Network Working Group   M. Hasebe
Request for Comments: 5407   J. Koshiko
BCP: 147   NTT-east Corporation
Category: Best Current Practice   Y. Suzuki
                                                    NTT Corporation
                                                     T. Yoshikawa
                                                 NTT-east Corporation
                                                       P. Kyzivat
                                                 Cisco Systems, Inc.
                                                    December 2008

Example Call Flows of Race Conditions in the
Session Initiation Protocol (SIP)

Status of This Memo

   This document specifies an Internet Best Current Practices for the
   Internet Community, and requests discussion and suggestions for
   improvements. Distribution of this memo is unlimited.

Abstract

   This document gives example call flows of race conditions in the
   Session Initiation Protocol (SIP). Race conditions are inherently
   confusing and difficult to thwart; this document shows the best
   practices to handle them. The elements in these call flows include
   SIP User Agents and SIP Proxy Servers. Call flow diagrams and
   message details are given.

Table of Contents

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
   conditions, which stem from transitions in dialog states -- mainly
   transitions during session establishment after the sending of an
   INVITE.

   When implementing SIP, various complex situations may arise.
   Therefore, it is helpful to provide implementors of the protocol with
   examples of recommended terminal and server behavior.

   This document clarifies SIP User Agent (UA) behaviors when messages
   cross each other as race conditions. By clarifying the 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 version 2.0 of SIP, defined in RFC 3261
   [1], with SDP usage as 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 architectural, 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 of 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 of 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 the
   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 errors) of SIP methods, headers, and
parameters. Note: The flows in this document must not be copied
as-is by implementors because additional annotations have been
incorporated into this document for ease of explanation. To sum up,
the procedures described in this document represent well-reviewed
examples of SIP usage, which exemplify best common practice according
to IETF consensus.

For reasons of 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 replicated, 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 that 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

The term "session" is used in this document in the same way it is
used in Sections 13-15 of RFC 3261 [1] (which differs somewhat from
the definition of the term in RFC 3261). RFC 5057 [6] introduces
another term, "invite dialog usage", which is more precisely defined.
The term "session" used herein is almost, but not quite, identical to
the term "invite dialog usage". The two have differing definitions
of when the state ends -- the session ends earlier, when BYE is sent
or received.

