Internet Engineering Task Force
Internet-Draft
Intended status: Best Current Practice
Expires: Dec 30, 2007

M. HASEBE
J. KOSHIKO
Y. SUZUKI
T. YOSHIKAWA
NTT-East
P. Kyzivat
Cisco Systems, Inc.
Jun 30th, 2007

Examples call flow in race condition on Session Initiation Protocol
draft-ietf-sipping-race-examples-02.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of 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 December 30, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

This document gives examples of the Session Initiation Protocol (SIP) call flows in race condition. Call flows in race condition are confusing, and this document shows the best practice to handle them. The elements in these call flows include SIP User Agents and SIP Proxies. Call flow diagrams and message details are shown.

Table of Contents

1. Overview 2
1.1 General Assumptions. 3
1.2 Legend for Message Flows 3
1.3 SIP Protocol Assumptions 3
2. The Dialog State Machine for INVITE dialog usage 4
3. Race Conditions. 9
3.1 Receiving message in the Moratorium State. 9
3.1.1 Receiving Initial INVITE retransmission. 10
3.1.2 Receiving CANCEL(Early state). 11
3.1.3 Receiving BYE (Early state). 13
3.1.4 Receiving re-INVITE (Established state)(case 1). 15
3.1.5 Receiving re-INVITE (Established state)(case 2). 19
3.1.6 Receiving BYE (Established state). 22
3.2 Receiving message in the Mortal State. 24
3.2.1 Receiving BYE(Established state) 24
3.2.2 Receiving re-INVITE(Established state) 26
3.2.3 Receiving 200 OK for re-INVITE(Established state). . . .28
3.2.4 Receiving ACK (Moratorium state) 31
3.3 Other race conditions. 32
3.3.1 Re-INVITE crossover. 32
3.3.2 UPDATE and re-INVITE crossover 36
3.3.3 Receiving REFER(Established state) 40
4. IANA Considerations. 42
5. Security Considerations. 42
6. Acknowledgements 42
7. References 42
7.1 Normative References 42
7.2 Informative References 42
Appendix A. BYE on the Early Dialog 43
Appendix B. BYE request overlapped on re-INVITE 44
Appendix C. UA's behavior for CANCEL. 46
Appendix D. Notes on the request in Mortal state. 48
Appendix E. Forking and receiving new To tags 49
Author's Addresses. 53
Intellectual Property and Copyright Statements. 54

1. Overview

The call flows shown in this document were developed in the design of a SIP IP communications network. These examples are of race condition, which stems from the dialog state transition mainly established by INVITE.

When implementing SIP, various complex situations may arise. Therefore, it will be helpful to provide implementors of the protocol

with examples of recommended terminal and server behavior.

This document clarifies SIP UA behaviors when messages cross each other as race conditions. By clarifying operation under race conditions, inconsistent interpretations between implementations are avoided and interoperability is expected to be promoted.

It is the hope of the authors that this document will be useful for SIP implementors, designers, and protocol researchers and will help them achieve the goal of a standard implementation of RFC 3261 [1].

These call flows are based on the version 2.0 of SIP defined in RFC 3261 [1] with SDP usage described in RFC 3264 [2].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [3].

1.1 General Assumptions

A number of architecture, network, and protocol assumptions underlie the call flows in this document. Note that these assumptions are not requirements. They are outlined in this section so that they may be taken into consideration and help understanding the call flow examples.

These flows do not assume specific underlying transport protocols such as TCP, TLS, and UDP. See the discussion in RFC 3261 [1] for details on the transport issues for SIP.

1.2 Legend for Message Flows

Dashed lines (---) and slash lines (/, \) represent signaling messages that are mandatory to the call scenario. (X) represents crossover of signaling messages. (->x, x<-) indicate that the packet is lost. The arrow indicates the direction of message flow. Double dashed lines (===) represent media paths between network elements.

Messages are identified in the figures as F1, F2, etc. These numbers are used for references to the message details that follow the Figure.
Comments in the message details are shown in the following form:

```
/* Comments.  */
```

1.3 SIP Protocol Assumptions

This document does not prescribe the flows precisely as they are

shown, but rather illustrates the principles for best practice.

They are best practice usages (orderings, syntax, selection of features for the purpose, or handling of error) of SIP methods, headers and parameters. Note: The flows in this document must not be copied as they are by implementors because additional characteristics were incorporated into the document for ease of explanation. To sum up, the procedures described in this document represent well-reviewed examples of SIP usage, which are best common practice according to IETF consensus.

For simplicity in reading and editing the document, there are a number of differences between some of the examples and actual SIP messages. For instance, Call-IDs are often repeated, CSeq often begins at 1, header fields are usually shown in the same order, usually only the minimum required header field set is shown, and other headers which would usually be included such as Accept, Allow, etc. are not shown.

Actors:

Element	Display Name	URI	IP Address
User Agent	Alice	sip:alice@atlanta.example.com	192.0.2.101
User Agent	Bob	sip:bob@biloxi.example.com	192.0.2.201
User Agent	Carol	sip:carol@chicago.example.com	192.0.2.202
Proxy Server		ss.atlanta.example.com	192.0.2.111

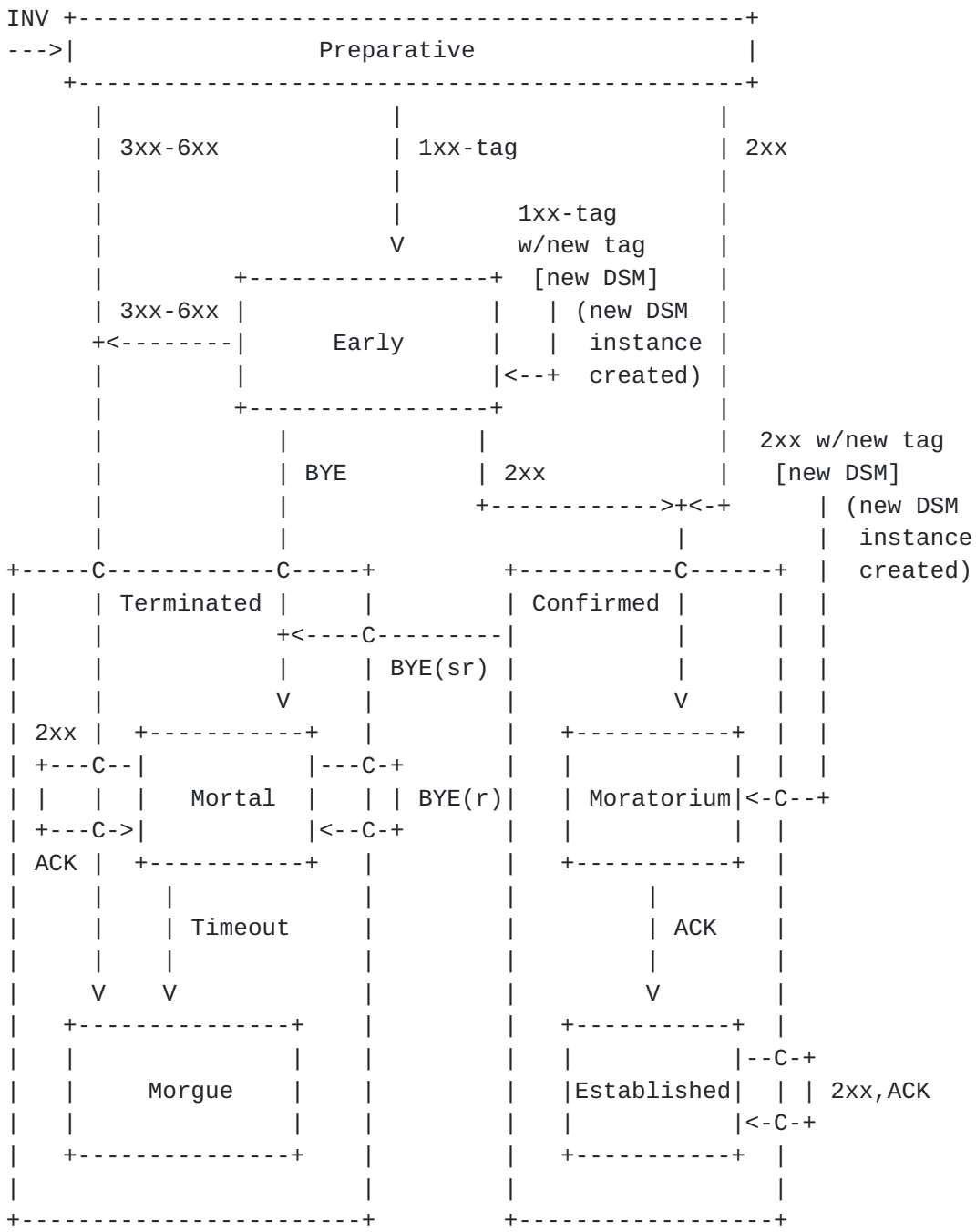
2. The Dialog State Machine for INVITE dialog usage

Race conditions are generated when the dialog state of the receiving side differs from that of the sending side.

For instance, a race condition occurs when UAC (User Agent Client) sends a CANCEL in the Early state while UAS (User Agent Server) is transiting from the Early state to the Confirmed state by sending a 200 OK to initial INVITE (indicated as "ini-INVITE" hereafter). The DSM (dialog state machine) for the INVITE dialog usage is presented as follows to help understanding UA's behavior in race conditions.

The DSM clarifies UA's behavior by subdividing the dialog state shown in RFC 3261 [1] into some internal states. We call the state before a dialog establishment the Preparative state. The Confirmed state is subdivided into two substates, the Moratorium and Established states, and the Terminated state is subdivided into the Mortal and Morgue states. Messages which are the triggers of the state transitions between these states are indicated with arrows. In this figure, messages which are not related to state transition are omitted.

Below are the DSMs for UAC and UAS respectively.

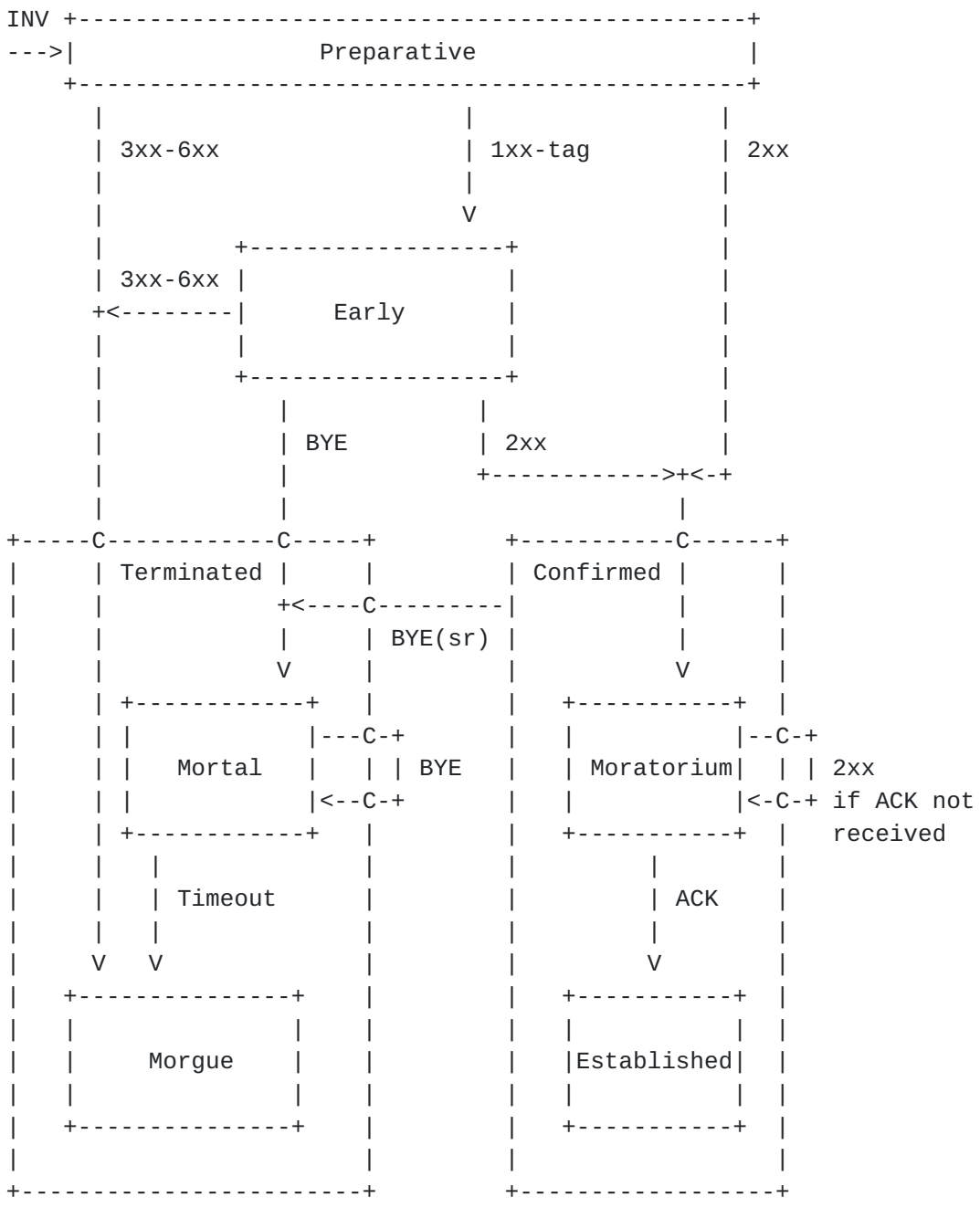


(r): indicates only reception is allowed.
Where (r) is not indicated, response means receive, request means send.

Figure 1. DSM for INVITE dialog usage (Caller)

Figure 1 represents the caller's DSM for the INVITE dialog usage.
Caller MAY send a BYE in the Early state, even though this behavior

is NOT RECOMMENDED. The BYE sent in the Early state terminates the early dialog with a specific To tag. That is, when a proxy is performing forking, the BYE is only able to terminate the early dialog with a particular UA. If caller wants to terminate all early dialogs instead of that with a particular UA, it needs to send CANCEL, not BYE. However, it is not illegal to send BYE in the Early state to terminate a specific early dialog according to the caller's intent. Moreover, until caller receives a final response and terminates the INVITE transaction, the caller MUST be prepared to establish a dialog by receiving a new response to the INVITE even though it had sent a CANCEL or BYE and terminated the dialog (see Appendix A).



