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

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

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

Abstract

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

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

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

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

Table of Contents

1. Introduction

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

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

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

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

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

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

3. Session Identifier Requirements and Use Cases

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

4. Constructing and Conveying the Session Identifier

4.1. Constructing the Session Identifier

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

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

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

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

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

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

4.2. Conveying the Session Identifier

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

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

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

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

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

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

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

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

5. Transmitting the Session Identifier in SIP

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

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

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

local-uuid    = sess-uuid / null

remote-uuid   = sess-uuid / null

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

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

remote-param  = "remote" EQUAL remote-uuid

null          = 32("0")
```

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

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

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

The "local-uuid" in the Session-ID header represents the UUID value

of the UA transmitting the message. If the UA transmitting the message previously received a UUID value from its peer endpoint, it

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

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

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

6. Endpoint Behavior

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

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

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

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

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

A UA MUST assume that the UUID value of the peer UA MAY change at any

time due to service interactions. If the UUID value of the peer UA changes, the UA MUST accept the new UUID as the peer's UUID and

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

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

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

7. Processing by Intermediaries

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

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

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

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

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

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

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

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

8. Associating Endpoints in a Multipoint Conference

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

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

Intermediary devices, such as proxies or session border controllers, or network diagnostics equipment might assume that when they see two or more sessions with different Session Identifiers, but with one UUID in common, that the sessions are part of the same conference. However, the assumption that two sessions having one common UUID

being part of the same conference is not always correct. In a SIP forking scenario, for example, there might also be what appears to be multiple sessions with a shared UUID value; this is intended. The

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

9. Various Call Flow Operations Utilizing the Session ID

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

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

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

9.1. Basic Session ID Construction with 2 UUIDs

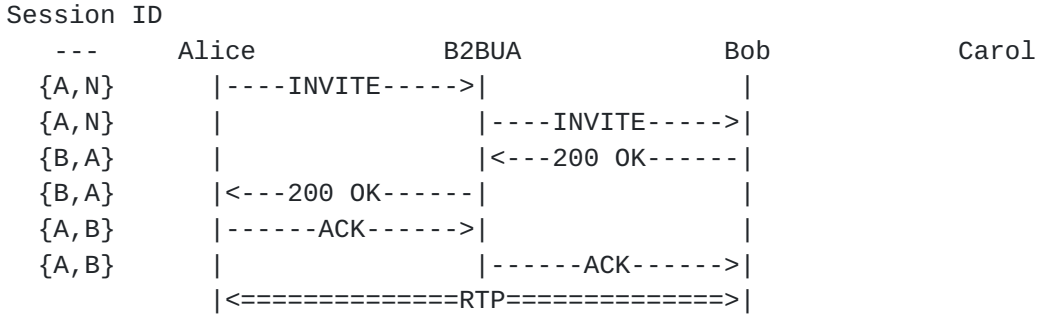


Figure 1 - Session ID Creation when Alice calls Bob

General operation of this example:

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

uuid" portion of the Session-ID header-value in the 200 OK SIP response.

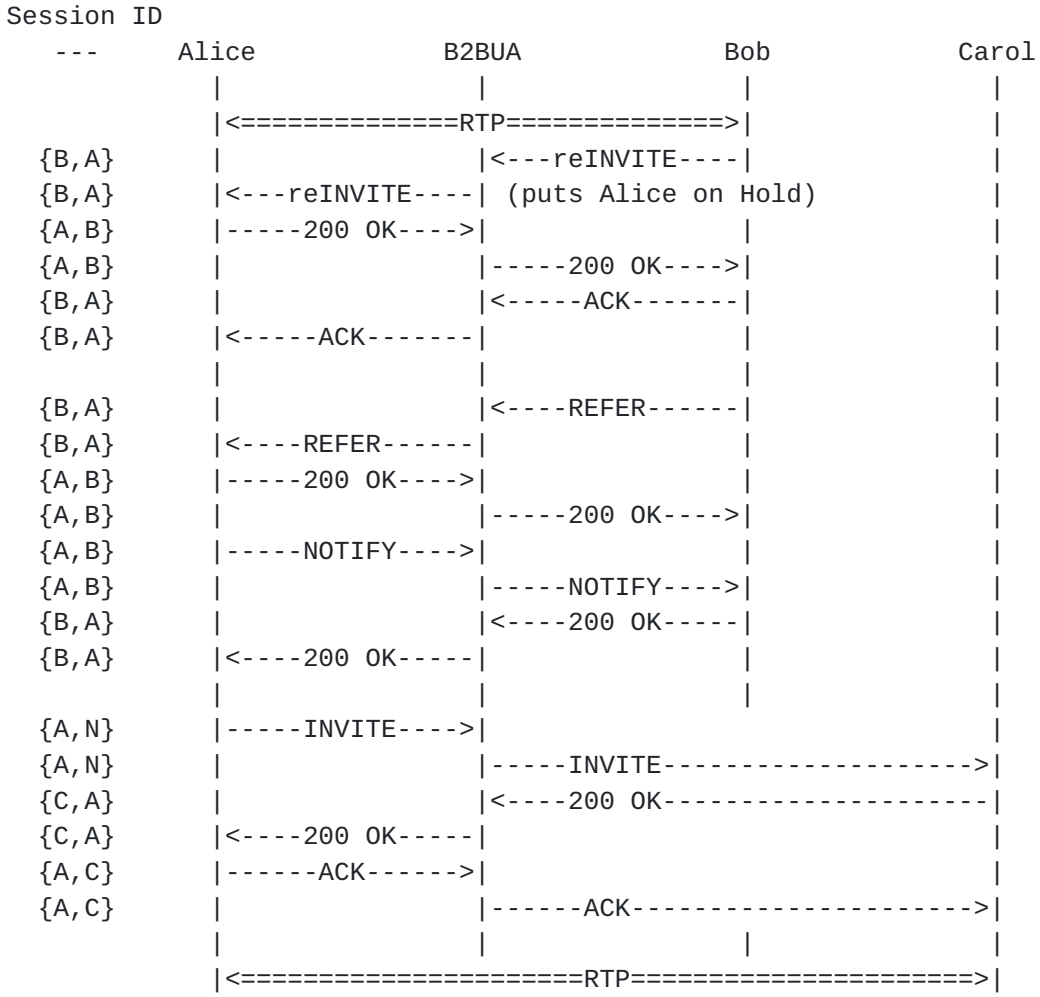
- o B2BUA receives the 200 OK response with a complete Session-ID header-value from UA-Bob, and transmits 200 OK towards UA-Alice with an unchanged Session-ID header-value.

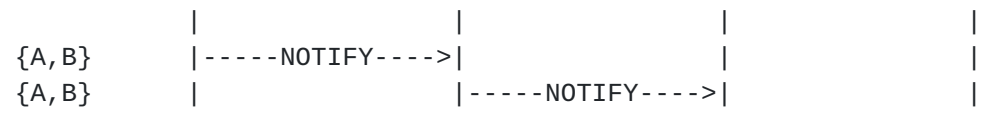
- o UA-Alice, upon reception of the 200 OK from the B2BUA, transmits the ACK towards the B2BUA. The construction of the Session-ID header-value in this ACK is that of Alice's UUID is the "local-uuid", and Bob's UUID populates the "remote-uuid" portion of the header-value.

- o B2BUA receives the ACK with a complete Session-ID header-value from UA-Alice, and transmits ACK towards UA-Bob with an unchanged Session-ID header-value.

9.2. Basic Call Transfer using REFER

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




{A,B}
NOTIFY
{A,B}