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

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

   Below are the DSMs, first for the caller and then for the callee.

```
INV +-----------------------------------------------+
--->|                    Preparative                |
    +-----------------------------------------------+
     |                  |                     |
     | 3xx-6xx          | 1xx-tag             | 2xx
     |                  |                     |
     |                  |            1xx-tag  |
     |                  V            w/new tag |
     |          +----------------+  [new DSM] |
     | 3xx-6xx  |                |  | (new DSM |
    +<---------|      Early     |  | instance |
     |          |                |<--+ created) |
     |          +----------------+            |
     |            |          |                |        2xx w/new tag
     |            | BYE      | 2xx            |        [new DSM]
     |            |          +----------->+<-+          | (new DSM
     |            |                       |             |  instance
+-----C-----------C-----+       +----------C------+    |  created)
|     | Terminated |    |       | Confirmed |    | |
|     |          +<----C---------|          |    | |
|     |          |     BYE(sr)   |          |    | |
|     |          V               |          V    | |
| 2xx |  +-----------+   |       |  +-----------+ | |
| +---C--|           |   |--C-+  |  |           | | | |
| | | |  |   Mortal  |   | | | BYE(r)|  Moratorium|<-C--+
| +---C->|           |   |<--C-+  |  |           | | |
| ACK |  +-----------+   |       |  +-----------+ | |
|     |    |  |          |       |     |         | |
|     |    |  | Timeout  |       |     | ACK     | |
|     |    |  |          |       |     |         | |
|     V    V  |          |       |     V         | |
|  +--------------+      |       |  +-----------+ | |
|  |              |      |       |  |           |--C-+
|  |    Morgue    |      |       |  |Established| | | 2xx,ACK
|  |              |      |       |  |           |<-C-+
|  +--------------+      |       |  +-----------+ |
|                        |       |               |
+------------------------+       +---------------+
```
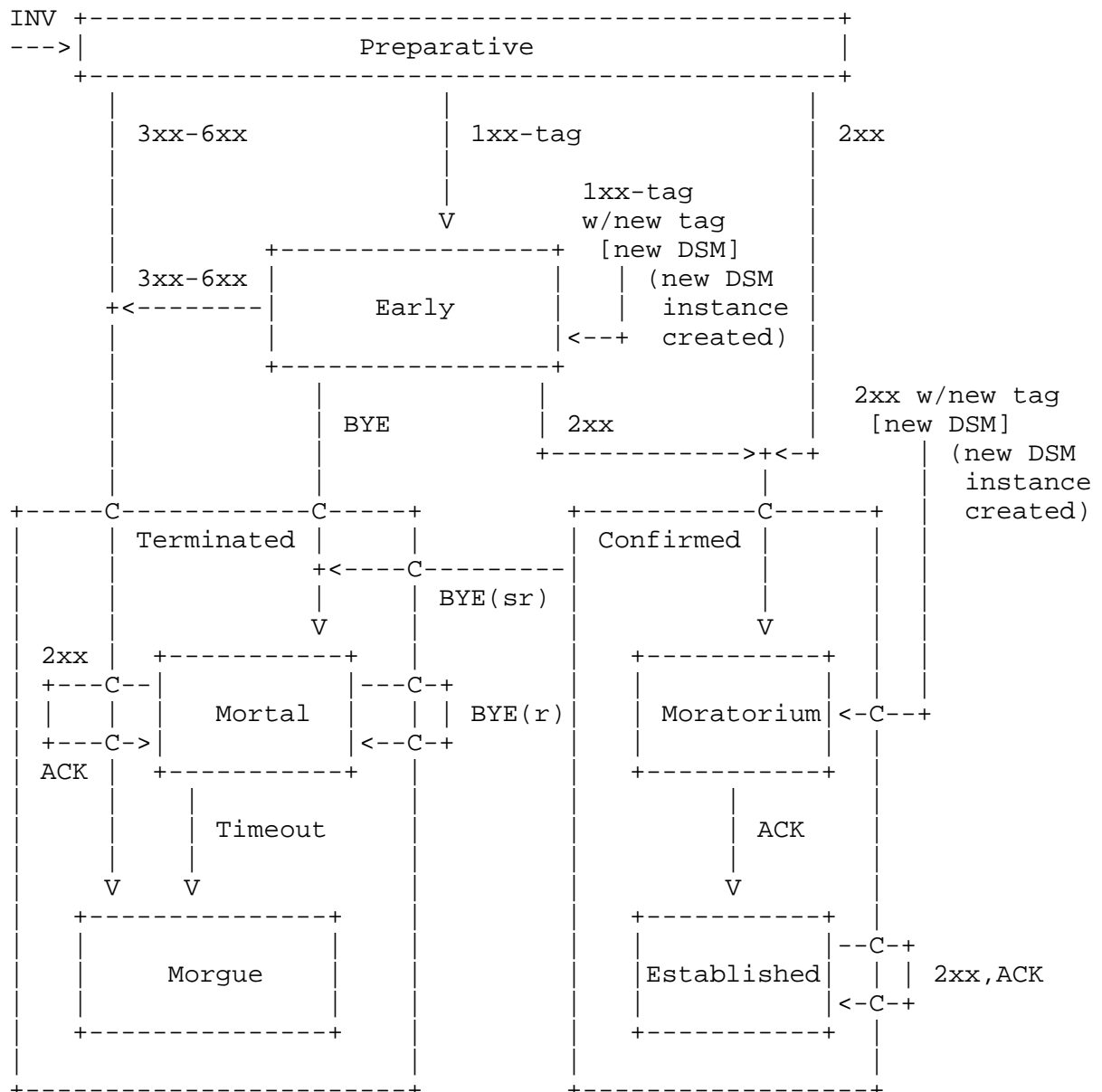
    (r): indicates that only reception is allowed.
         Where (r) is not used as an indicator, "response" means
         receive, and "request" means send.
    (sr): indicates that both sending and reception are allowed.

             Figure 1: DSM for INVITE dialog usage (caller)