(sr): indicates that both sending and reception are allowed.
Where (sr) is not indicated, response means send,
request means receive.

Figure 2. DSM for INVITE dialog usage (Callee)

Figure 2 represents callee's DSM for the INVITE dialog usage. The figure does not illustrate the state transition related to CANCEL request. CANCEL request does not cause a dialog state transition. However, the callee terminates the dialog and triggers

the dialog transition by sending 487 immediately after the reception of the CANCEL. Considering this, the behavior upon the reception of

the CANCEL request is further explained in Appendix C.

Following are UA's behaviors in each state.

Preparative (Pre): The Preparative state is a state until the early dialog is established by sending or receiving a provisional response with To tag after an ini-INVITE is sent or received. The dialog has not existed yet in the Preparative state. If UA sends or receives a 2xx response, the dialog state transit from the Preparative to the Moratorium state which is a substate of the Confirmed state.
In addition, if UA sends or receives a 3xx-6xx response the dialog state transit to the Morgue state which is a substate of the Terminated state. Sending an ACK for a 3xx-6xx response and retransmissions of 3xx-6xx are not expressed on the DSMs because they are sent by the INVITE transaction.

Early (Ear): The early dialog is established by sending or receiving a provisional response with To tag. The early dialog exists though the dialog does not exist in this state yet.
The dialog state transits from the Early to Moratorium state, a substate of the Confirmed state, by sending or receiving a 2xx response. In addition, the dialog state transits to the Morgue state, a substate of the Terminated state, by sending or receiving a 3xx-6xx response. Sending an ACK for a 3xx-6xx response and retransmissions of 3xx-6xx are not expressed on this DSM because they are automatically processed on transaction layer and don't influence the dialog state. UAC may send CANCEL in the Early state. UAC may send BYE (although it is not recommended). UAS may send a 1xx-6xx response. Sending or receiving of a CANCEL request does not have direct influences on dialog state. The UA's behavior upon the reception of the CANCEL request is further explained in Appendix C.

Confirmed (Con): Sending or receiving of a 2xx final response establishes a dialog. Dialog exists in this state. The Confirmed state transits to the Mortal state, a substate of the Terminated state, by sending or receiving a BYE request. The Confirmed state has two substates, the Moratorium and Established state, which are different in messages UAs are allowed to send.

Moratorium (Mora): The Moratorium state is a substate of the Confirmed state and inherits the behavior of the superstate. The Moratorium state transits to the Established state by sending or receiving an ACK request. UAC may send ACK and UAS may send a 2xx final response.

Established (Est): The Established state is a substate of the

Confirmed state and inherits the behavior of superstate. Both caller and callee may send various messages which influence a dialog. Caller supports the transmission of ACK in response to retransmission of a 2xx response to an ini-INVITE.

Terminated (Ter): The Terminated state is divided into two substates, the Mortal and Morgue states, to cover the behavior when a dialog is being terminated. In this state, UAs hold information about the dialog which is being terminated.

Mortal (Mort): Caller and callee enter the Mortal state by sending or receiving a BYE. UA MUST NOT send any new requests within the dialog because there is no dialog. (Here the new requests do not include ACK for 2xx and BYE for 401 or 407 as further explained in Appendix D below.)
In this state, only BYE or its response can be handled, and no other messages can be received. This addresses the case where BYE is sent by both a caller and a callee to exchange reports about the session when it is being terminated. Therefore the UA possesses dialog information for internal processing but the dialog shouldn't be externally visible. The UA stops managing its dialog state and changes it to the Morgue state, when the BYE transaction is terminated.

Morgue (Morg): The dialog no longer exists in this state.
Sending or receiving of signaling which influences a dialog is not performed. (A dialog is literally terminated.)
Caller and callee enter the Morgue state via the termination of the BYE or INVITE transaction.

3. Race conditions

This section details race condition between two SIP UAs, Alice and Bob. Alice (sip:alice@atlanta.example.com) and Bob (sip:bob@biloxi.example.com) are assumed to be SIP phones or SIP-enabled devices. Only significant signaling is illustrated. Dialog state transitions caused by sending or receiving of SIP messages as well as '*race*', which indicates race condition, are shown. (For abbreviations for the dialog state transitions, refer to Section 2.)
'*race*' indicates the moment when a race condition occurs.

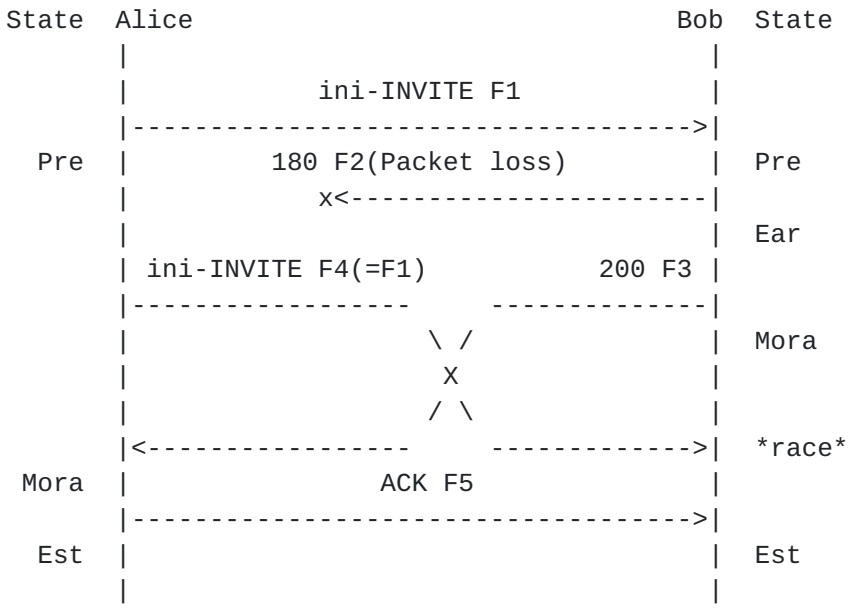
Examples of race conditions are shown below.

3.1 Receiving message in the Moratorium State

This section shows some examples of call flow in race condition when

receiving the message from other states in the Moratorium state.

3.1.1 Receiving Initial INVITE retransmission (Preparative state) in Moratorium state



This scenario illustrates the race condition which occurs when UAS receives a Preparative message in the Moratorium state. All provisional responses to the initial INVITE (ini-INVITE F1) are lost, and UAC retransmits an ini-INVITE (F4). At the same time as retransmission, UAS generates a 200 OK (F3) to the ini-INVITE and it terminates an INVITE server transaction, according to Section 13.3.1.4 of RFC 3261 [1].
However, it is reported that terminating an INVITE server transaction by 200 OK is a SIP bug. (http://bugs.sipit.net/, #769)
Therefore, the INVITE server transaction is not terminated at F3, and the F4 MUST be properly handled as a retransmission.
(UAs that do not deal with this bug still need to recognize the dialog relying on its From tag and Call-ID, and the retransmitted request relying on the CSeq header field value even though it does not match the transaction.)

In RFC 3261 [1], it is not specified whether UAS retransmits 200 to the retransmission of ini-INVITE. Considering the retransmission of 200 triggered by timer (TU keeps retransmitting 200 based on T1 and T2 until it receives an ACK), according to Section 13.3.1.4 of RFC 3261 [1], it seems unnecessary to retransmit 200 when the UAS receives the retransmission of ini-INVITE. (For implementation, it does not matter if the UAS sends the retransmission of 200, since the 200 does not cause any problem.)

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

/* 180 response is lost and does not reach Alice. */

F3 200 OK Bob -> Alice

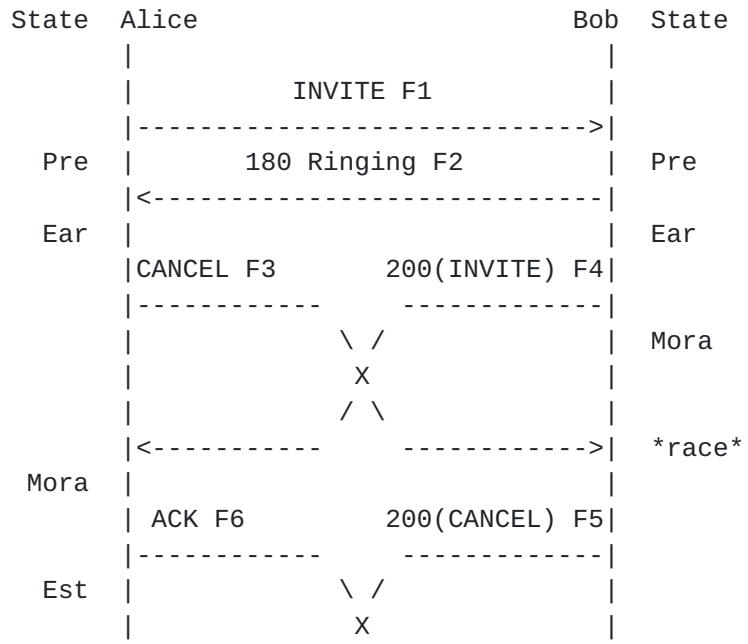
/* According to 13.3.1.4 of RFC 3261, an INVITE server transaction is terminated at this point. However, this has been reported as a SIP bug, and the UAS MUST correctly recognize the ini-INVITE (F4) as a retransmission. */

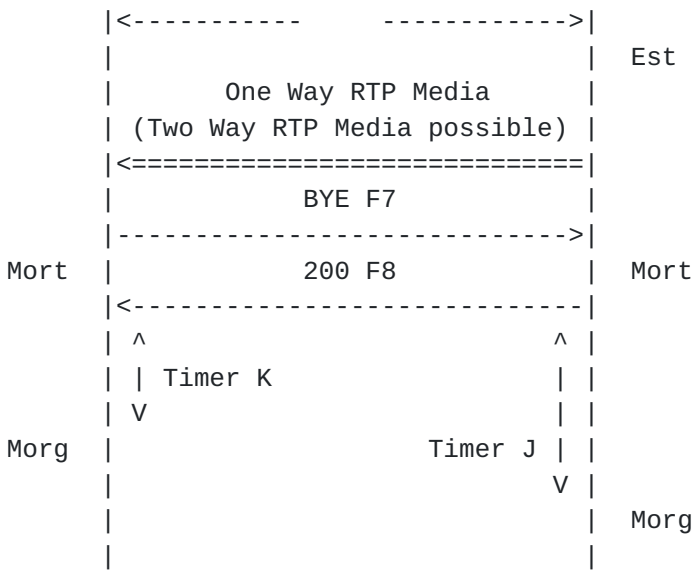
F4 INVITE (retransmission) Alice -> Bob

/* F4 is a retransmission of F1. They are exactly the same INVITE request. For UAs do not deal with the bug reported in #769 (an INVITE server transaction is terminated by 200 to INVITE), this request does not match the transaction as well as the dialog since it does not have a To tag.
However, Bob have to recognize the retransmitted INVITE correctly, without treating it as a new INVITE. */

F5 ACK Alice -> Bob

3.1.2 Receiving CANCEL (Early state) in Moratorium state





This scenario illustrates the race condition which occurs when UAS receives an Early message, CANCEL, in the Moratorium state. Alice sends a CANCEL and Bob sends a 200 OK response to the initial INVITE message at the same time. As described in the previous section, according to RFC 3261, an INVITE server transaction is supposed to be terminated by a 200 response, but this has been reported as a bug #769.

This section describes a case in which an INVITE server transaction is not terminated by a 200 response to the INVITE request. In this case, there is an INVITE transaction which the CANCEL request matches, so a 200 response is sent to the request. This 200 response simply means that the next hop received the CANCEL request (Successful CANCEL (200) does not mean an INVITE failure). When UAS does not deal with #769, UAC MAY receive a 481 response for CANCEL since there is no transaction which the CANCEL request matches. This 481 simply means that there is no matching INVITE server transaction and CANCEL is not sent to the next hop.
Regardless of the success/failure of the CANCEL, Alice checks the final response to INVITE, and if she receives 200 to the INVITE request she immediately sends a BYE and terminates the dialog. (Section 15, RFC 3261 [1])

From the time F1 is received by Bob until the time that F8 is sent by Bob, media may be flowing one way from Bob to Alice. From the time that an answer is received by Alice from Bob there is the possibility that media may flow from Alice to Bob as well. However, once Alice has decided to cancel the call, she presumably will not send media, so practically speaking the media stream will remain one way.