NOTIFY

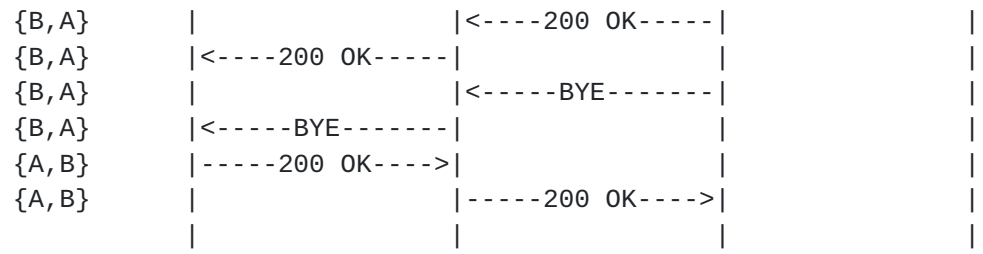


Figure 2 - Call Transfer using REFER

General operation of this example:

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

- UA-Bob requests Alice to call Carol, using a REFER transaction, as described in [RFC3515]. UA-Alice is initially put on hold, then told in the REFER who to contact with a new INVITE, in this case UA-Carol. This Alice-to-Carol dialog will have a new Call-ID, therefore it requires a new Session-ID header-value. The wrinkle here is we can, and will, use Alice's UUID from her existing dialog with Bob in the new INVITE to Carol.

- UA-Alice retains her UUID from the Alice-to-Bob call {A} when requesting a call with UA-Carol. This is placed in the "local-uuid" portion of the Session-ID header-value, at the same time inserting a null "remote-uuid" value (because Carol's UA has not yet received the UUID value). This same UUID traverses the B2BUA unchanged.

- UA-Carol receives the INVITE with a Session ID UUID {A,N}, replaces the A UUID value into the "remote-uuid" portion of the Session-ID header-value and creates its own UUID {C} and places this value in the "local-uuid" portion of the Session-ID header-value - thereby removing the N (null) value altogether. This combination forms a full Session ID {C,A} in the 200 OK to the INVITE. This Session-ID header-value traverses the B2BUA unchanged towards UA-Alice.

- UA-Alice receives the 200 OK with the Session ID {C,A} and both responds to UA-Carol with an ACK (just as in Figure 1 - switches places of the two UUID fields), and generates a NOTIFY to Bob with a Session ID {A,B} indicating the call transfer was successful.

- It does not matter which UA terminates the Alice-to-Bob call; Figure 2 shows UA-Bob doing this transaction.

9.3. Basic Call Transfer using reINVITE

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

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

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

Figure 3 - Call transfer using reINVITE

General operation of this example:

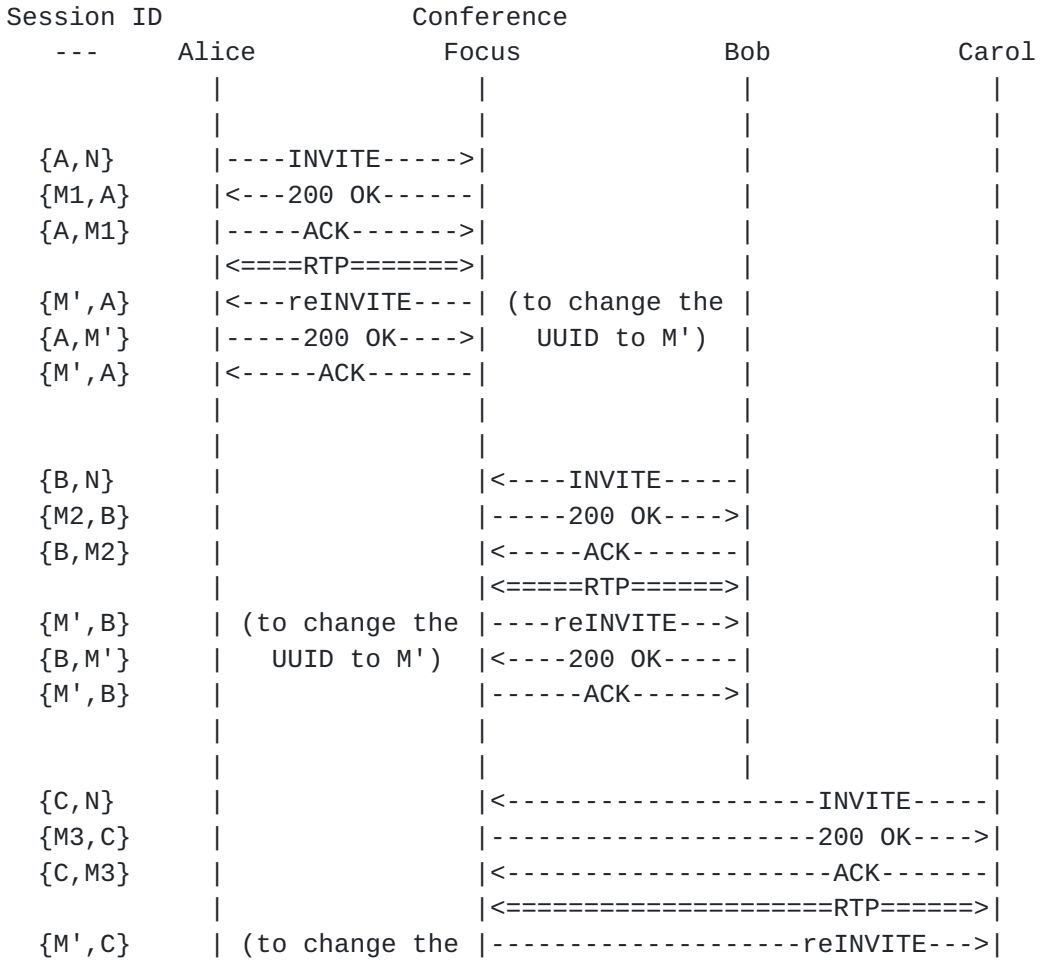
- We assume the call between Alice and Bob from Section 9.1 is operational with Session ID {A,B}.
- Bob sends a reINVITE to Alice (with the Session-ID "local-uuid" = Bob's UUID and "remote-uuid" = Alice's UUID), informing her to transfer her existing call to Carol.
- The B2BUA intercepts this reINVITE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

- Carol receives the INVITE and accepts the request and adds her UUID {C} to the Session ID for this session {"local-uuid" = "C", "remote-uuid" = "A"}.

- Bob terminates the call with a BYE using the Session ID {"local-uuid" = "B", "remote-uuid" = "A"}. The B2BUA intercepts this BYE and responds to Bob since Alice and Carol are now in a new call.

9.4. Single Focus Conferencing

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



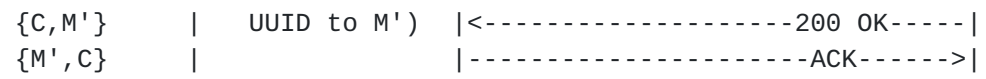

{C,M'}
UUID to M')
200 OK
{M',C}
ACK