Figure 1 represents the caller's DSM for the INVITE dialog usage.
The caller MAY send a BYE in the Early state, even though this
behavior is not recommended. A BYE sent in the Early state
terminates the early dialog using 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 the 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 if this is the
caller's intent. Moreover, until the 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 if it has already sent a CANCEL or BYE and terminated the
dialog (see Appendix A).

```
 INV +-----------------------------------------------+
 --->|                    Preparative                |
     +-----------------------------------------------+
        |                     |                    |
        | 3xx-6xx             | 1xx-tag            | 2xx
        |                     |                    |
        |                     V                    |
        |          +-----------------+             |
        | 3xx-6xx  |                 |             |
       +<--------- |      Early      |             |
        |          |                 |             |
        |          +-----------------+             |
        |            |             |               |
        |            |BYE/487(INV) | 2xx           |
        |            |             |    +---------->+<-+
        |            |             |    |              |
   +-----C-----------C-----+   +----------C------+     |
   |   | Terminated |      |   | Confirmed |     |     |
   |   |           +<----C---------|           |     |     |
   |   |           |   | BYE(sr)|   |           |     |
   |   |           V   |        |   |           V     |
   |   |   +-----------+ |        |   |   +-----------+ |
   |   |   |           |---C-+   |   |   |           |--C-+
   |   |   |   Mortal  | | | BYE |   |   |Moratorium | | | 2xx
   |   |   |           |<--C-+   |   |   |           |<-C-+ if ACK not
   |   |   +-----------+ |        |   |   +-----------+ |   received
   |   |     |           |        |   |     |           |
   |   |     | Timeout   |        |   |     | ACK       |
   |   |     |           |        |   |     |           |
   |   V     V           |        |   |     V           |
   |   +---------------+  |        |   |   +-----------+ |
   |   |               |  |        |   |   |           | |
   |   |    Morgue     |  |        |   |   |Established| |
   |   |               |  |        |   |   |           | |
   |   +---------------+  |        |   |   +-----------+ |
   |                      |        |   |                 |
   +----------------------+        |   +-----------------+
```
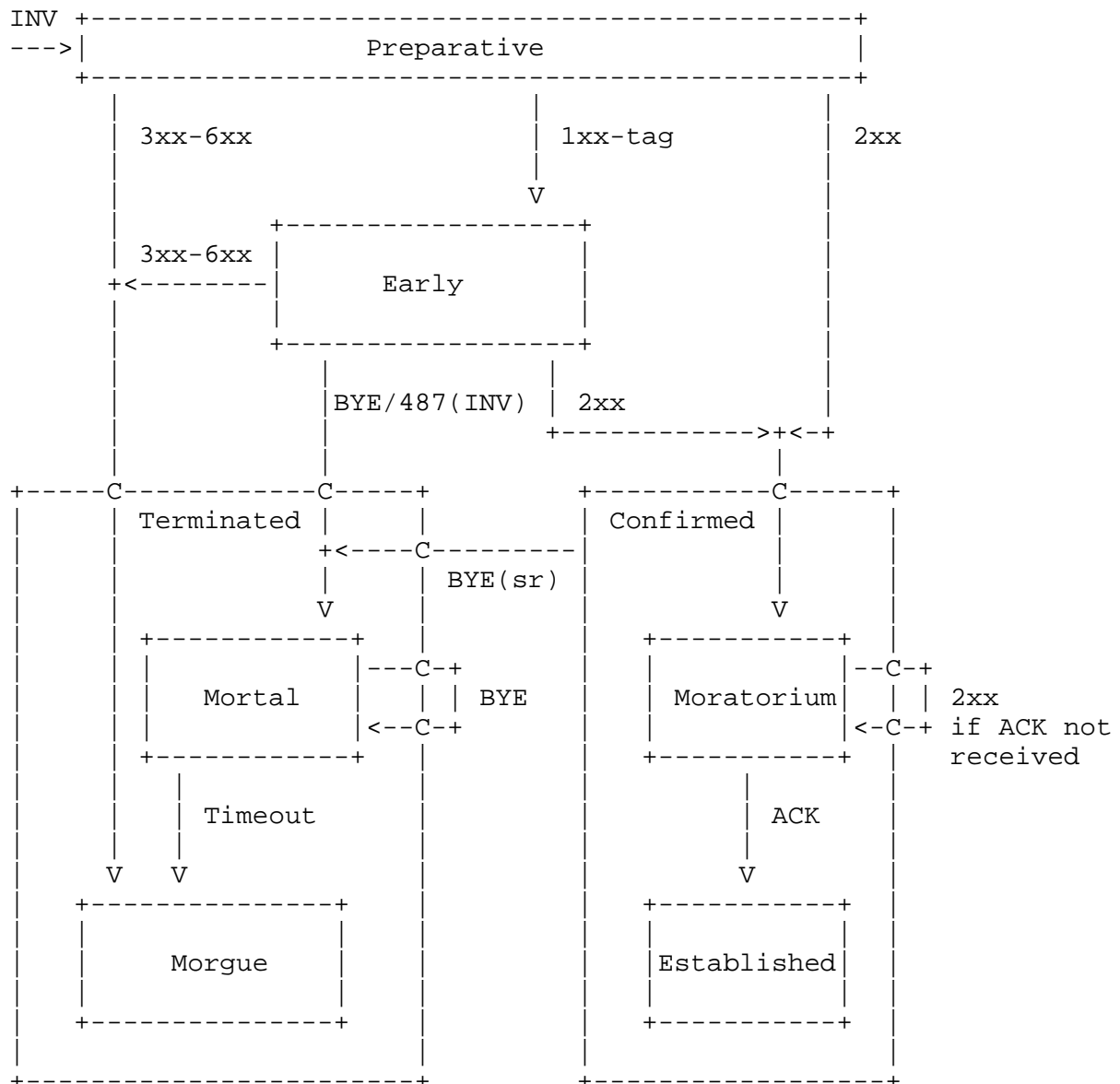
          (sr): indicates that both sending and reception are allowed.
             Where (sr) is not used as an indicator, "response" means send,
             and "request" means receive.

               Figure 2: DSM for INVITE dialog usage (callee)

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

transition by sending a 487 immediately after the reception of the
CANCEL. This behavior upon the reception of the CANCEL request is
further explained in Appendix C.

The UA's behavior in each state is as follows.

