                     |<=========== Both way RTP Established ==========>   |
                     |                |                |                  |
       {A,B2}        |----BYE-------->|                |                  |
       {A,B2}        |                |--------------------BYE------->    |
       {B2,A}        |                |<-----------------200 OK------     |
       {B2,A}        |<--200 OK-------|                |                  |
                     |                |                |                  |
```
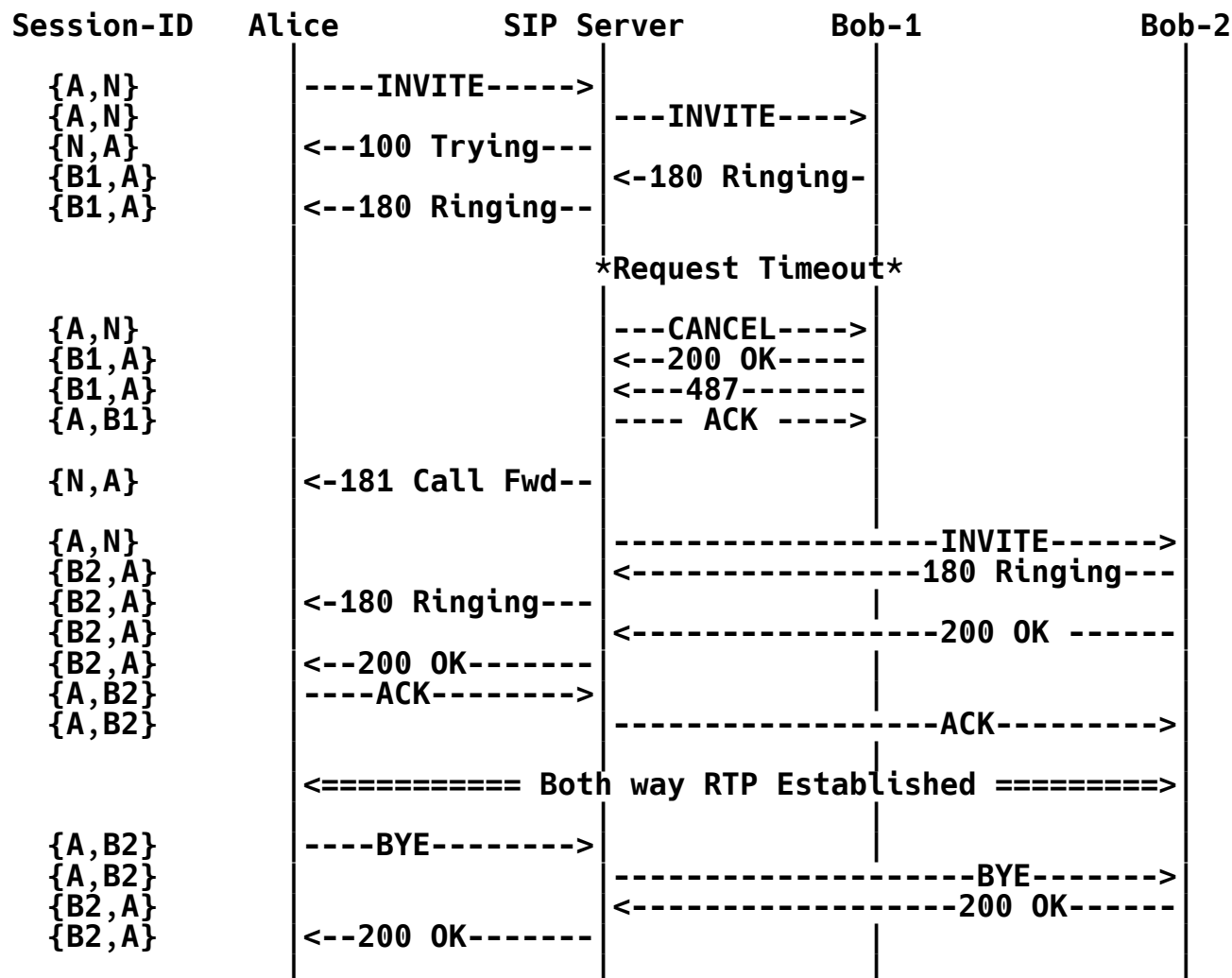
   Figure 10: Session Identifier in the 100 Trying and CANCEL Messaging

   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 she 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 request, generates the optional provisional response 100
      Trying. Since the SIP server has no knowledge of 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 request with her
      UUID value in the Session-ID header field.

   o Bob-1 responds to this INVITE request 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 request and a CANCEL request
      always uses the same Session-ID header field as the original
      INVITE request. Thus, the CANCEL request 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's (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

      Session-ID      Bob                   Alice                 Carol

         {A,N}         |<-----INVITE--------|                      |
         {B,A}         |------200 OK------->|                      |
         {A,B}         |<------ACK----------|                      |

         {B,A}         |--INVITE {hold}---->|                      |
         {A,B}         |<-200 OK------------|                      |
         {B,A}         |--- ACK ----------->|                      |

         {B,A}         |--REFER------------>|(Refer-To:Carol)      |
         {A,B}         |<-202 Accepted------|                      |

         {A,B}         |<NOTIFY {100 Trying}|                      |
         {B,A}         |-200 OK------------>|                      |

         {A,N}         |                    |--INVITE------------->|
         {C,A}         |                    |<-200 OK--------------|
         {A,C}         |                    |---ACK--------------->|

         {A,B}         |<--NOTIFY {200 OK}--|                      |
         {B,A}         |---200 OK---------->|                      |

         {B,A}         |--BYE-------------->|                      |
         {A,B}         |<-200 OK------------|                      |
         {C,A}         |                    |<------------BYE-----|
         {A,C}         |                    |------------200 OK->|
```
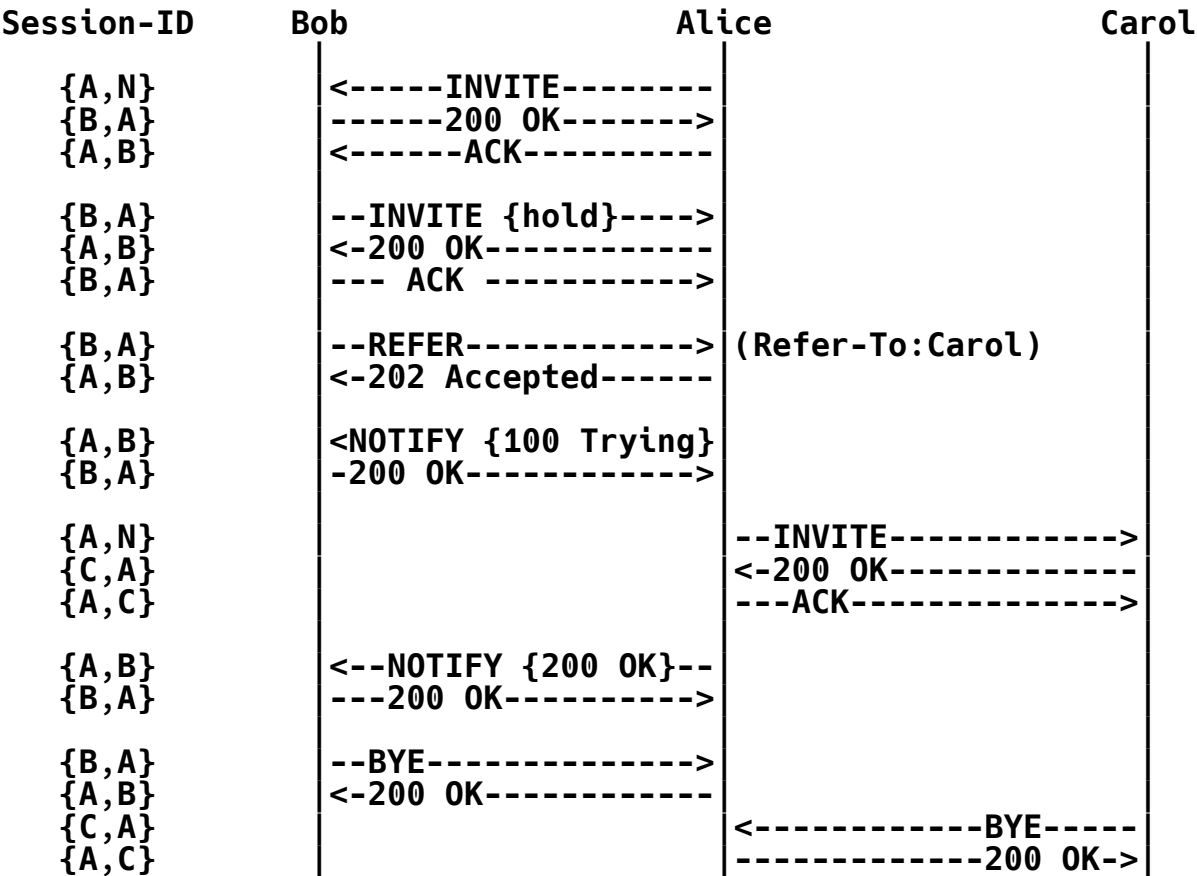
                    Figure 11: Out-Of-Dialog Call Transfer