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 9.1). What is unique about this call is the second step: the conference server calls back with a reINVITE request with its second UUID, but maintaining the UUID Alice sent in the first INVITE. This subsequent UUID from the conference server will be the same for each UA that calls into this conference server participating in this same conference bridge/call, which is generated once Alice typically authenticates and identifies which bridge she wants to participate on.

- o Alice sends an INVITE to the conference server with her UUID {A} and a "remote" UUID of N.

- o The conference server responds with a 200 OK response which replaces the N UUID with a temporary UUID ("M1") as the "local-uuid" and a "remote-uuid" = "A".

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

- o Once Alice, the user, gains access to the IVR for this conference server, she enters a specific conference ID and whatever passcode (if needed) to enter a specific conference call.

- o Once the conference server is satisfied Alice has identified which conference she wants to attend (including any passcode verification), the conference server reINVITEs Alice to the specific conference and includes the Session-ID header-value of "local-uuid" = "M'" (and "remote-uuid" = "A") for that conference. All valid participants in the same conference will receive this same UUID for identification purposes and to better enable monitoring, and tracking functions.

- o Bob goes through this two-step process of an INVITE transaction, followed by a reINVITE transaction to get this same UUID ("M'") for that conference.

- o In this example, Carol (and each additional user) goes through the same procedures and steps as Alice and Bob to get on this

same conference.

9.5. Single Focus Conferencing using WebEx

Alice, Bob and Carol call into same Webex conference.

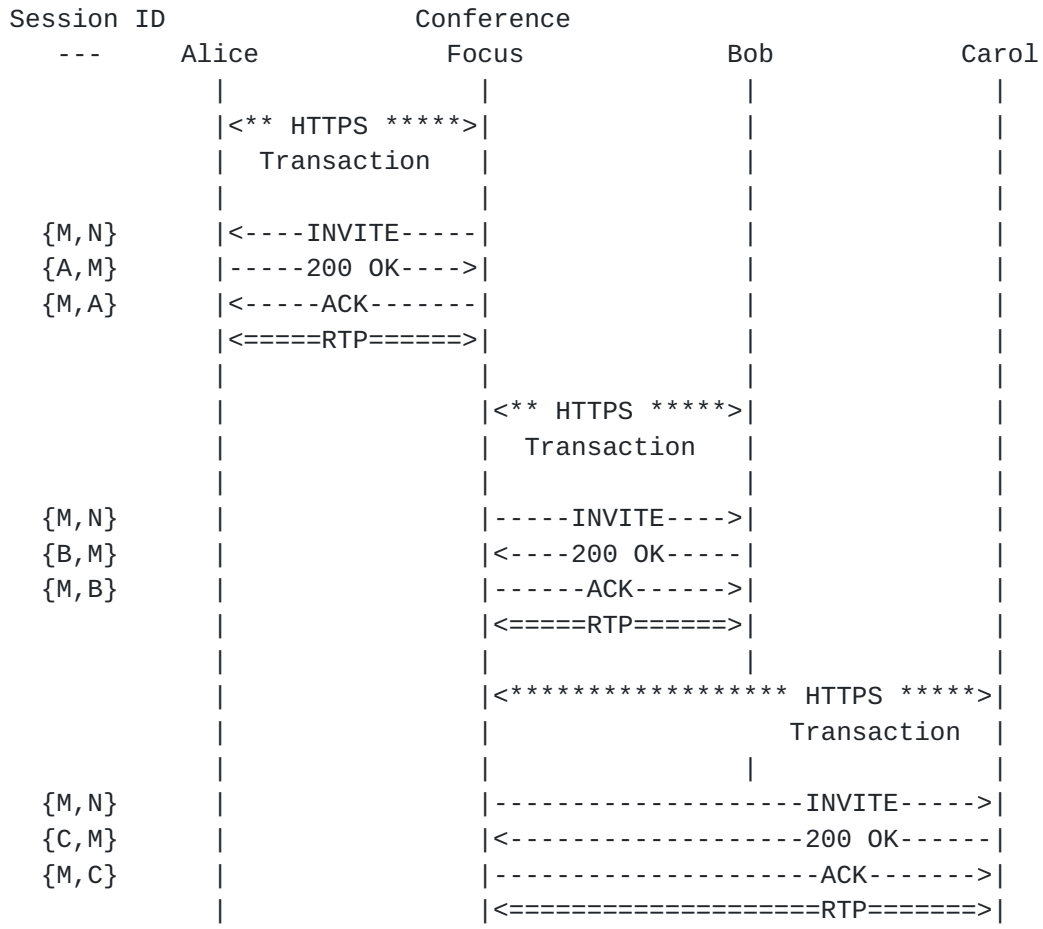


Figure 5 - Single Focus Webex Conference

General operation of this example:

- Alice communicates with Webex server with desire to join a certain meeting, by meeting number; also includes UA-Alice's contact information (phone number, URI and/or IP address, etc.) for each device she wants for this conference call. For example, the audio and video play-out devices could be separate units.

- Conference Focus server sends INVITE (Session-ID header-value "local-uuid" = M and a remote UUID of N, where M equals the "local-uuid" for each participant on this conference bridge) to UA-Alice to start session with that server for this A/V conference call.

- Upon receiving the INVITE request from the conference focus server, Alice responds with a 200 OK. Her UA moves the "local-uuid" unchanged into the "remote-uuid" field, and generates her

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

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

9.6. Cascading Conference Bridge Support for the Session ID

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