Preparative (Pre): The Preparative state is in effect until the
    early dialog is established by sending or receiving a provisional
    response with a To tag after an ini-INVITE is sent or received.
    The dialog does not yet exist in the Preparative state. If the UA
    sends or receives a 2xx response, the dialog state transitions
    from the Preparative state to the Moratorium state, which is a
    substate of the Confirmed state. In addition, if the UA sends or
    receives a 3xx-6xx response, the dialog state transitions 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 shown 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 except 100 Trying. The early dialog exists
    even though the dialog does not exist in this state yet. The
    dialog state transitions from the Early state to the Moratorium
    state, a substate of the Confirmed state, by sending or receiving
    a 2xx response. In addition, the dialog state transitions 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 shown on this DSM
    because they are automatically processed on the transaction layer
    and don't influence the dialog state. The UAC may send a CANCEL
    in the Early state. The UAC may also send a BYE (although it is
    not recommended). The UAS may send a 1xx-6xx response. The
    sending or receiving of a CANCEL request does not have a direct
    influence on the dialog state. The UA's behavior upon the
    reception of the CANCEL request is explained further in Appendix
    C.

Confirmed (Con): The sending or receiving of a 2xx final response
    establishes a dialog. The dialog starts in this state. The
    Confirmed state transitions 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 the
    Established states, which are different with regard to the
    messages that UAs are allowed to send.

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

Established (Est): The Established state is a substate of the
   Confirmed state and inherits its behavior. Both caller and callee
   may send various messages that influence a dialog. The caller
   supports the transmission of ACK to the retransmission of a 2xx
   response to an ini-INVITE.

Terminated (Ter): The Terminated state is subdivided into two
   substates, the Mortal and Morgue states, to cover the behavior
   when a dialog is being terminated. In this state, the UA holds
   information about the dialog that is being terminated.

Mortal (Mort): The caller and callee enter the Mortal state by
   sending or receiving a BYE. The 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 the Mortal state, BYE
   can be accepted, and the other messages in the INVITE dialog usage
   are responded to with an error. This addresses the case where a
   caller and a callee 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. The
   sending or receiving of signaling that influences a dialog is not
   performed. (A dialog is literally terminated.) The caller and
   callee enter the Morgue state via the termination of the BYE or
   INVITE transaction.

3. Race Conditions

   This section details a 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 the sending or receiving of SIP messages
   are shown, and race conditions are indicated by '*race*'. (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 described below.

3.1. Receiving Message in the Moratorium State

   This section shows some examples of call flow race conditions when
   receiving messages from other states while in the Moratorium state.

3.1.1. Callee Receives Initial INVITE Retransmission (Preparative
        State) While in the Moratorium State

```
     State  Alice                                   Bob  State
            |                                        |
            |             ini-INVITE F1              |
            |-------------------------------------->|
      Pre   |           180 F2(Packet loss)         |  Pre
            |              x<----------------------|
            |                                        |  Ear
            | ini-INVITE F4(=F1)          200 F3   |
            |-----------------   -------------|
            |                 \ /                    |  Mora
            |                  X                     |
            |                 / \                    |
            |<----------------   ------------->|  *race*
      Mora  |                ACK F5                  |
            |-------------------------------------->|
      Est   |                                        |  Est
            |                                        |
```
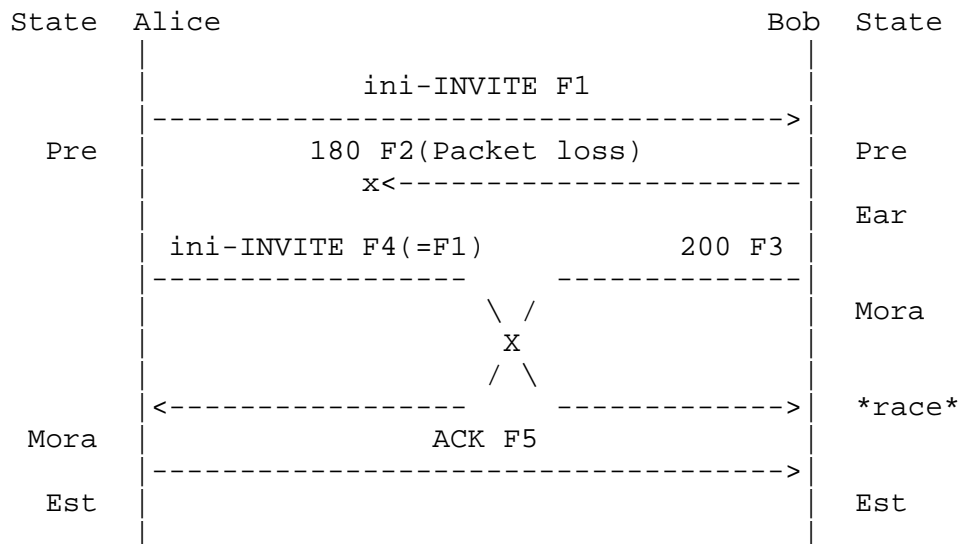
   This scenario illustrates the race condition that occurs when the UAS
   receives a Preparative message while in the Moratorium state. All
   provisional responses to the initial INVITE (ini-INVITE F1) are lost,
   and the UAC retransmits an ini-INVITE (F4). At the same time as this
   retransmission, the UAS generates a 200 OK (F3) to the ini-INVITE and
   terminates the INVITE server transaction, according to Section
   13.3.1.4 of RFC 3261 [1].

   However, it is reported that terminating an INVITE server transaction
   when sending a 200 OK is an essential correction to SIP [7].
   Therefore, the INVITE server transaction is not terminated by F3, and
   F4 MUST be handled properly as a retransmission.