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 CANCEL Alice -> Bob

/* Alice sends a CANCEL in the Early state. */

F4 200 OK (INVITE) Bob -> Alice

/* Alice receives a 200 to INVITE (F1) in the Moratorium state.
Alice has the potential to send as well as receive media, but in
practice will not send because there is an intent to end the
call. */

F5 200 OK (CANCEL) Bob -> Alice

/* 200 to CANCEL simply means that the CANCEL was received.
The 200 response is sent, since this document deals with the bug
reported in #769. When an INVITE server transaction is terminated
as the procedure stated in RFC 3261, UAC MAY receive 481 response
instead of 200. */

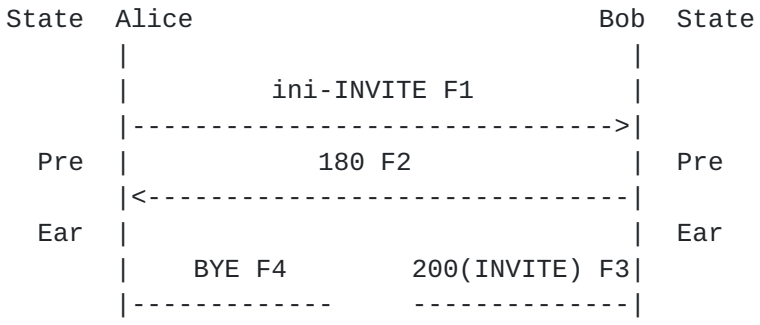
F6 ACK Alice -> Bob

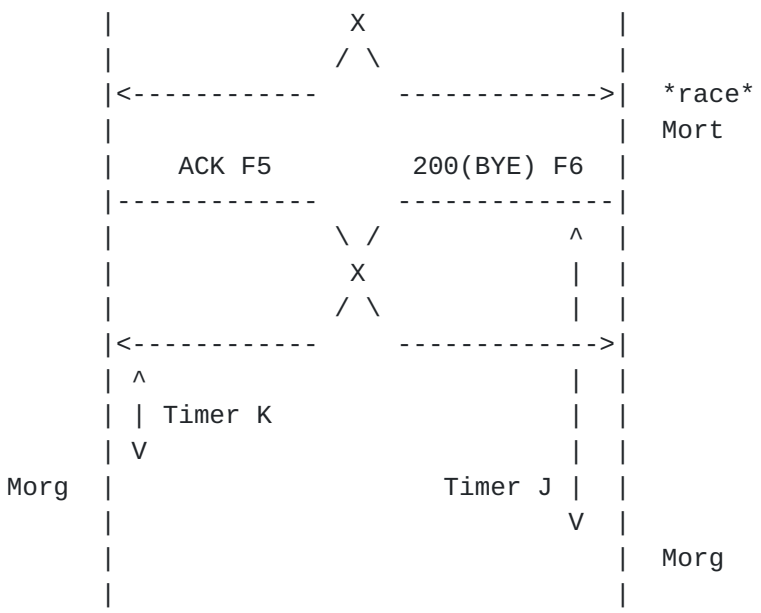
/* INVITE is successful, and the CANCEL becomes invalid. Bob
establishes RTP streams.
However, the next BYE request immediately terminates the dialog
and session. */

F7 BYE Alice -> Bob

F8 200 OK Bob -> Alice

3.1.3 Receiving BYE (Early state) in Moratorium state





This scenario illustrates the race condition which occurs when UAS receives an Early message, BYE, in the Moratorium state. Alice sends a BYE in the Early state and Bob sends a 200 OK response to the initial INVITE request at the same time. Bob receives the BYE in the Confirmed dialog state though Alice sent the request in the Early state (As explained in Section 2 and Appendix A, this behavior is NOT RECOMMENDED). When a proxy is performing forking, the BYE is only able to terminate the early dialog with a particular UA. If caller wants to terminate all early dialogs instead of that with a particular UA, it needs to send CANCEL, not BYE. However, it is not illegal to send BYE in the Early state to terminate a specific early dialog according to the caller's intent.

The BYE functions normally even if it is received after the INVITE transaction termination because BYE differs from CANCEL, and is sent not to the request but to the dialog. Alice gets into the Mortal state on sending the BYE request, and remains Mortal until the Timer K timeout occurs. In the Mortal state, UAC does not establish a session, even though it receives a 200 response to INVITE. Even so, the UAC sends an ACK to 200 for the completion of INVITE transaction. The ACK is always sent to complete the three-way handshake of INVITE transaction (Further explained in Appendix D below).

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK (ini-INVITE) Bob -> Alice

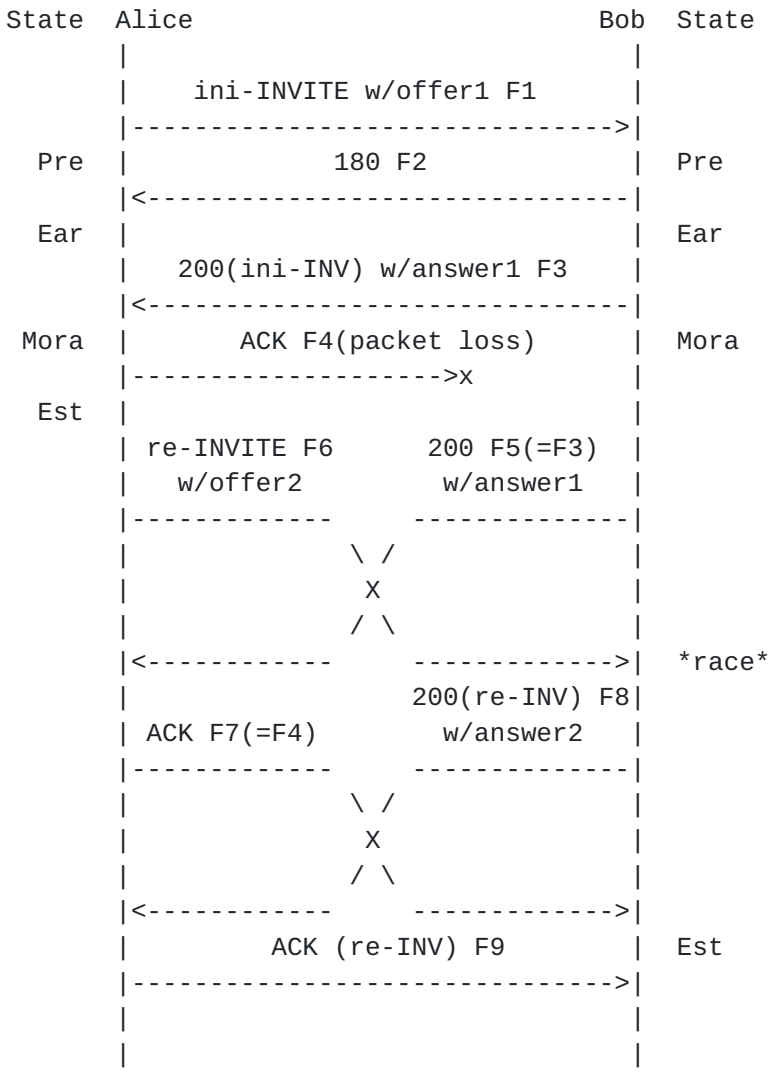
F4 BYE Alice -> Bob

/* Alice transits to the Mortal state upon sending BYE.
Therefore, after this, she does not begin a session even though
she receives a 200 response with an answer. */

F5 ACK Alice -> Bob

F6 200 OK (BYE) Bob -> Alice

3.1.4 Receiving re-INVITE (Established state) in Moratorium state (case 1)



This scenario illustrates the race condition which occurs when UAS

receives re-INVITE request sent from the Established state, in the Moratorium state.

UAS receives a re-INVITE (w/offer2) before receiving an ACK for ini-INVITE (w/offer1). UAS sends a 200 OK (w/answer2) to the re-INVITE (F8) because it has sent a 200 OK (w/answer1) to the ini-INVITE (F3, F5) and the dialog has already been established. (Because F5 is a retransmission of F3, SDP negotiation is not performed here.)

If a 200 OK to the ini-INVITE has an offer and the answer is in the ACK, it is recommended that UA return a 491 to the re-INVITE (refer to 3.1.5). (Note: 500 with Retry-After header may be returned, if the 491 response is understood to indicate request collision. However, 491 is recommended here because 500 applies to so many cases that it is difficult to determine what the real problem was.) As it can be seen in Section 3.3.2 below, the 491 response seems to be closely related to session establishment, even in cases other than INVITE cross-over. This example recommends 200 be sent instead of 491 because it does not have influence on session. However, a 491 response can also lead to the same outcome, so the either response can be used.

Moreover, if UAS doesn't receive an ACK for a long time, it should send a BYE and terminate the dialog. Note that ACK F7 has the same CSeq number as ini-INVITE F1 (See Section 13.2.2.4 of RFC 3261). The UA should not reject or drop the ACK on grounds of CSeq number.

Note: There is a hint for implementation to avoid the race conditions of this type. That is for the caller to delay sending re-INVITE F6 for some period of time (2 seconds, perhaps), from which the caller is reasonably able to assume that its ACK has been received. Implementors can decouple the actions of the user (e.g. pressing the hold button) from the actions of the protocol (the sending of re-INVITE F6), so the UA can behave as such. In this case, it is dependent on the implementor's choice how long it waits. In most cases, such implementation may be useful to prevent a type of race condition shown in this section.

Message Details

F1 INVITE Alice -> Bob

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
```

```
   Call-ID: 384827629822018851l@atlanta.example.com
```

```
   CSeq: 1 INVITE
   Contact: <sip:alice@client.atlanta.example.com;transport=udp>
   Content-Type: application/sdp
   Content-Length: 137

   v=0
   o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com
   s=-
   c=IN IP4 192.0.2.101
   t=0 0
   m=audio 49172 RTP/AVP 0
   a=rtpmap:0 PCMU/8000

   /* Detailed messages are shown for the sequence to illustrate offer
      and answer examples.  */


   F2 180 Ringing Bob -> Alice

   SIP/2.0 180 Ringing
   Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
    ;received=192.0.2.101
   From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 INVITE
   Contact: <sip:bob@client.biloxi.example.com;transport=udp>
   Content-Length: 0


   F3 200 OK Bob -> Alice

   SIP/2.0 200 OK
   Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
    ;received=192.0.2.101
   From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 INVITE
   Contact: <sip:bob@client.biloxi.example.com;transport=udp>
   Content-Type: application/sdp
   Content-Length: 133

   v=0
   o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com
   s=-
   c=IN IP4 192.0.2.201
   t=0 0
   m=audio 3456 RTP/AVP 0
```

```
   a=rtpmap:0 PCMU/8000
```

F4 ACK Alice -> Bob

```
ACK sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bKnashd8
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 ACK
Content-Length: 0

/* ACK request is lost.  */
```

F5(=F3) 200 OK Bob -> Alice (retransmission)

```
/* UAS retransmits a 200 OK to the ini-INVITE since it has not
   received an ACK.  */
```

F6 re-INVITE Alice -> Bob

```
INVITE sip:sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf91
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Content-Length: 147

v=0
o=alice 2890844526 2890844527 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=sendonly
```

F7(=F4) ACK Alice -> Bob (retransmission)

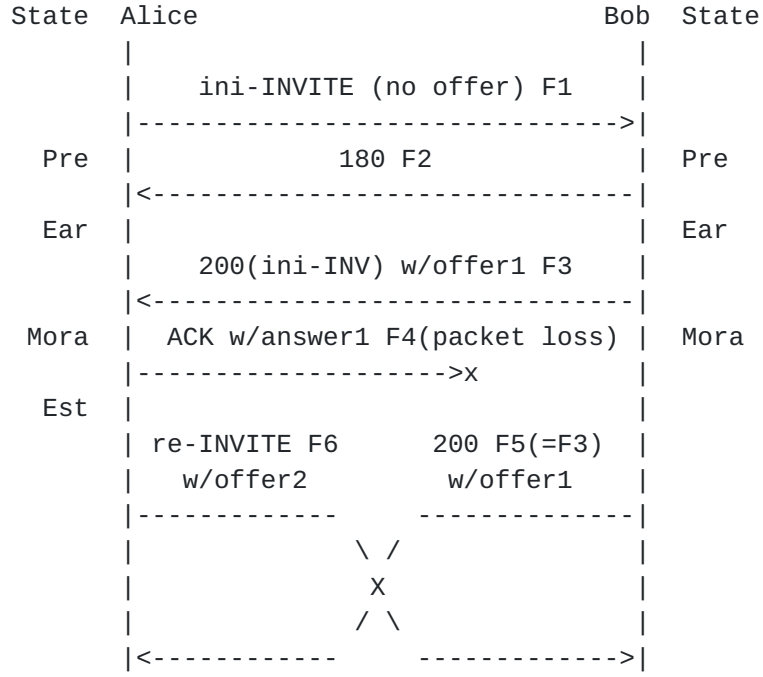
F8 200 OK (re-INVITE) Bob -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf91
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
```

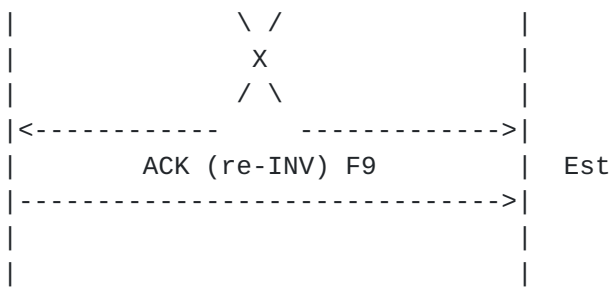
```
   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
```

```
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Content-Length: 143

v=0
o=bob 2890844527 2890844528 IN IP4 client.biloxi.example.com
s=-
c=IN IP4 192.0.2.201
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=recvonly


F9 ACK (re-INVITE) Alice -> Bob

ACK sip:sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK230f21
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 ACK
Content-Length: 0
```

3.1.5 Receiving re-INVITE (Established state) in Moratorium state (case 2)



| ACK F7(=F4)
491(re-INV) F8|



This scenario is basically the same as that of Section 3.1.4, but differs in sending an offer in 200 and an answer in ACK. Different to the previous case, the offer in the 200 (F3) and the offer in the re-INVITE (F6) collide with each other.

Bob sends 491 to re-INVITE (F6) since he is not able to properly handle a new request until he receives an answer. (Note: 500 with Retry-After header may be returned, if the 491 response is understood to indicate request collision. However, 491 is recommended here because 500 applies to so many cases that it is difficult to determine what the real problem was.)
Even if F6 is UPDATE with offer, it will reach the same result.

Note: As noted in Section 3.1.4, it may be useful for the caller to delay sending re-INVITE F6 for some period of time (2 seconds, perhaps), from which the caller is reasonably able to assume that its ACK has been received, to prevent a type of race condition shown in this section.