```
   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 ID UUID for Bridge

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

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

- o All by one MCU (that initially generates the UUID for the conference),

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

```
   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------->|               |
                |                |                |               |
```

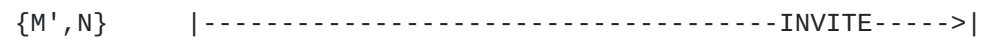

{M',N}
INVITE

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

General operation of this example:

- The MCU generating the Session ID UUID communicates this in a separate INVITE, having a Contact header with the 'isfocus' header parameter. This will identify the MCU as what RFC 4579 conference-aware SIP entity.

- An MCU that receives this {M',N} UUID pair in an inter-MCU transaction, can communicate the M' UUID in a manner in which it was received (though this time this second MCU would be the UAC MCU), unless local policy dictates otherwise.

9.6.1. Calling into Cascaded Conference Bridge for the Session ID

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

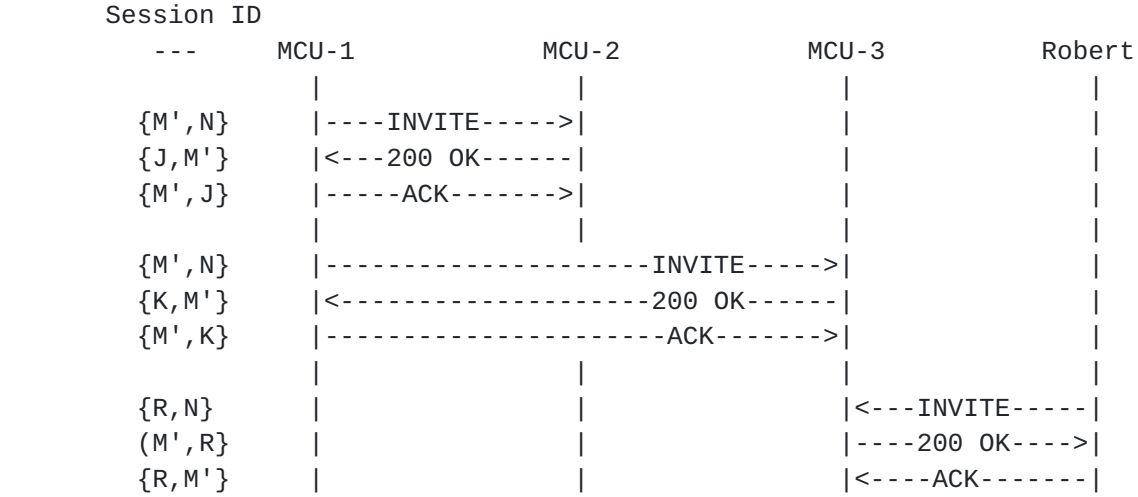


Figure 8 - A UA Calling into a Cascaded MCU UUID

General operation of this example:

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

- The cascaded MCU, upon receiving this INVITE from Robert, replaces the null UUID with the UUID value communicated from MCU-1 for this conference call as the "local-uuid" in the SIP response. Thus, moving Robert's UUID "R" to the "remote-uuid" value.

- The ACK has the Session-ID {R,M'}, completing the 3-way handshake for this call establishment. Robert has now joined the conference call originated from MCU-1.