   In RFC 3261 [1], it is not specified whether the UAS retransmits 200
   to the retransmission of ini-INVITE. Considering the retransmission
   of 200 triggered by a timer (the transaction user (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 the
   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 Section 13.3.1.4 of RFC 3261 [1], the INVITE server
   transaction is terminated at this point. However, this has been
   reported as an essential correction to SIP, 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 that have not dealt with the correction [7] (an
   INVITE server transaction is terminated when sending 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 must
   recognize the retransmitted INVITE correctly, without treating it
   as a new INVITE. */

F5 ACK Alice -> Bob

3.1.2. Callee Receives CANCEL (Early State) While in the Moratorium
        State
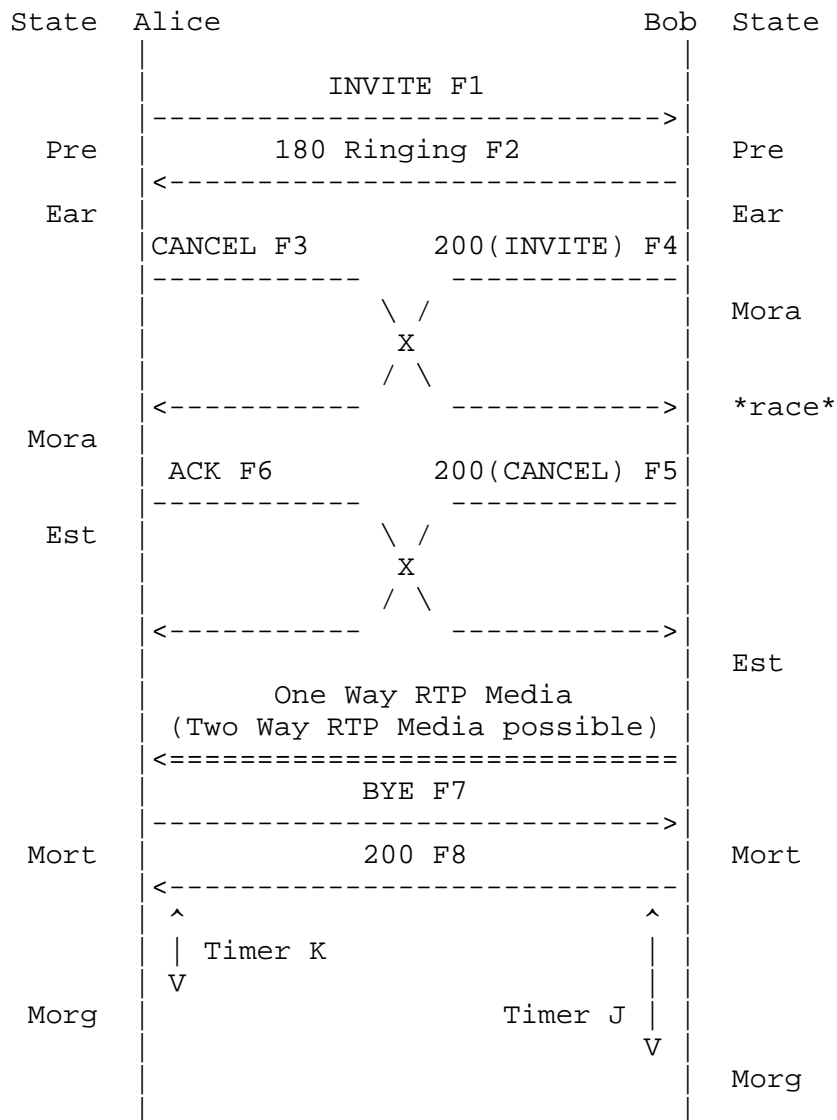
```
    State  Alice                           Bob  State
           |                                 |
           |            INVITE F1            |
           |-------------------------------->|
     Pre   |          180 Ringing F2         | Pre
           |<--------------------------------|
     Ear   |                                 | Ear
           |CANCEL F3        200(INVITE) F4|
           |-----------      ------------|
           |             \ /                 | Mora
           |              X                  |
           |             / \                 |
           |<----------      ----------->| *race*
    Mora   |                                 |
           |  ACK F6         200(CANCEL) F5|
           |-----------      ------------|
     Est   |             \ /                 |
           |              X                  |
           |             / \                 |
           |<----------      ----------->|
           |                                 | Est
           |        One Way RTP Media        |
           |   (Two Way RTP Media possible)  |
           |<================================|
           |             BYE F7              |
           |-------------------------------->|
    Mort   |             200 F8              | Mort
           |<--------------------------------|
           | ^                           ^ |
           | | Timer K                   | |
           | V                           | |
    Morg   |                    Timer J  | |
           |                          V  |
           |                                 | Morg
           |                                 |
```

   This scenario illustrates the race condition that occurs when the UAS
   receives an Early message, CANCEL, while 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 [1], an INVITE server transaction is
   supposed to be terminated by a 200 response, but this has been
   corrected in [7].