Message Details

F1 INVITE Alice -> Bob

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=udp>
Content-Length: 0

/* The request does not contain an offer.  Detailed messages are
   shown for the sequence to illustrate offer and answer
   examples.  */
```

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
 ;received=192.0.2.101
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:bob@client.biloxi.example.com;transport=udp>
Content-Type: application/sdp
Content-Length: 133

v=0
o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com
s=-
c=IN IP4 192.0.2.201
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000

/* An offer is made in 200.  */
```

F4 ACK Alice -> Bob

```
ACK sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bKnashd8
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 ACK
Content-Type: application/sdp
Content-Length: 137

v=0
o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

/* The request contains an answer, but the request is lost.  */
```

F5(=F3) 200 OK Bob -> Alice (retransmission)

```
   /* UAS retransmits a 200 OK to the ini-INVITE since it has not
```

received an ACK. */

F6 re-INVITE Alice -> Bob

```
INVITE sip:sip:bob@client.biloxi.example.com SIP/2.0

Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf91
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Content-Length: 147

v=0
o=alice 2890844526 2890844527 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=sendonly
```

/* The request contains an offer. */

F7(=F4) ACK Alice -> Bob (retransmission)

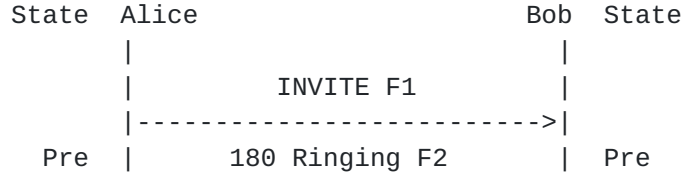
/* A retransmission triggered by the reception of a retransmitted
200. */

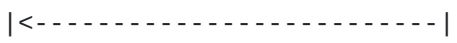
F8 491 (re-INVITE) Bob -> Alice

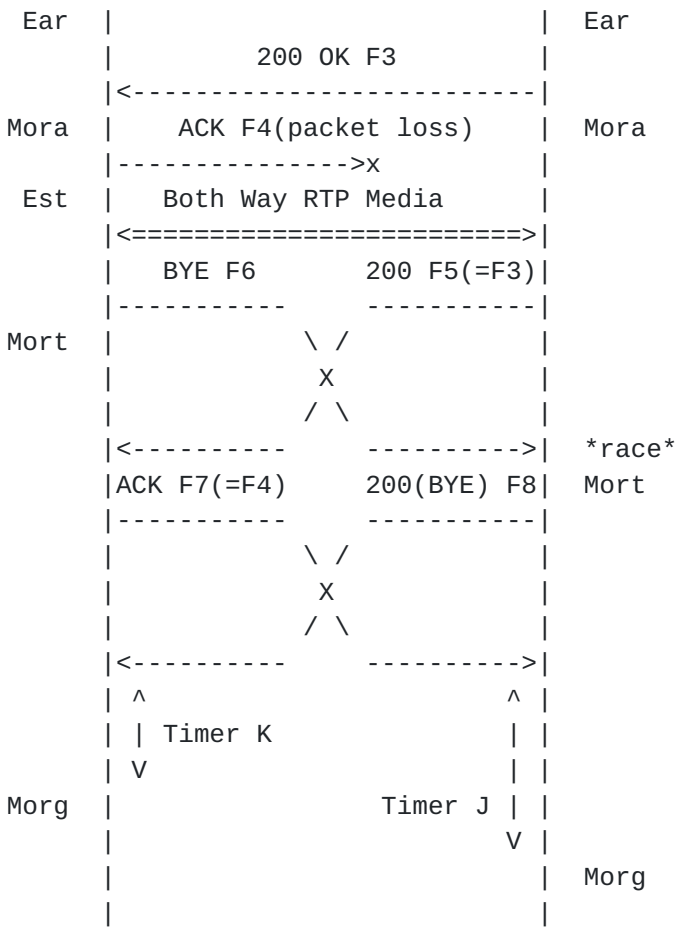
/* Bob sends 491 (Request Pending), since Bob has a pending
offer. */

F9 ACK (re-INVITE) Alice -> Bob

3.1.6 Receiving BYE (Established state) in Moratorium state







This scenario illustrates the race condition which occurs when UAS receives an Established message, BYE, in the Moratorium state. An ACK request for a 200 OK response is lost (or delayed). Immediately after Bob retransmits the 200 OK to ini-INVITE, and Alice sends a BYE request at the same time. Depending on the implementation of a SIP UA, Alice may start a session again by the reception of the retransmitted 200 OK with SDP since she has already terminated a session by sending a BYE. In that case, when UAC receives a retransmitted 200 OK after sending a BYE, a session should not be started again since the session which is not associated with dialog still remains. Moreover, in the case where UAS sends an offer in a 200 OK , UAS should not start a session again for the same reason if UAS receives a retransmitted ACK after receiving a BYE.

Note: As noted in Section 3.1.4, there is a hint for implementation to avoid the race conditions of this type. That is for the caller to delay sending BYE F6 for some period of time (2 seconds, perhaps), from which the caller is reasonably able to assume that its ACK has been received. Implementors can decouple the actions of the user (e.g. hanging up) from the actions of the protocol (the sending of

BYE F6), so the UA can behave as such. In this case, it is dependent on the implementor's choice how long it waits. In most cases, such

implementation may be useful to prevent a type of race condition shown in this section.

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

/* ACK request is lost. */

F5(=F3) 200 OK Bob -> Alice

/* UAS retransmits a 200 OK to the ini-INVITE since it has not received an ACK. */

F6 BYE Alice -> Bob

/* Bob retransmits a 200 OK and Alice sends a BYE at the same time. Alice transits to the Mortal state, so she does not begin a session after this even though she receives a 200 response to the re-INVITE. */

F7(=F4) ACK Alice -> Bob

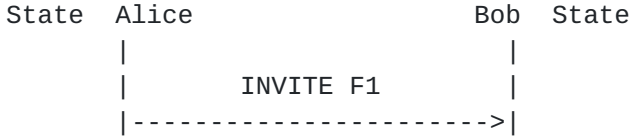
F8 200 OK (BYE) Bob -> Alice

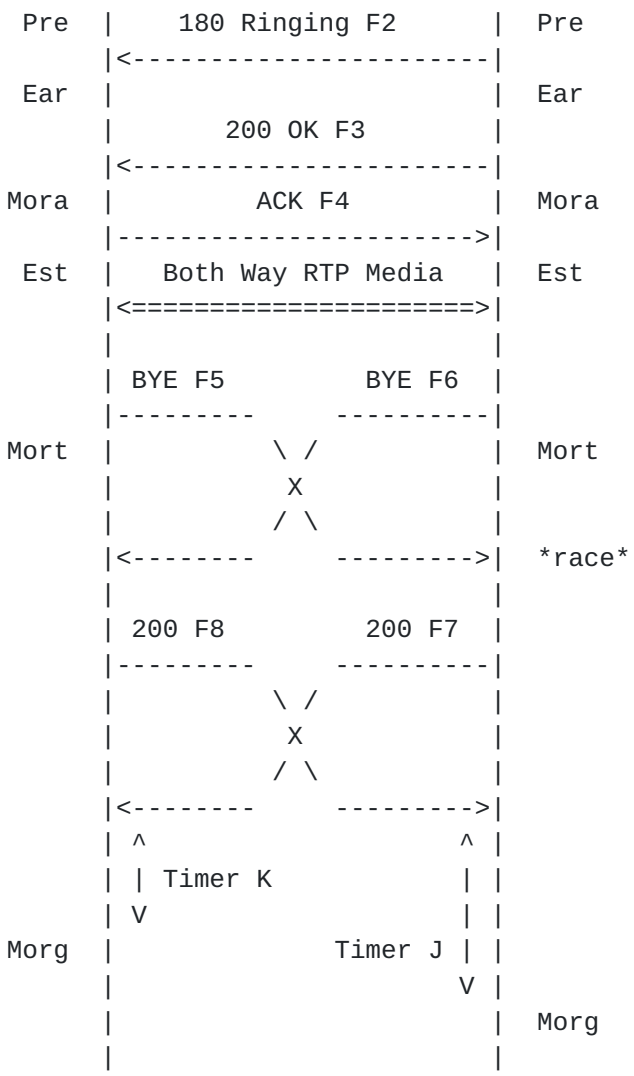
/* Bob sends a 200 OK to the BYE. */

3.2 Receiving message in the Mortal State

This section shows some examples of call flow in race condition when receiving the message from other states in the Mortal state.

3.2.1 Receiving BYE (Establish state) in Mortal state





This scenario illustrates the race condition which occurs when UAS receives an Established message, BYE, in the Mortal state. Alice and Bob send a BYE at the same time. A dialog and session are ended shortly after a BYE request is passed to a client transaction. As shown in Section 2, UA remains in the Mortal state.

UAs in the Mortal state return error responses to the requests that operate dialog or session, such as re-INVITE, UPDATE, or REFER. However, UA shall return 200 OK to the BYE taking the use case into consideration where BYE request is sent by both a caller and a callee to exchange reports about the session when it is being terminated. (Since the dialogue and the session both terminate when a BYE is sent, the choice of sending 200 or an error response upon receiving BYE in the Mortal state does not affect the resulting termination. Therefore, even though this example uses a 200 response, other responses can also be used.)

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

F5 BYE Alice -> Bob

/* The session is terminated at the moment Alice sends a BYE.
The dialog still exists then, but it is certain to be terminated
in a short period of time. The dialog is completely
terminated when the timeout of the BYE request occurs. */

F6 BYE Bob -> Alice

/* Bob has also transmitted a BYE simultaneously with Alice.
Bob terminates the session and the dialog. */

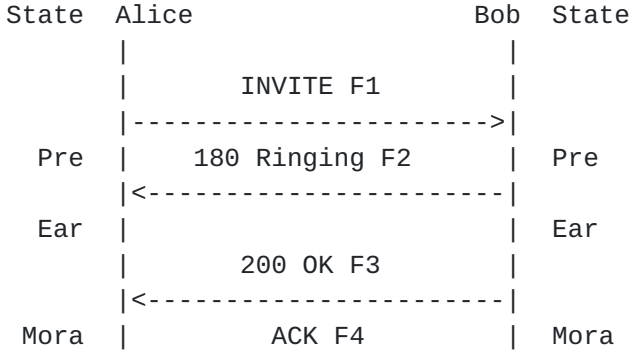
F7 200 OK Bob -> Alice

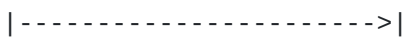
/* Since the dialog is in the Moratorium state, Bob responds with
a 200 to the BYE request. */

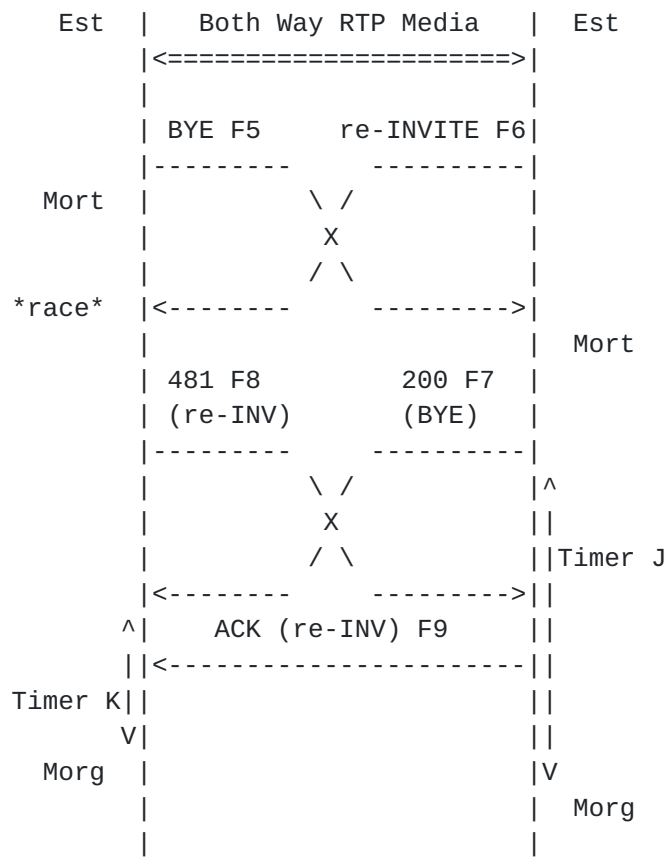
F8 200 OK Alice -> Bob

/* Since Alice has transited from the Established state to the Mortal
state by sending BYE, Alice responds with a 200 to the BYE
request. */

3.2.2 Receiving re-INVITE (Establish state) in Mortal state







This scenario illustrates the race condition which occurs when UAS receives an Established message, re-INVITE, in the Mortal state. Bob sends a re-INVITE, and Alice sends a BYE at the same time. The re-INVITE is responded by a 481, since the TU of Alice has transited from the Established state to the Mortal state by sending BYE. Bob sends ACK for the 481 response, because the ACK for error responses is handled by the transaction layer and at the point of receiving the 481 the INVITE client transaction still remains (even though the dialog has been terminated).

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

F5 BYE Alice -> Bob

```
   /* Alice sends a BYE and terminates the session, and transits from
```

the Established state to the Mortal state. */

F6 re-INVITE Bob -> Alice

/* Alice sends a BYE, and Bob sends a re-INVITE at the same time.
The dialog state transits to the Mortal state at the moment
Alice sends the BYE, but Bob does not know it until he receives
the BYE. Therefore, the dialog is in the Terminated state from
Alice's point of view, but it is the Confirmed state
from Bob's point of view. A race condition occurs. */

F7 200 OK (BYE) Bob -> Alice

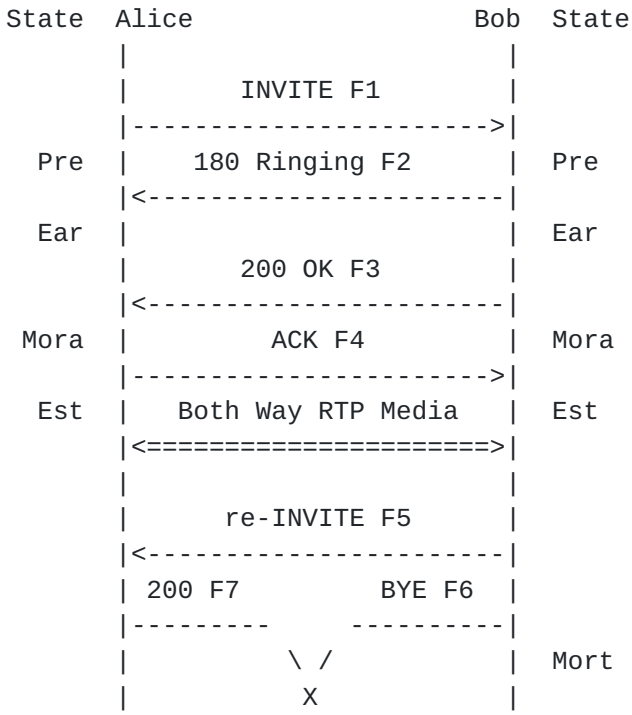
F8 481 Call/Transaction Does Not Exist (re-INVITE) Alice -> Bob

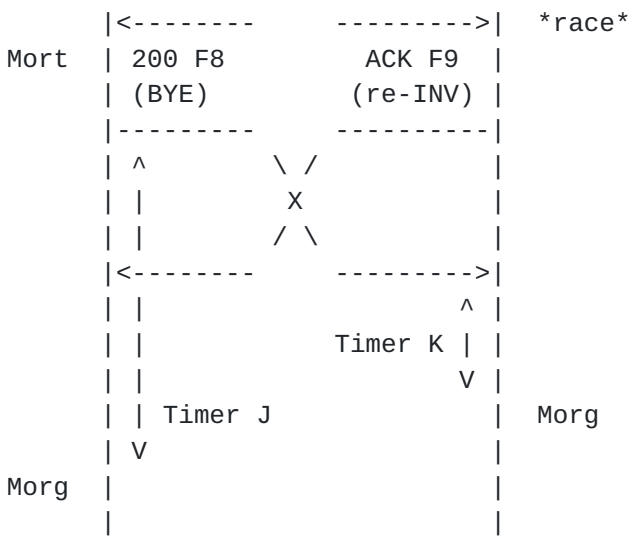
/* Since Alice is in the Mortal state, she responds with a 481 to the
re-INVITE. */

F9 ACK (re-INVITE) Bob -> Alice

/* ACK for an error response is handled by Bob's INVITE client
transaction. */

3.2.3 Receiving 200OK for re-INVITE (Established state) in Mortal state





This scenario illustrates the race condition which occurs when UAS receives an Established message, 200 to re-INVITE, in the Mortal state. Bob sends a BYE immediately after sending a re-INVITE. (A user is not conscious that refresher sends re-INVITE automatically. For example, in the case of a telephone application, it is possible that a user places a receiver immediately after refresher.)
Bob sends ACK for a 200 response to INVITE in the Mortal state, so that he completes the INVITE transaction.