9.7. Basic 3PCC for two UAs

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

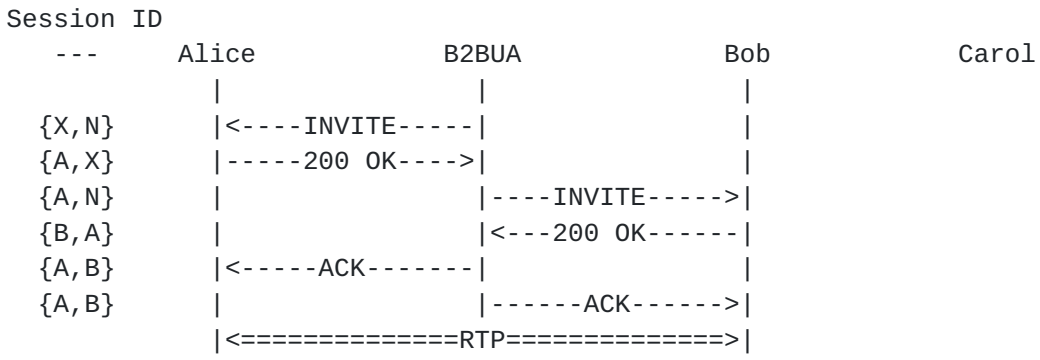


Figure 8 - 3PCC initiated call between Alice and Bob

General operation of this example:

- Some out of band procedure directs a B2BUA (or other SIP server) to have Alice and Bob talk to each other. In this case, the SIP server MUST be transaction stateful, if not dialog stateful.

- The SIP server INVITEs Alice to a session and uses a temporary UUID {X} and a null UUID pairing.

- Alice receives and accepts this call set-up and replaces the null UUID with her UUID {A} in the Session ID, now {A,X}.

- The transaction stateful SIP server receives Alice's UUID {A} in the local UUID portion and keeps it there, and discards its own UUID {X}, replacing this with a null UUID value in the INVITE to Bob as if this came from Alice originally.

- Bob receives and accepts this INVITE and adds his own UUID {B} to the Session ID, now {B,A} for the response.

- And the session is established.