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 that the CANCEL request matches,
so a 200 response to the request is sent. This 200 response simply
means that the next hop receives the CANCEL request (successful
CANCEL (200) does not mean the INVITE was actually canceled). When a
UAS has not dealt with the correction [7], the UAC MAY receive a 481
response to the CANCEL since there is no transaction that 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 the INVITE, and if she receives 200 to the INVITE
request she immediately sends a BYE and terminates the dialog. (See
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 case assumes the correction [7] has
   been made. If an INVITE server transaction is terminated
   according to the procedure stated in RFC 3261 [1], the UAC MAY
   receive a 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. Callee Receives BYE (Early State) While in the Moratorium State
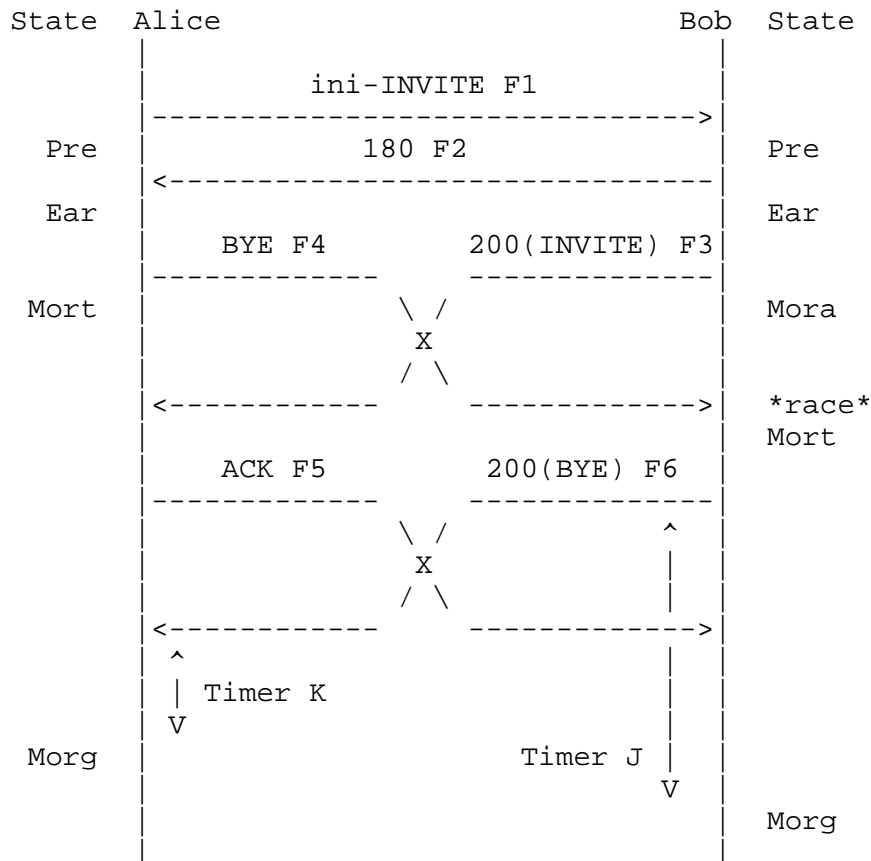
    State Alice   Bob State
           | |
           | ini-INVITE F1 |
           |--------------------------------->|
     Pre | 180 F2 | Pre
           |<---------------------------------|
     Ear | | Ear
           | BYE F4   200(INVITE) F3 |
           |------------   ------------- |
     Mort | \ / | Mora
           | X |
           | / \ |
           |<-----------   -------------> | *race*
           | | Mort
           | ACK F5   200(BYE) F6 |
           |------------   ------------- |
           | \ /   ^ |
           | X | |
           | / \ | |
           |<-----------   -------------> |
           | ^ | |
           | | Timer K | |
           | V | |
     Morg | Timer J | |
           | V |
           | | Morg
           | |

    This scenario illustrates the race condition that occurs when the UAS
    receives an Early message, BYE, while in the Moratorium state. Alice
    sends a BYE in the Early state, and Bob sends a 200 OK to the initial
    INVITE request at the same time. Bob receives the BYE in the
    Confirmed dialog state although 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 the