Note: As noted in Section 3.1.4, there is a hint for implementation to avoid the race conditions of this type. That is for the UAC to delay sending BYE F6 until re-INVITE F5 transaction completes. Implementors can decouple the actions of the user (e.g. hanging up) from the actions of the protocol (the sending of BYE F6), so the UA can behave as such. In this case, it is dependent on the implementor's choice how long it waits. In most cases, such implementation may be useful to prevent a type of race condition shown in this section.

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

F5 re-INVITE Bob -> Alice

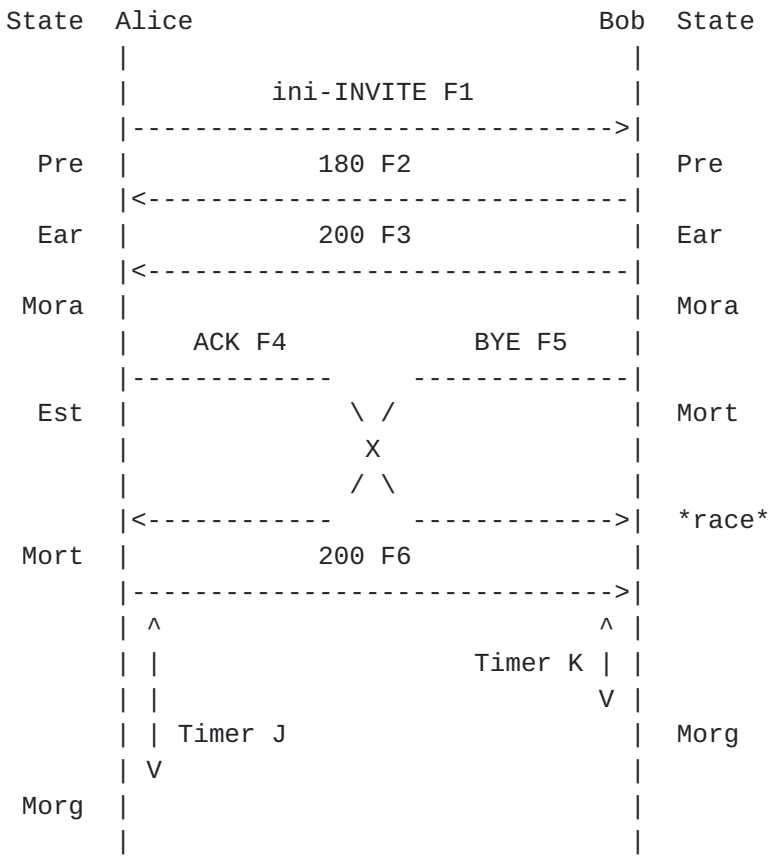
```
   INVITE sip:alice@client.atlanta.example.com SIP/2.0
```

```
   Via: SIP/2.0/UDP client.biloxi.example.com:5060;branch=z9hG4bKnashd7
   Session-Expires: 300;refresher=uac
   Supported: timer
   Max-Forwards: 70
   From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 INVITE
   Content-Length: 0

   /* Some detailed messages are shown for the sequence to illustrate
      that the re-INVITE is handled in the usual manner in the Mortal
      state.  */


   F6 BYE Bob -> Alice

   /* Bob sends BYE immediately after sending the re-INVITE.
      Bob terminates the session and transits from the Established
      state to the Mortal state.  */


   F7 200 OK (re-INVITE) Alice -> Bob

   SIP/2.0 200 OK
   Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bKnashd7
    ;received=192.0.2.201
   Require: timer
   Session-Expires: 300;refresher=uac
   From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 INVITE
   Content-Length: 0

   /* Bob sends BYE, and Alice responds with a 200 OK to the re-INVITE.
      A race condition occurs.  */


   F8 200 OK (BYE) Alice -> Bob

   F9 ACK (re-INVITE) Bob -> Alice

   ACK sip:alice@client.atlanta.example.com SIP/2.0
   Via: SIP/2.0/UDP client.biloxi.example.com:5060;branch=z9hG4bK74b44
   Max-Forwards: 70
   From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   Call-ID: 3848276298220188511@atlanta.example.com
```

```
   CSeq: 2 ACK
```

```
   Content-Length: 0

   /* Bob sends ACK in the Mortal state to complete the three-way
      handshake of the INVITE transaction.  */
```

3.2.4 Receiving ACK (Moratorium state) in Mortal state



This scenario illustrates the race condition which occurs when UAS receives an Established message, ACK to 200, in the Mortal state. Alice sends an ACK and Bob sends a BYE at the same time. When the offer is in a 2xx, and the answer is in an ACK, this example is in a race condition. The session is not started by receiving the ACK because Bob has already terminated the session by sending the BYE. The answer in the ACK request is just ignored.

Note: As noted in Section 3.1.4, there is a hint for implementation to avoid the race conditions of this type. Implementors can decouple the actions of the user (e.g. hanging up) from the actions of the protocol (the sending of BYE F5), so the UA can behave as such. In this case, it is dependent on the implementor's choice how long it

waits. In most cases, such implementation may be useful to prevent a type of race condition shown in this section.

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

/* RTP streams are established between Alice and Bob */

F5 BYE Alice -> Bob

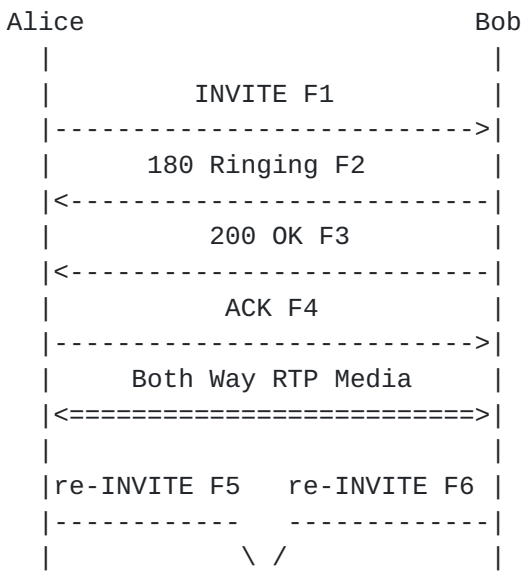
F6 200 OK Bob -> Alice

/* Alice sends a BYE and terminates the session and dialog. */

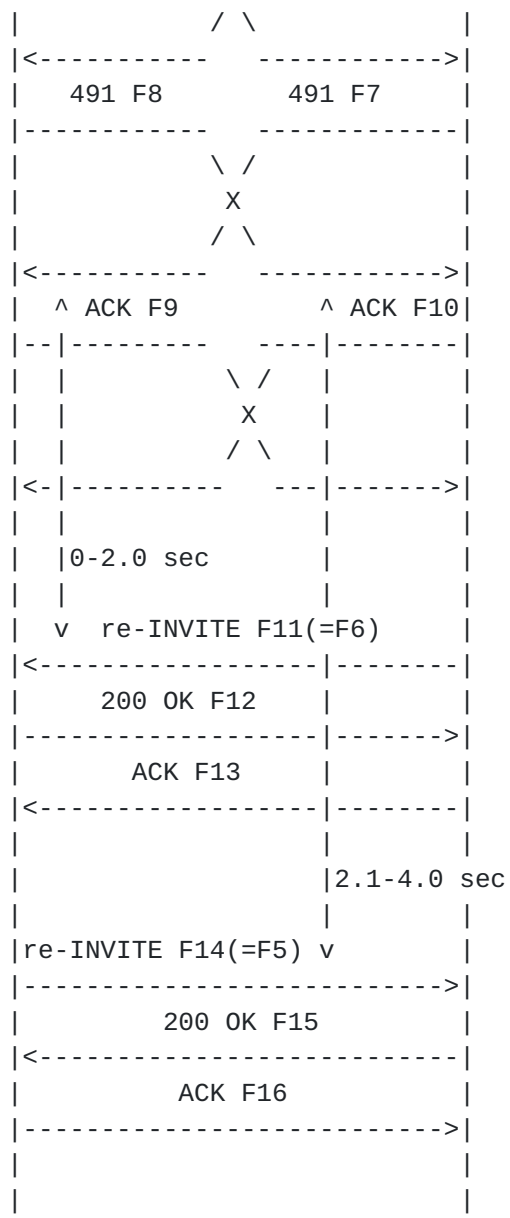
3.3 Other race conditions

This section shows the examples in race condition that are not directly related to the dialog state transition. In SIP, processing functions are deployed in three layers, dialog, session, and transaction. There are related each other, but have to be treated separately. Section 17 of RFC 3261 [1] details the processing on transactions. This draft has tried so far to clarify the processing on dialogs. This section explains race conditions which are related to sessions established by SIP.

3.3.1 Re-INVITE crossover



```
     |             X              |
```



In this scenario, Alice and Bob send re-INVITE at the same time. When two re-INVITEs cross in the same dialog, they resend re-INVITEs after different interval for each, according to Section 14.1, of RFC 3261 [1]. When Alice sends the re-INVITE and it crosses, the re-INVITE will be sent again after 2.1-4.0 seconds because she owns the Call-ID (she generated it). Bob will send an INVITE again after 0.0-2.0 seconds, because Bob isn't the owner of the Call-ID. Therefore, each user agent must remember whether it has generated the Call-ID of the dialog or not, in case an INVITE may cross with another INVITE.

In this example, Alice's re-INVITE is for session modification and

Bob's re-INVITE is for session refresh. In this case, after the 491 responses, Bob retransmits the re-INVITE for session refresh earlier

than Alice. If Alice was to retransmit her re-INVITE (that is, if she was not the owner of Call-ID), the request would refresh and modify the session at the same time. Then Bob would know that he would not need to retransmit his re-INVITE to refresh the session.

In another instance where two re-INVITEs for session modification, cross over, retransmitting the same re-INVITE again after 491 by the Call-ID owner (the UA which retransmits its re-INVITE after the other UA) may result in a behavior different from what the user originally intended to, so the UA needs to decide if the retransmission of the re-INVITE is necessary.
(For example, when a call hold and an addition of video media cross over, mere retransmission of the re-INVITE at the firing of the timer may result in the situation where the video is transmitted immediately after the holding of the audio. This behavior is probably not intended by the users.)

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

F5 re-INVITE Alice -> Bob

```
INVITE sip:sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Content-Length: 147

v=0
o=alice 2890844526 2890844527 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=sendonly
```

/* Some detailed messages are shown for the sequence to illustrate

```
      what sort of INVITE requests crossed over each other.  */
```

F6 re-INVITE Bob -> Alice

```
INVITE sip:alice@client.atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP client.biloxi.example.com:5060;branch=z9hG4bKnashd7
Session-Expires: 300;refresher=uac
Supported: timer
Max-Forwards: 70
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Content-Length: 0

/* A re-INVITE request for a session refresh and that for a call hold
   are sent at the same time.  */
```

F7 491 Request Pending Bob -> Alice

```
/* Since a re-INVITE is in progress, a 491 response is returned.  */
```

F8 491 Request Pending Alice -> Bob

F9 ACK (INVITE) Alice -> Bob

F10 ACK (INVITE) Bob -> Alice

F11 re-INVITE Bob -> Alice

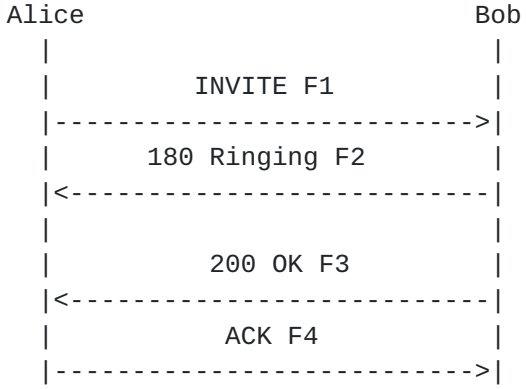
```
INVITE sip:alice@client.atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP client.biloxi.example.com:5060;branch=z9hG4bKnashd71
Session-Expires: 300;refresher=uac
Supported: timer
Max-Forwards: 70
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Content-Type: application/sdp
Content-Length: 133

v=0
o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com
s=-
c=IN IP4 192.0.2.201
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