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 request 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. This section contains 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 implementor 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
   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 nonessential 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 7989

Header name: 'Session-ID'

Compact form: none

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

13.2. Registration of the "remote" Parameter

The following parameter has been added to the "Header Field
Parameters and Parameter Values" section of the "Session Initiation
Protocol (SIP) Parameters" registry:

Header Field	Parameter Name	Predefined Values	Reference
Session-ID	remote	No	[RFC7989]

14. References

14.1. 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>.

14.2. Informative References

   [H.323]   International Telecommunications Union, "Packet-based
               multimedia communications systems", ITU-T
               Recommendation H.323, December 2009.

   [H.460.27] International Telecommunications Union, "End-to-End
               Session Identifier for H.323 Systems", ITU-T
               Recommendation H.460.27, 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>.

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.

Authors' Addresses

    Paul E. Jones
    Cisco Systems, Inc.
    7025 Kit Creek Rd.
    Research Triangle Park, NC 27709
    United States of America

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

    Gonzalo Salgueiro
    Cisco Systems, Inc.
    7025 Kit Creek Rd.
    Research Triangle Park, NC 27709
    United States of America

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

    Chris Pearce
    Cisco Systems, Inc.
    2300 East President George Bush Highway
    Richardson, TX 75082
    United States of America

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

    Paul Giralt
    Cisco Systems, Inc.
    7025 Kit Creek Rd.
    Research Triangle Park, NC 27709
    United States of America

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