caller wants to terminate all early dialogs instead of only 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 if that is 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 enters the Mortal state
on sending the BYE request, and remains Mortal until the Timer K
timeout occurs. In the Mortal state, the UAC does not establish a
session even though it receives a 200 response to the INVITE. Even
so, the UAC sends an ACK to 200 in order to complete the INVITE
transaction. The ACK is always sent to complete the three-way
handshake of the 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 transitions 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. Callee Receives re-INVITE (Established State) While in the
           Moratorium State (Case 1)

```
      State  Alice                              Bob  State
             |                                  |
             |       ini-INVITE w/offer1 F1     |
             |--------------------------------->|
       Pre   |              180 F2              |  Pre
             |<---------------------------------|
       Ear   |                                  |  Ear
             |      200(ini-INV) w/answer1 F3   |
             |<---------------------------------|
      Mora   |         ACK F4(packet loss)      |  Mora
             |-------------------->x            |
       Est   |                                  |
             | re-INVITE F6       200 F5(=F3)   |
             |   w/offer2            w/answer1  |
             |------------        -------------|
             |           \ /                    |
             |            X                     |
             |           / \                    |
             |<-----------        ------------->|  *race*
             |                  200(re-INV) F8  |
             | ACK F7(=F4)          w/answer2   |
             |------------        -------------|
             |           \ /                    |
             |            X                     |
             |           / \                    |
             |<-----------        ------------->|
             |          ACK (re-INV) F9         |  Est
             |--------------------------------->|
             |                                  |
             |                                  |
```
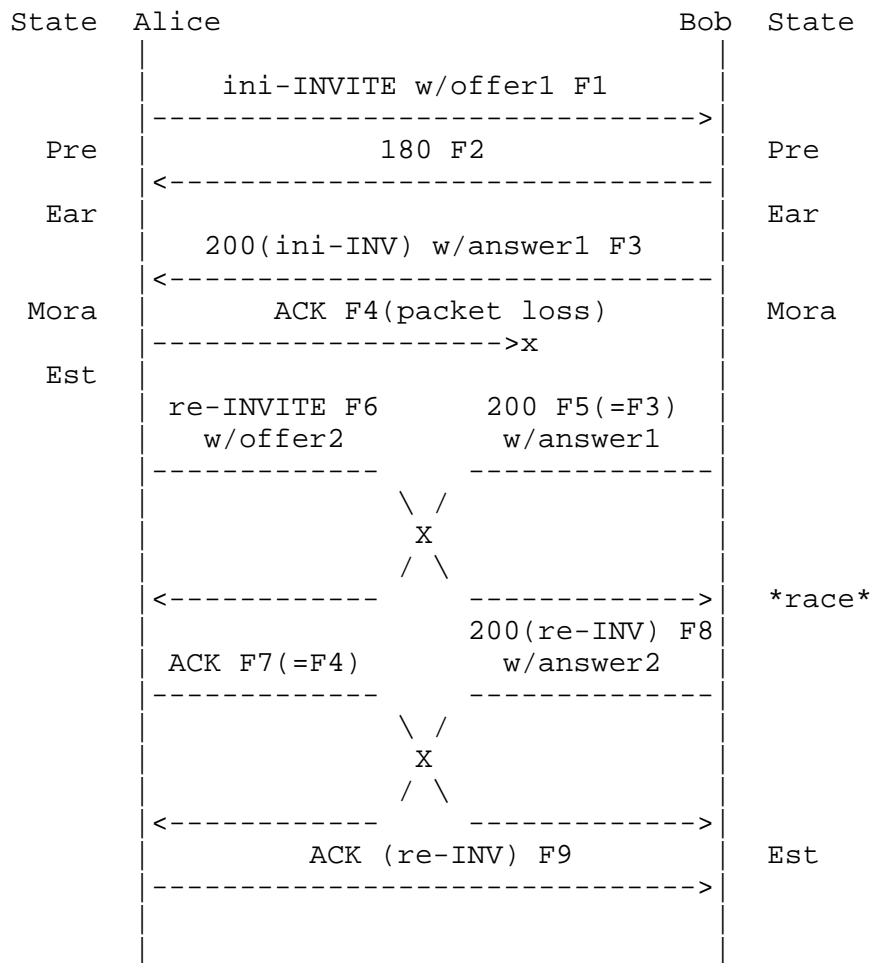
   This scenario illustrates the race condition that occurs when a UAS
   in the Moratorium state receives a re-INVITE sent by a UAC in the
   Established state.

   The UAS receives a re-INVITE (with offer2) before receiving an ACK
   for the ini-INVITE (with offer1). The UAS sends a 200 OK (with
   answer2) to the re-INVITE (F8) because it has sent a 200 OK (with
   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.)

   As 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 crossover. This example recommends that 200 be sent instead

of 491 because it does not have an influence on the session.
However, a 491 response can also lead to the same outcome, so either
response can be used.

Moreover, if the 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
[1]).   The UA should not reject or drop the ACK on grounds of the
CSeq number.

Note: Implementation issues are outside the scope of this document,
but the following tip is provided for avoiding race conditions of
this type. The caller can delay sending re-INVITE F6 for some period
of time (2 seconds, perhaps), after which the caller can reasonably
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 that the UA can
behave like this. In this case, it is the implementor's choice as to
how long to wait. In most cases, such an implementation may be
useful to prevent the type of race condition shown in this section.
This document expresses no preference about whether or not they
should wait for an ACK to be delivered. After considering the impact
on user experience, implementors should decide whether or not to wait
for a while, because the user experience depends on the
implementation and has no direct bearing on protocol behavior.