```
   /* Since Bob is not the owner of the Call-ID, he sends a re-INVITE
```

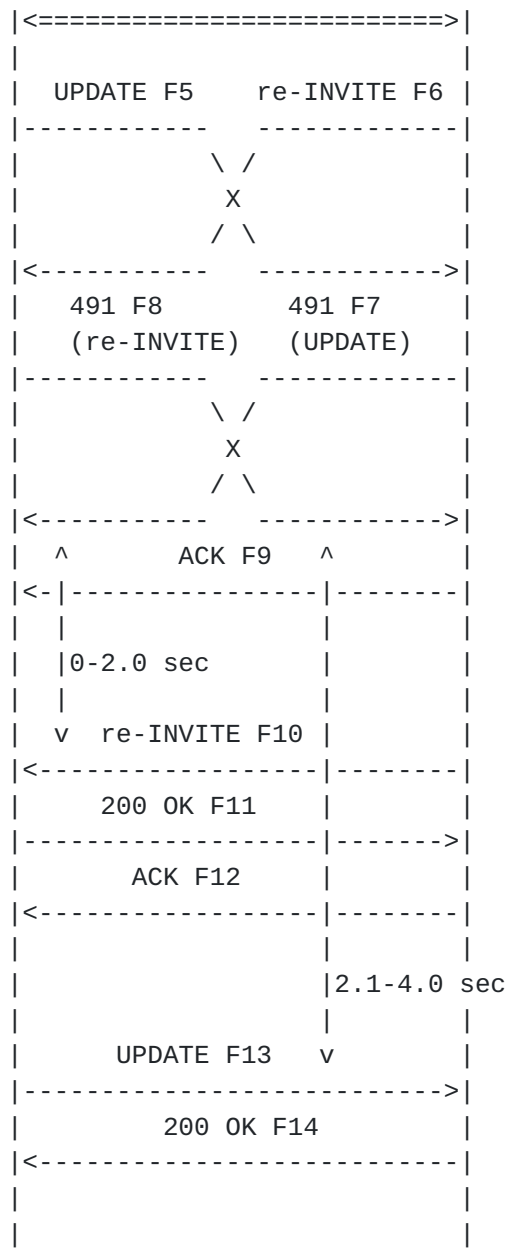
```
      again after 0.0-2.0 seconds.  */
```

```
   F12 200 OK Alice -> Bob

   F13 ACK Bob -> Alice

   F14 re-INVITE Alice -> Bob

   INVITE sip:sip:bob@client.biloxi.example.com SIP/2.0
   Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf91
   Max-Forwards: 70
   From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 3 INVITE
   Content-Length: 147

   v=0
   o=alice 2890844526 2890844527 IN IP4 client.atlanta.example.com
   s=-
   c=IN IP4 192.0.2.101
   t=0 0
   m=audio 49172 RTP/AVP 0
   a=rtpmap:0 PCMU/8000
   a=sendonly

   /* Since Alice is the owner of the Call-ID, Alice sends a re-INVITE
      again after 2.1-4.0 seconds.  */


   F15 200 OK Bob -> Alice

   F16 ACK Alice -> Bob
```

3.3.2 UPDATE and re-INVITE crossover



```
     |     Both Way RTP Media     |
```



In this scenario, the UPDATE contains a SDP offer, therefore the UPDATE and re-INVITE are responded with 491 as in the case of "re-INVITE crossover". When an UPDATE for refresher which doesn't contain a session description and a re-INVITE crossed each other, both requests succeed with 200 (491 means that UA have a pending request). Moreover, the same is equally true for UPDATE crossover. In the former case where either UPDATE contains a session description the requests fail with 491, and in the latter cases succeed with 200.

Editor's Note:
A 491 response is sent because SDP offer is pending, and 491 is an

error which is related to matters that behave on the session
established by SIP.

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

F5 UPDATE Alice -> Bob

```
UPDATE sip:sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 UPDATE
Content-Length: 147

v=0
o=alice 2890844526 2890844527 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=sendonly

/* Some detailed messages are shown for the sequence to illustrate
   the example messages crossed over each other.  */
```

F6 re-INVITE Bob -> Alice

```
INVITE sip:alice@client.atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP client.biloxi.example.com:5060;branch=z9hG4bKnashd7
Session-Expires: 300;refresher=uac
Supported: timer
Max-Forwards: 70
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Content-Length: 0

/* A case where a re-INVITE for a session refresh and a UPDATE for a
   call hold are sent at the same time.  */
```

F7 491 Request Pending (UPDATE) Bob -> Alice

/* Since a re-INVITE is in process, a 491 response are returned. */

F8 491 Request Pending (re-INVITE) Alice -> Bob

F9 ACK (re-INVITE) Alice -> Bob

F10 re-INVITE Bob -> Alice

```
INVITE sip:alice@client.atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP client.biloxi.example.com:5060;branch=z9hG4bKnashd71
Session-Expires: 300;refresher=uac
Supported: timer
Max-Forwards: 70
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Content-Type: application/sdp
Content-Length: 133

v=0
o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com
s=-
c=IN IP4 192.0.2.201
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

/* Since Bob is not the owner of Call-ID, Bob sends an INVITE again after 0.0-2.0 seconds. */

F11 200 OK Alice -> Bob

F12 ACK Bob -> Alice

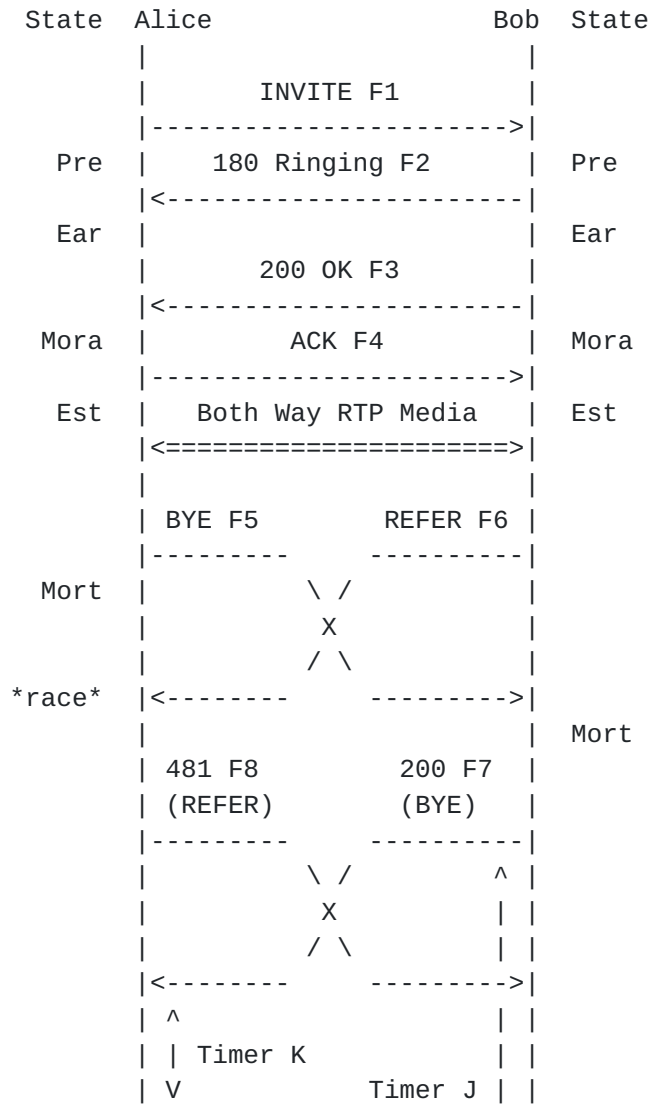
F13 UPDATE Alice -> Bob

```
UPDATE sip:sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf91
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 3 UPDATE
Content-Length: 147
```

```
v=0
```

```
   o=alice 2890844526 2890844527 IN IP4 client.atlanta.example.com
   s=-
   c=IN IP4 192.0.2.101
   t=0 0
   m=audio 49172 RTP/AVP 0
   a=rtpmap:0 PCMU/8000
   a=sendonly

   /* Since Alice is the owner of the Call-ID, Alice sends the UPDATE
      again after 2.1-4.0 seconds.  */

   F14 200 OK Bob -> Alice
```

3.3.3 Receiving REFER (Establish state) in Mortal state



Morg
V
Morg

This scenario illustrates the race condition which occurs when UAS receives an Established message, REFER, in the Mortal state.
Bob sends a REFER, and Alice sends a BYE at the same time. Bob send the REFER in the same dialog. Alice's dialog state moves to the Mortal state at the point of sending BYE. In the Mortal state, UA possesses dialog information for internal process but dialog shouldn't exist outwardly. Therefore, UA sends an error response to a REFER which is transmitted as a mid-dialog request. So, Alice in the Mortal state sends an error response to the REFER.
However, Bob has already started the SUBSCRIBE usage with REFER, so the dialog continues until the SUBSCRIBE usage terminates, even though the INVITE dialog usage terminates by receiving BYE. Bob's behavior in this case needs to follow the procedure in the dialog usage draft [6]. (For handling of dialogs with multiple usages see the dialog usage draft [6].)

Message Details

F1 INVITE Alice -> Bob

F2 180 Ringing Bob -> Alice

F3 200 OK Bob -> Alice

F4 ACK Alice -> Bob

F5 BYE Alice -> Bob

/* Alice sends a BYE request and terminates the session, and transits from the Confirmed state to Terminated state. */

F6 REFER Bob -> Alice

/* Alice sends a BYE, and Bob sends a REFER at the same time. Bob sends the REFER on the INVITE dialog. The dialog state transits to the Mortal state at the moment Alice sends the BYE, but Bob doesn't know it until he receives the BYE. A race condition occurs. */

F7 200 OK (BYE) Bob -> Alice

F8 481 Call/Transaction Does Not Exist (REFER) Alice -> Bob

/* Alice in the Mortal state sends a 481 to the REFER. */

4. IANA Considerations

This document has no actions for IANA.

5. Security Considerations

This document contains clarifications of behavior specified in RFCs 3261 [1], 3264 [2], and 3515 [4]. The security considerations of those documents continue to apply after the application of these clarifications.

6. Acknowledgements

The authors would like to thank Robert Sparks, Dean Willis, Cullen Jennings, James M. Polk, Gonzalo Camarillo, Kenichi Ogami, Akihiro Shimizu, Mayumi Munakata, Yasunori Inagaki, Tadaatsu Kidokoro, Kenichi Hiragi, Dale Worley, Vijay K. Gurbani and Anders Kristensen for their comments on this document.

7. References

7.1 Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with the Session Description Protocol (SDP)", RFC 3264, June 2002.

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

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

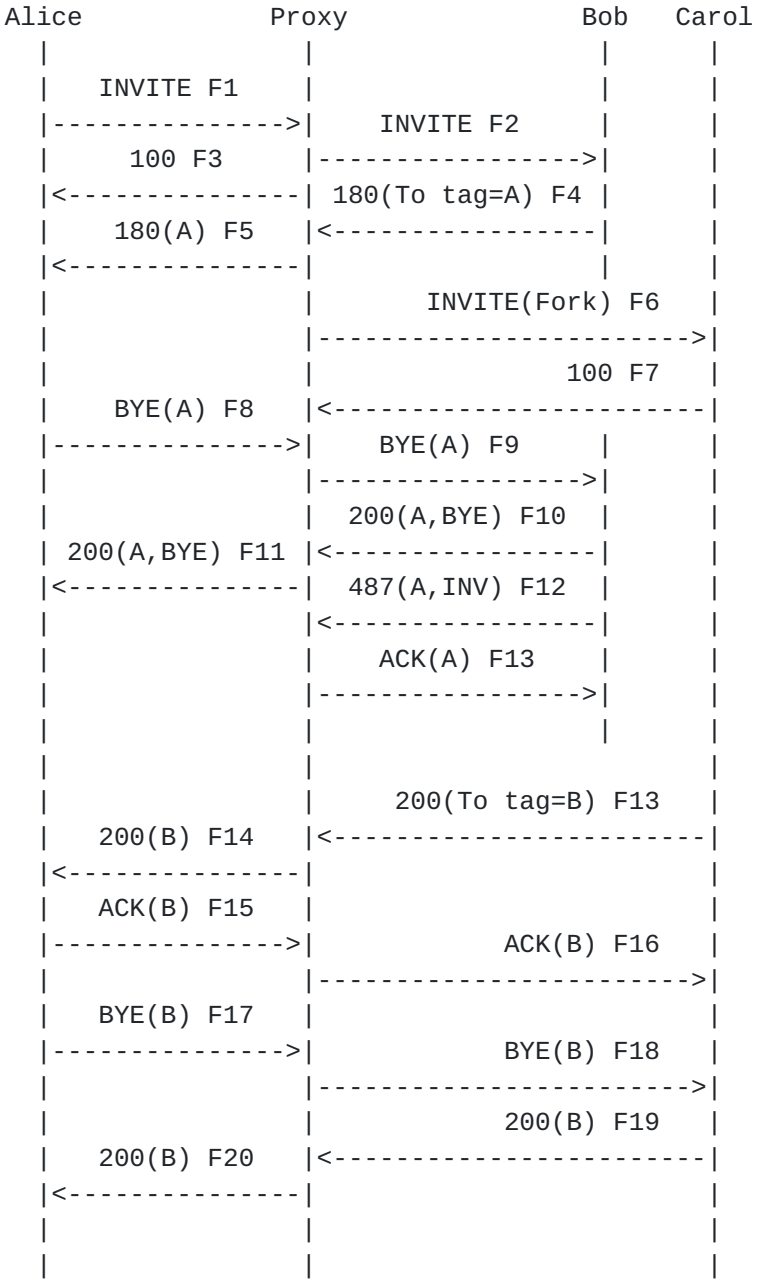
[5] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in the Session Initiation Protocol (SIP)", RFC 3262, June 2002.

7.2 Informative References

[6] Sparks, R., "Multiple Dialog Usages in the Session Initiation Protocol", draft-ietf-sipping-dialogusage-06 (work in progress), February 16, 2007.

Appendix A - BYE on the Early Dialog

This section, related to Section 3.1.3, explains why BYE is not recommended in the Early state, illustrating the case in which BYE in the early dialog triggers confusion.



Care is advised in sending BYE in the Early state when forking by a proxy is expected. In this example, the BYE request progresses normally, and it succeeds in correctly terminating the dialog with Bob. After Bob terminates the dialog by receiving the BYE, he sends 487 to the ini-INVITE. According to Section 15.1.2 of RFC 3261 [1],

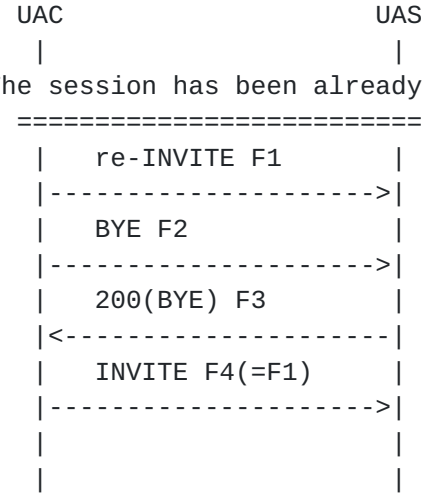
it is RECOMMENDED for UAS to generate 487 to any pending requests after receiving BYE. In the example, Bob sends 487 to ini-INVITE

since he receives BYE while the ini-INVITE is in pending state.