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

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

9.8.1. Session ID Handling in a 100 Trying SIP Response

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

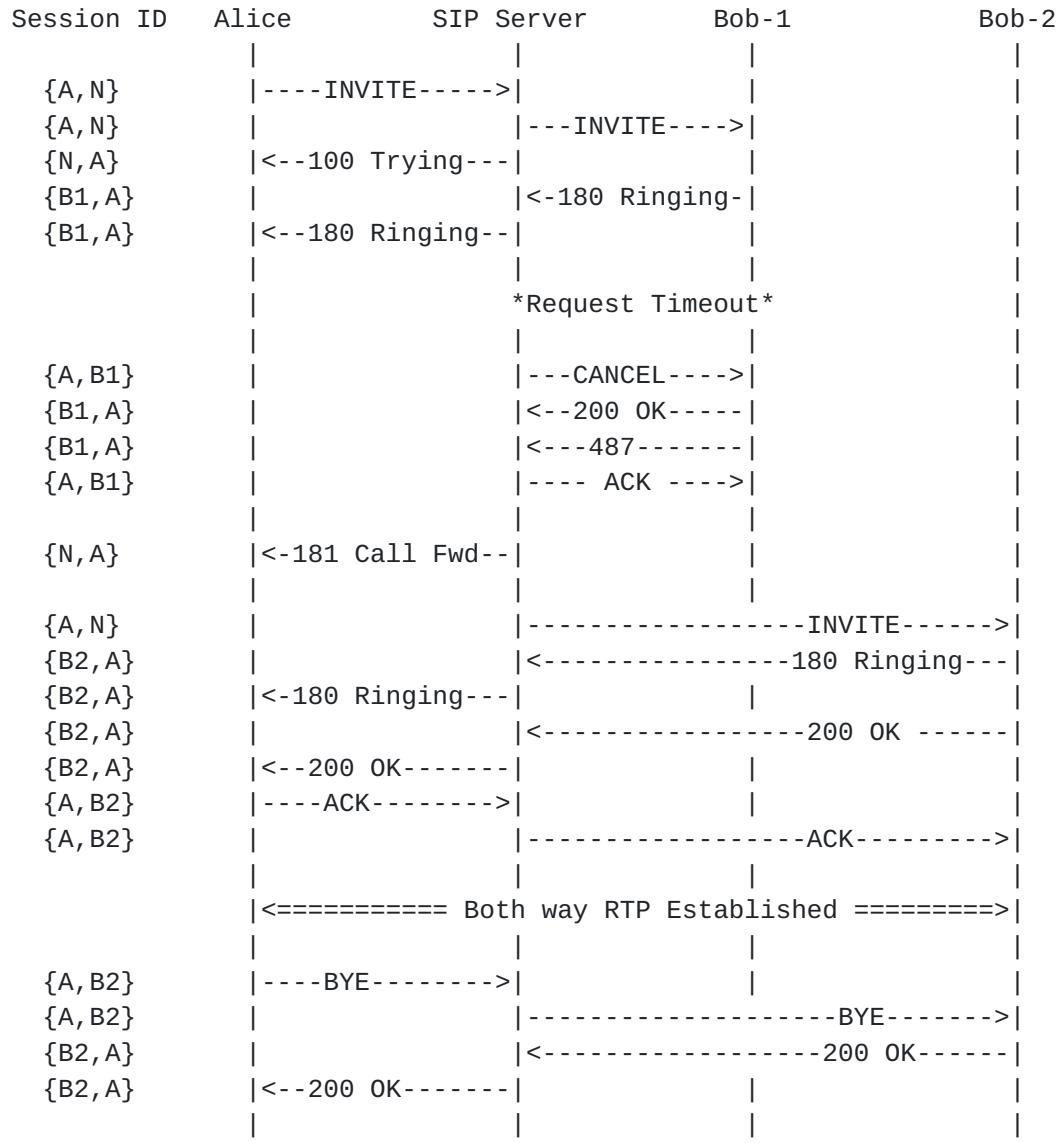


Figure 9 - Session ID in the 100 Trying and CANCEL Messaging

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

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

General operation of this example:

- Alice generates an INVITE request because she wants to invite Bob to join her session. She creates a UUID as described in section 9.1, and places that value in the "local-uuid" field of the Session-ID header-value. Alice also generates a "remote-uuid" of null and sends this along with the "local-uuid".

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

9.8.2. Session ID in a CANCEL SIP Request

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

General operation of this example:

- In Figure 9 above, Alice generates an INVITE with her UUID value in the Session-ID header-value.

- Bob-1 responds to this INVITE with a 180 Ringing. In that response, he includes his UUID in the Session-ID header-value (i.e., {B1,A}); thus completing the Session-ID header-value for this session, even though no final response has been generated by any of Bob's UAs.

- This means that if the SIP server were to generate a SIP request within this session, in this case a CANCEL request, it would have a complete Session ID to include in that request. In this case, the "local-uuid" = "A", and the "remote-uuid" = "B1".

- As it happens with this CANCEL, the SIP server intends to invite another UA of Bob (i.e., B2) for Alice to communicate with.

- In this example call-flow, taken from RFC 5359, Section 2.9, a 181 (Call is being Forwarded) response is sent to Alice. Since the SIP server generated this SIP request, and has no knowledge

of Bob-2's UUID value, it cannot include that value in this 181. Thus, and for the exact reasons the 100 Trying including the