However, Alice receives a final response to INVITE (a 200 from Carol) even though she has successfully terminated the dialog with Bob. This means that, regardless of the success/failure of the BYE in the Early state, Alice MUST be prepared for the establishment of a new dialog until receiving the final response for the INVITE and terminating the INVITE transaction.

It is not illegal to send BYE in the Early state to terminate a specific early dialog according to the caller's intent. However, the choice of BYE or CANCEL in the Early state must be made carefully. CANCEL is appropriate when the goal is to abandon the call attempt entirely. BYE is appropriate when the goal is to abandon a particular early dialog while allowing the call to be completed with other destinations. When using either BYE or CANCEL the UAC must be prepared for the possibility that a call may still be established to one (or more) destinations.

Appendix B - BYE request overlapped on re-INVITE



This case could look similar to the one in Section 3.2.3. However, it is not a race condition. This case describes the behavior where there is no response for INVITE for some reasons. The appendix explains the behavior in such case and its rationale behind, since this case is likely to cause confusion.

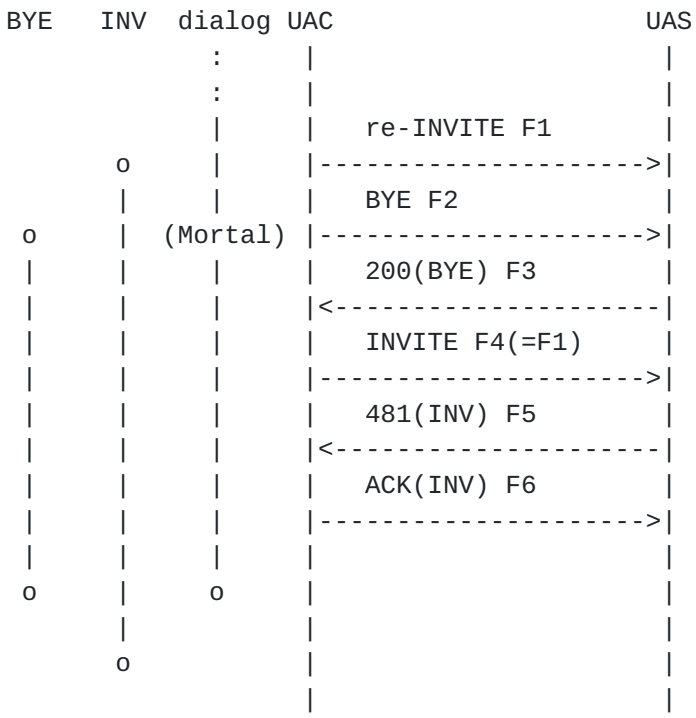
First of all, it is important not to confuse the behavior of the transaction layer and that of the dialog layer. RFC 3261 [1] details the transaction layer behavior. The dialog layer behavior is explained in this document. It has to be noted that these behaviors are independent of each other, even though the both layers change

their states triggered by sending or receiving of the same SIP
messages (A dialog can be terminated even though a transaction still

remain, and vice versa).

In the sequence above, there is no response for F1, and F2 (BYE) is sent immediately after F1 (F1 is a mid-dialog request. If F1 was ini-INVITE, BYE could not be sent before UAC received a provisional response to the request with To tag).

Below is a figure which illustrates UAC's dialog state and transaction state.



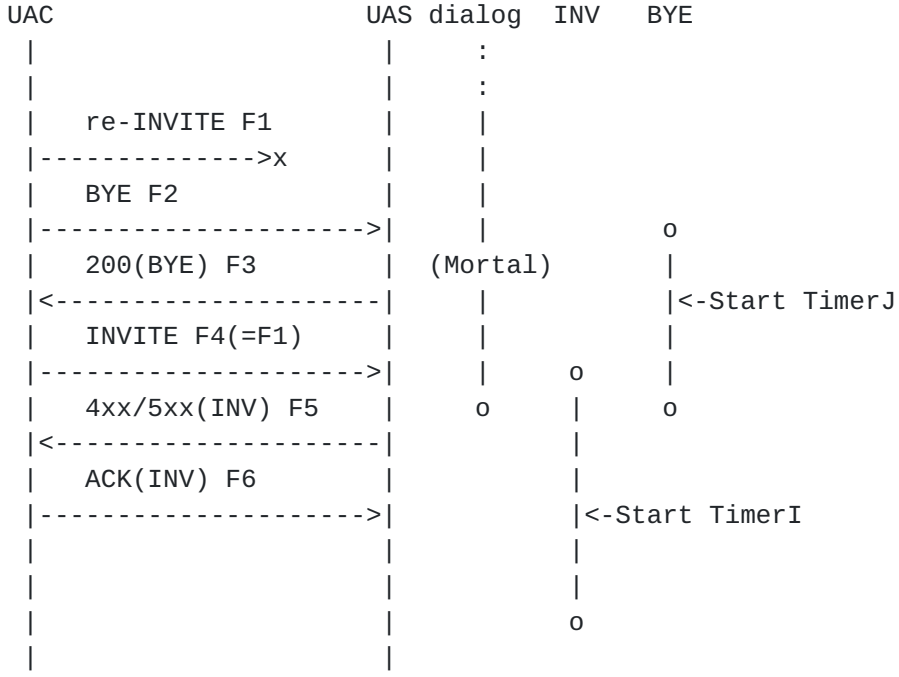
For UAC, the INVITE client transaction begins at the point F1 is sent. The UAC sends BYE (F2) immediately after F1. This is a legitimate behavior. (Usually the usage of each SIP method is independent, for BYE and others. However, it should be noted that it is prohibited to send a request with a SDP offer while the previous offer is in progress.)

After that, F2 triggers the BYE client transaction. At the same time, the dialog state transits to the Mortal state and then only a BYE or its response can be handled.

It is permitted to send F4 (a retransmission of INVITE) in the Mortal state, because the retransmission of F1 is handled by the transaction layer, and the INVITE transaction has not yet transited to the Terminated state. As it is mentioned above, the dialog and the transaction behave independently each other.
Therefore the transaction handling has to be continued even though

the dialog moved to the Terminated state.

Next, UAS's state is shown below.



For UAS, it can be regarded that F1 packet is lost or delayed (Here the behavior is explained for the case UAS receives F2 BYE before F1 INVITE). Therefore, F2 triggers the BYE transaction for UAS, and simultaneously the dialog moves to the Mortal state.
Then, upon the reception of F4 the INVITE server transaction begins. (It is allowed to start the INVITE server transaction in the Mortal state. The INVITE server transaction begins to handle received SIP request regardless of the dialog state.)
UAS's TU sends an appropriate error response for F4 INVITE, either 481 (because the TU knows that the dialog which matches to the INVITE is in the Terminated state) or 500 (because the re-sent F4 has an out-of-order CSeq).
(It is mentioned above that F4 (and F1) INVITE is a mid-dialog request. Mid-dialog requests have a To tag. It should be noted that UAS's TU does not begin a new dialog upon the reception of INVITE with a To tag.)

Appendix C - UA's behavior for CANCEL

This section explains the CANCEL behaviors which indirectly involve in the dialog state transition in the Early state. CANCEL does not have any influence on UAC's dialog state. However, the request has indirect influence on the dialog state transition because it has a significant effect on ini-INVITE. For UAS the CANCEL request has more direct effects on the dialog than the sending of CANCEL by UAC,

because they can be a trigger to send the 487 response. Figure 3 explains UAS's behavior in the Early state. This flow diagram is

only an explanatory figure, and the actual dialog state transition is as illustrated in Figure 1 and 2.

In the flow, full lines are related to dialog state transition, and dotted lines are involved with CANCEL. (r) represents the reception of signaling, and (s) means sending. There is no dialog state for CANCEL, but here the Cancelled state is virtually handled just for the ease of understanding of the UA's behavior when it sends and receives CANCEL.

Below, UAS's flow is explained.

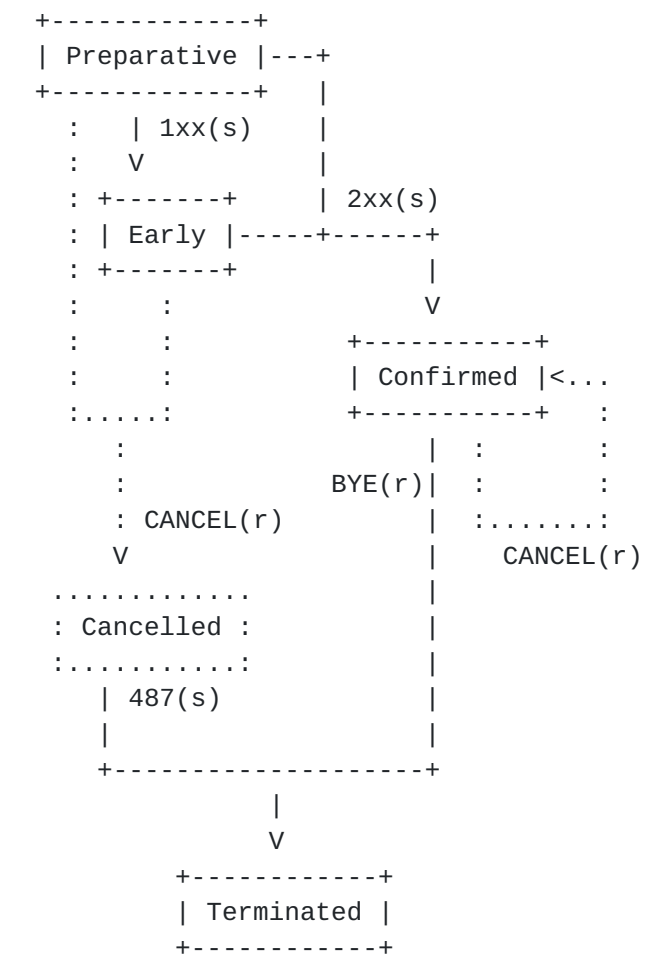


Figure 3. CANCEL flow diagram for UAS

There are two behaviors for UAS depending on the state when it receives CANCEL.

One is when UAS receives CANCEL in the Early states. In this case the UAS immediately sends 487 for the INVITE, and the dialog transits to the Terminated state.

The other is the case in which UAS receives CANCEL in the Confirmed

state. In this case the dialog state transition does not occur because UAS has already sent a final response to the INVITE to which the CANCEL is targeted.
(Note that, because of UAC's behavior, a UAS that receives a CANCEL in the Confirmed state can expect to receive a BYE immediately and move to the Terminated state. However, the UAS's state does not transit until it actually receives BYE.)

Appendix D - Notes on the request in Mortal state

This section describes UA's behavior in the Mortal state which needs careful attention. Note that every transaction completes independent of others, following the principle of RFC 3261 [1].

In the Mortal state, BYE can be accepted, and the other messages in the INVITE dialog usage are responded with an error. However, sending of ACK and the authentication procedure for BYE are conducted in this state. (The handling of messages concerning multiple dialog usages is out of the scope of this document. Refer to [6] for further information.)

ACK for error responses is handled by the transaction layer, so the handling is not related to the dialog state. Unlike the ACK for error responses, ACK for 2xx responses is a request newly generated by TU. However, the ACK for 2xx and the ACK for error responses are both a part of the INVITE transaction, even though their handling differs (Section 17.1.1.1, RFC 3261 [1]).
Therefore, the INVITE transaction is completed by the three-way handshake, which includes ACK, even in the Mortal state.

Considering actual implementation, UA needs to keep the INVITE dialog usage until the Mortal state finishes, so that it is able to ACK for a 2xx response in the Mortal state.
If a 2xx to INVITE is received in the Mortal state, the duration of the INVITE dialog usage will be extended to 64*T1 seconds after the receiving of the 2xx, to cope with the possible 2xx retransmission. (The duration of the 2xx retransmission is 64*T1, so the UA need to be prepared to handle the retransmission for this duration.)
However, the UA shall send error response to other requests, since the INVITE dialog usage in the Mortal state is kept only for the sending of ACK for 2xx.

BYE authentication procedure shall be processed in the Mortal state. When authentication is requested by 401 or 407 response, UAC resends BYE with an appropriate credentials. Also UAS handles the retransmission of the BYE which it requested authentication itself.

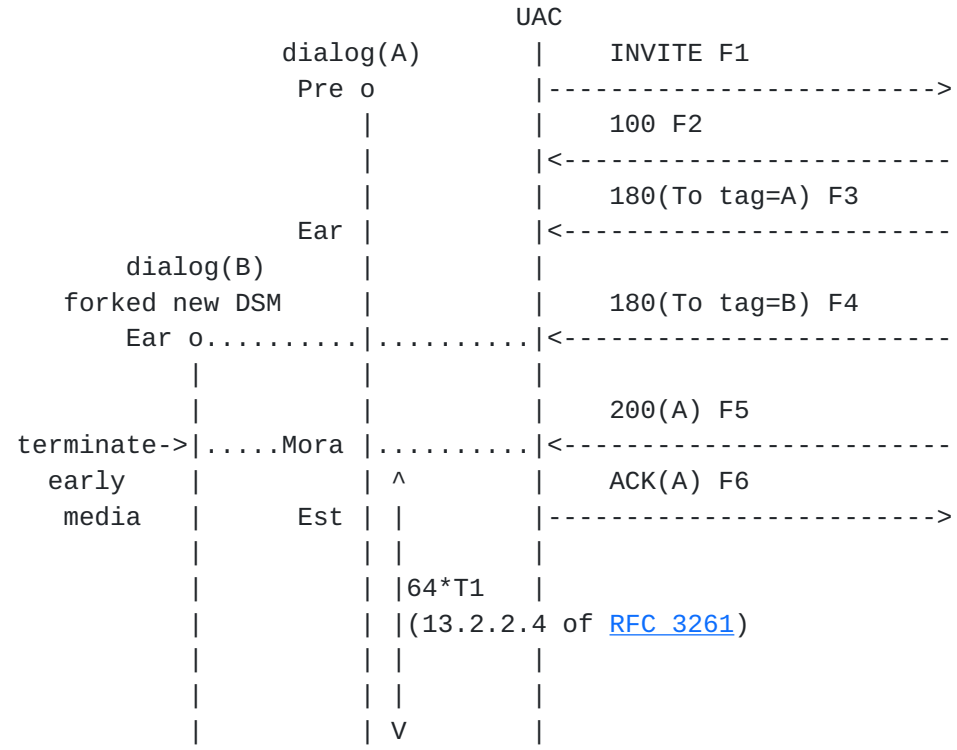
Appendix E - Forking and receiving new To tags

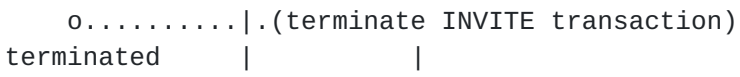
This section details the behavior of TU when it receives multiple responses with a different To tag to ini-INVITE.

When an INVITE is forked inside a SIP network, there is a possibility that the TU receives multiple responses with a different To tag to ini-INVITE (See Section 12.1, 13.1, 13.2.2.4, 16.7, 19.3, etc. of RFC 3261). If the TU receives multiple 1xx responses with a different To tag, the original DSM forks and a new DSM instance is created. As a consequence multiple early dialogs are generated.

If one of the multiple early dialogs receives a 2xx response, it naturally transits to the Confirmed state. No DSM state transition occurs for the other early dialogs, and their sessions (early media) terminate. The TU of the UAC terminates the INVITE transaction after 64*T1 seconds starting at the point of receiving the first 2xx response. Moreover, all mortal early dialogs which do not transit to the Established state are terminated (See Section 13.2.2.4 of RFC 3261). By "mortal early dialog" we mean any early dialog that the UA will terminate when another early dialog is confirmed.

Below is an example sequence in which two 180 responses with a different To tag are received, and then a 200 response for one of the early dialogs (dialog A) is received. Dot lines (..) in sequences are auxiliary lines to represent the influence on dialog B.




o.....|.(terminate INVITE transaction)
terminated |

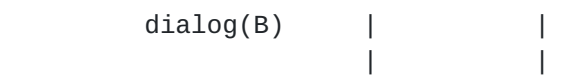


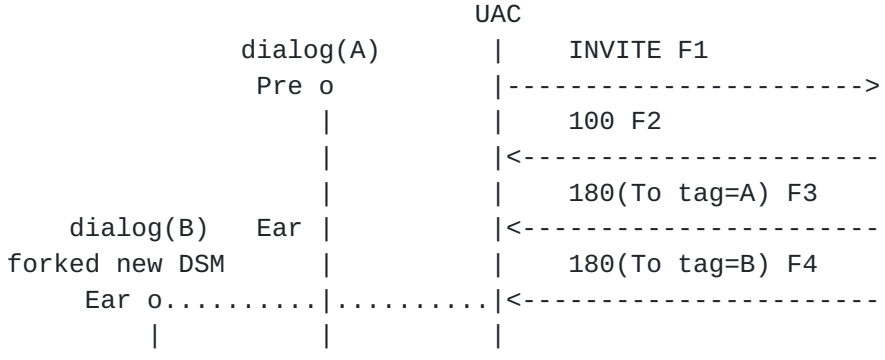
Figure 4. Receiving 1xx responses with different To tags

The figure above shows the DSM inside a SIP TU. Triggered by the reception of a provisional response with a different To tag (F4 180(To tag=B)), DSM forks and the early dialog B is generated. After 64*T1 seconds after the dialog A receives 200 OK response, the dialog B, which does not transit to the Established state, terminates.

Next, the behavior of a TU which receives multiple 2xx responses with a different To tag is explained. When a mortal early dialog, which did not match the first 2xx response the TU received, receives another 2xx response which matches its To tag before 64*T1 INVITE transaction timeout, its DSM state transits to the Confirmed state. However, the session on the mortal early dialog is terminated when the TU receives the first 2xx to establish a dialog, so no session is established for the mortal early dialog. Therefore, when the mortal early dialog receives a 2xx response, the TU send an ACK and, immediately after, the TU usually sends a BYE to terminate DSM. (In special cases, e.g. a UA intends to establish multiple dialogs, the TU may not send the BYE.)

The handling of the second early dialog after receiving the 200 for the first dialog is quite appropriate for a typical device, such as a phone. It is important to note that what is being shown is a typically useful action and not the only valid one. Some devices might want to handle things differently. For instance, a conference focus that has sent out an INVITE that forks may want to accept and mix all the dialogs it gets. In that case, no early dialog is treated as mortal.

Below is an example sequence in which two 180 responses with a different To tag are received and then a 200 response for each of the early dialogs is received.



terminate->|.....Mora |.....|<----------
200(A) F5

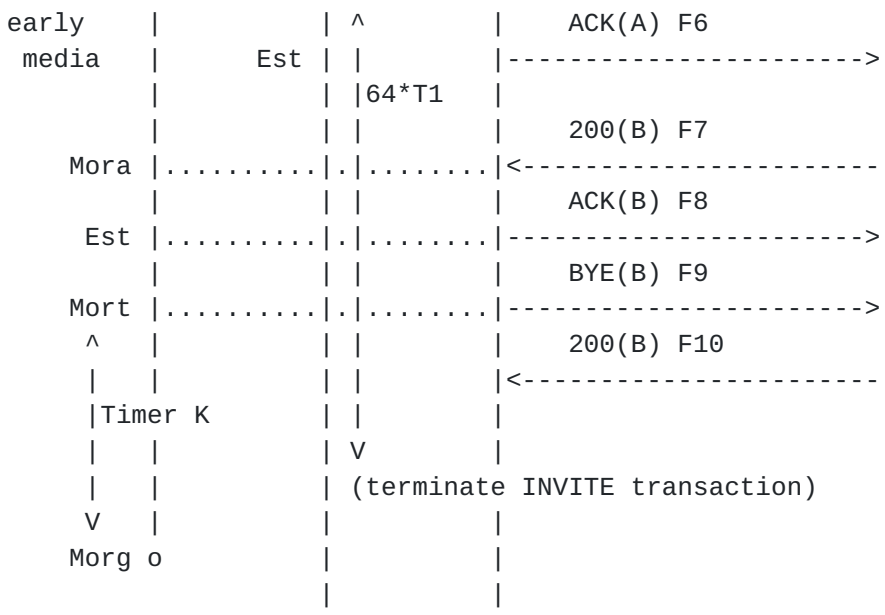
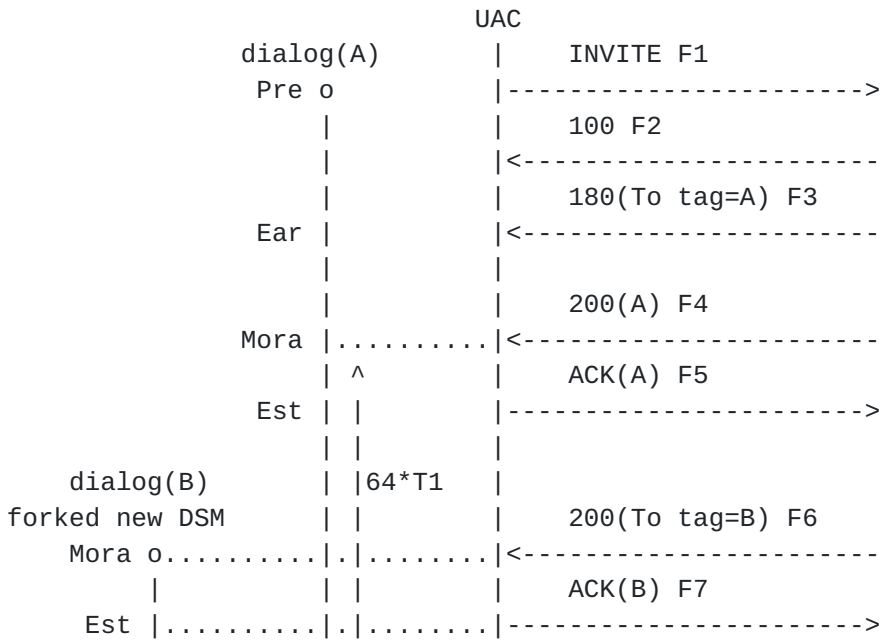
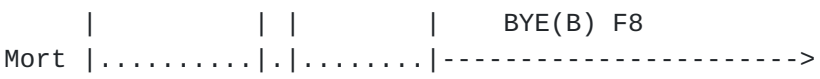


Figure 5. Receiving 1xx and 2xx responses with different To tags

Below is an example sequence when a TU receives multiple 200 responses with a different To tag before 64*T1 timeout of the INVITE transaction, even though a TU does not receive provisional response, the TU needs to respond to the 2xx responses (See Section 13.2.2.4 of RFC 3261). In that case, the DSM state is forked at the Confirmed state, and then the TU sends an ACK for the 2xx response and, immediately after, the TU usually sends a BYE. (In special cases, e.g. a UA intends to establish multiple dialogs, the TU may not send the BYE.)




BYE(B) F8
Mort

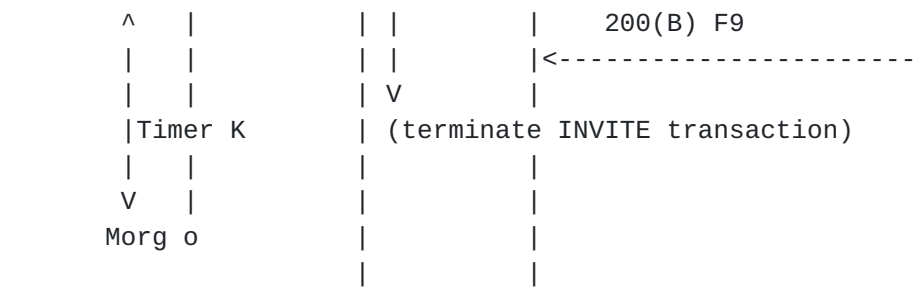
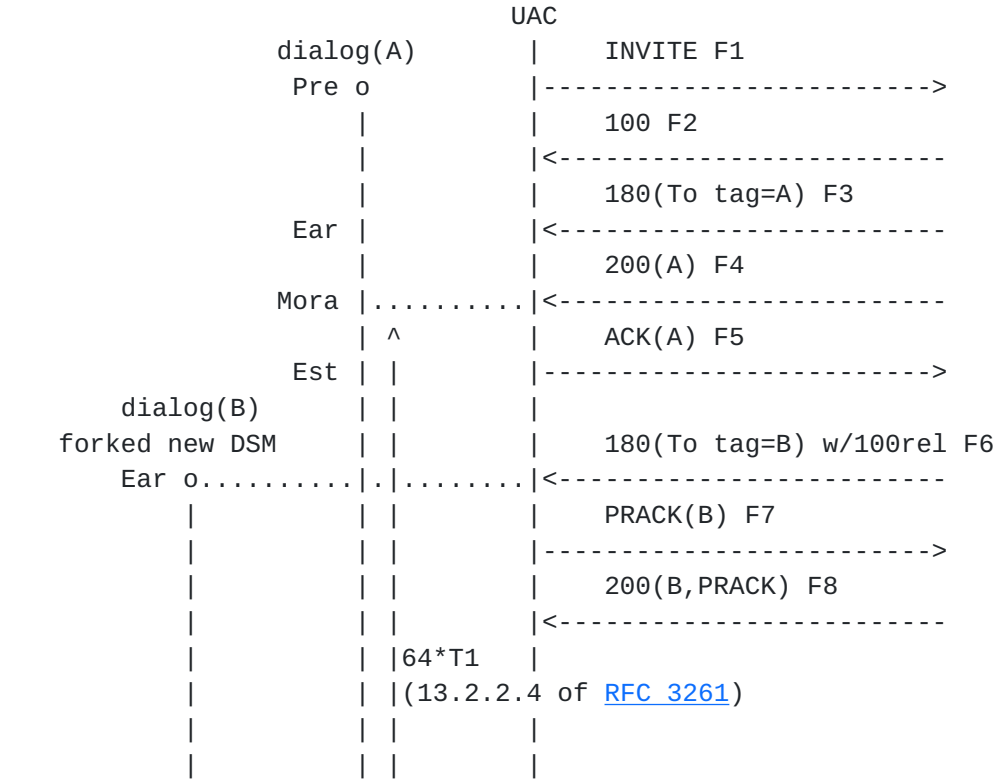


Figure 6. Receiving 2xx responses with different To tags

Below is an example sequence in which the option tag 100rel (RFC 3262 [5]) is required by a 180.

If a forking proxy supports 100rel, it transparently transmits to the UAC a provisional response which contains a Require header with the value of 100rel. Upon receiving a provisional response with 100rel, the UAC establishes the early dialog (B) and send PRACK. (Here, also every transaction completes independent of others".)

As Figure. 4, the early dialog (B) terminates at the same time the INVITE transaction terminates. In the case where a proxy does not support 100rel, the provisional response will be handled in the usual way (a provisional response with 100rel is discarded by the proxy, not to be transmitted to the UAC).



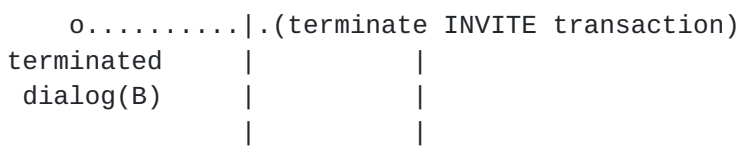


Figure 7. Receiving 1xx responses with different To tags (with 100rel)

Author's Addresses

All listed authors actively contributed large amounts of text to this document.

Miki Hasebe
NTT-east Corporation
19-2 Nishi-shinjuku 3-chome Shinjuku-ku Tokyo 163-8019 Japan

EMail: hasebe.miki@east.ntt.co.jp

Jun Koshiko
NTT-east Corporation
19-2 Nishi-shinjuku 3-chome Shinjuku-ku Tokyo 163-8019 Japan

EMail: j.koshiko@east.ntt.co.jp

Yasushi Suzuki
NTT-east Corporation
19-2 Nishi-shinjuku 3-chome Shinjuku-ku Tokyo 163-8019 Japan

EMail: suzuki.yasushi@east.ntt.co.jp

Tomoyuki Yoshikawa
NTT-east Corporation
19-2 Nishi-shinjuku 3-chome Shinjuku-ku Tokyo 163-8019 Japan

EMail: tomoyuki.yoshikawa@east.ntt.co.jp

Paul H. Kyzivat
Cisco Systems, Inc.
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Email: pkyzivat@cisco.com

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST, AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).