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

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

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

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

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

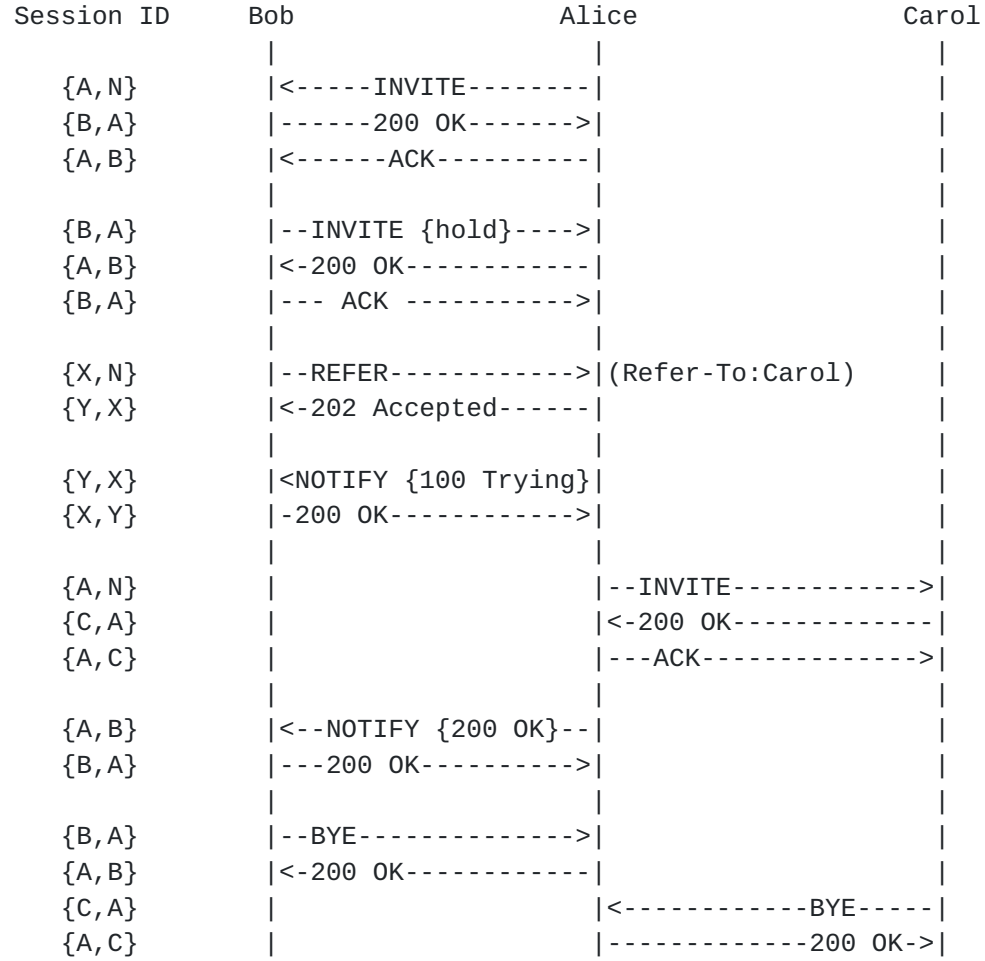


Figure 10: Basic Transfer Call Flow

General operation of this example:

- Just as in Section 9.2, Figure 2, Alice invites Bob to a session, and Bob eventually transfers Alice to communicate with Carol.

- What is different about the call-flow in Figure 10 is that Bob's REFER is not in-dialog, meaning it would have the same UUID pair. Rather, in this case, Bob's using an out-of-dialog REFER, meaning Bob generates a new UUID for this SIP request, and Alice, subsequently would also generate a new UUID for the 202 (Accepted) response, replacing the null "remote-uuid in the REFER.

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

10. Compatibility with a Previous Implementation

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

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

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

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

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

- since no option tags or feature tags are to be used for distinguishing versions, the presence and order of any "remote-

uuid" value within the Session-ID header value is to be used to distinguish implementation versions.

- if a SIP request has a "remote-uuid" value, this comes from a standard implementation, and not a pre-standard one.

- if a SIP request has no "remote-uuid" value, this comes from a pre-standard implementation, and not a standard one. In this case, one UUID is used to identify this dialog, even if the responder is a standard implementation of this specification.

- if a SIP response has a non-null "local-uuid" that is 32 octets long, this response comes from a standard implementation. There are two exceptions to this rule: a 100 Trying response and a 181 Call Forwarded response will have null "local-uuid" values.

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

- if a SIP response arrives that has the same value of Session-ID UUIDs in the same order as was sent, this comes from a pre-standard implementation, and MUST NOT be discarded for not altering the null "remote-uuid". In this case, any new transaction within this dialog MUST preserve the order of the two UUIDs within all Session-ID header-values, including the ACK, until this dialog is terminated.

- if a SIP response only contains the "local-uuid" that was sent originally, this comes from a pre-standard implementation and MUST NOT be discarded for removing the null "remote-uuid". In this case, all future transactions within this dialog MUST contain only the UUID received in the first SIP response. Any new transaction starting a new dialog from the standard Session-ID implementation MUST include and "local-uuid" and a null "remote-uuid", even if that new dialog is between the same two UAs.

- Standard implementations SHOULD NOT expect pre-standard implementations to be consistent in their implementation, even within the same dialog. For example, perhaps the first, third and tenth responses contain a "remote-uuid", but all the others do not. This behavior MUST be allowed by implementations of this specification.

- All of this does not apply to other parameters that might be defined in the future, i.e., currently unknown. They are discarded.

11. Security Considerations

When creating a UUID value, endpoints SHOULD ensure that there is no user or device-identifying information contained within the UUID. In

some environments, though, use of a MAC address, which is one option when constructing a UUID, may be desirable, especially in some

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

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

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

12. IANA Considerations

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

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

RFC number: RFC XXXX

Header name: 'Session-ID'

Compact form: none

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

12.2. Registration of the "remote" Parameter

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

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

13. Acknowledgments

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

14. References

14.1. Normative References

[RFC3261] Rosenberg, J., et al., "SIP: Session Initiation Protocol", RFC 3261, June 2002.

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

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

[RFC4122] Leach, P., Mealling, M., Salz, R., "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

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

[RFC4579] Johnston, A., Levin, O., "Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents", RFC 4579, August 2006.

[RFC3891] Mahy, R., Biggs, B., Dean, R., 'The Session Initiation Protocol (SIP) "Replaces" Header', RFC 3891, September 2004.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[I-D.kaplan-insipid-session-id] Kaplan, H., "A Session Identifier for the Session Initiation Protocol (SIP)", August 2013.

14.2. Informative References

[RFC3550] Schulzrinne, H., et al., "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[I-D.ietf-insipid-session-id-reqts] Jones, et al., "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks", draft-ietf-insipid-session-id-reqts-11, February 2014.

[RFC5359] Johnston, A., et al., "Session Initiation Protocol Service Examples", RFC 5359, October 2008.

[RFC5589] Sparks, R., Johnston, A., Petrie, D., "Session Initiation

Protocol (SIP) Call Control - Transfer", RFC 5359, June 2009.

Author's Addresses

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

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

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

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

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

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

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

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