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-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 the
   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
```

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

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

   /* The 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)

   /* "(=F4)" of ACK F7 shows that it is equivalent to F4 in that it is
      an ACK for F3.  This doesn't mean that F4 and F7 must be equal in
      Via-branch value.  Although it is ambiguous in RFC 3261 whether
      the Via-branch of ACK F7 differs from that of F4, it doesn't
      affect the UAS's behavior. */

   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. Callee Receives re-INVITE (Established State) While in the
        Moratorium State (Case 2)

```
   State  Alice                              Bob  State
          |                                  |
          |       ini-INVITE (no offer) F1   |
          |--------------------------------->|
    Pre   |              180 F2              | Pre
          |<---------------------------------|
    Ear   |                                  | Ear
          |       200(ini-INV) w/offer1 F3   |
          |<---------------------------------|
   Mora   |    ACK w/answer1 F4(packet loss) | Mora
          |------------------->x             |
    Est   |                                  |
          | re-INVITE F6        200 F5(=F3)  |
          |   w/offer2            w/offer1   |
          |------------         -------------|
          |            \ /                   |
          |             X                    |
          |            / \                   |
          |<------------         ----------->|
          | ACK F7(=F4)         491(re-INV) F8|
          |------------         -------------|
          |            \ /                   |
          |             X                    |
          |            / \                   |
          |<------------         ----------->|
          |         ACK (re-INV) F9          | Est
          |--------------------------------->|
          |                                  |
          |                                  |
          |                                  |
```

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

   Bob sends a 491 to the re-INVITE (F6) since he is not able to
   properly handle a new request until he receives an answer. (Note:
   500 with a 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.) The same result
   will be reached if F6 is an UPDATE with offer.

   Note: As noted in Section 3.1.4, the caller may delay sending a re-
   INVITE F6 for some period of time (2 seconds, perhaps), after which
   the caller may reasonably assume that its ACK has been received, to
   prevent this type of race condition. This document expresses no
   preference about whether or not they should wait for an ACK to be
   delivered. After considering the impact on user experience,
   implementors should decide whether or not to wait for a while,
   because the user experience depends on the implementation and has no
   direct bearing on protocol behavior.

   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
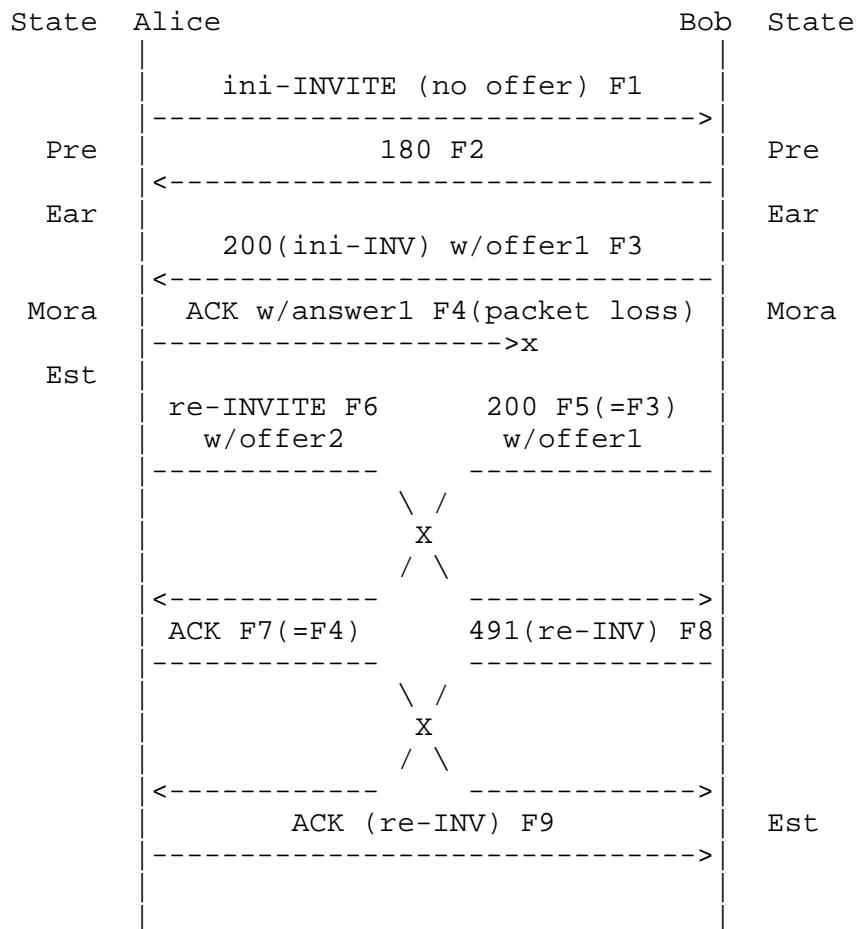   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 INVITE
   Contact: <sip:bob@client.biloxi.example.com;transport=udp>
   Content-Type: application/sdp
   Content-Length: 133

   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)

/* The 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. "(=F4)" of ACK F7 shows that it is equivalent to the F4 in
   that it is an ACK for F3. This doesn't mean that F4 and F7 are
   necessarily equal in Via-branch value. Although it is ambiguous
   in RFC 3261 whether the Via-branch of ACK F7 differs from that of
   F4, it doesn't affect the UAS's behavior. */

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. Callee Receives BYE (Established State) While in the Moratorium
        State
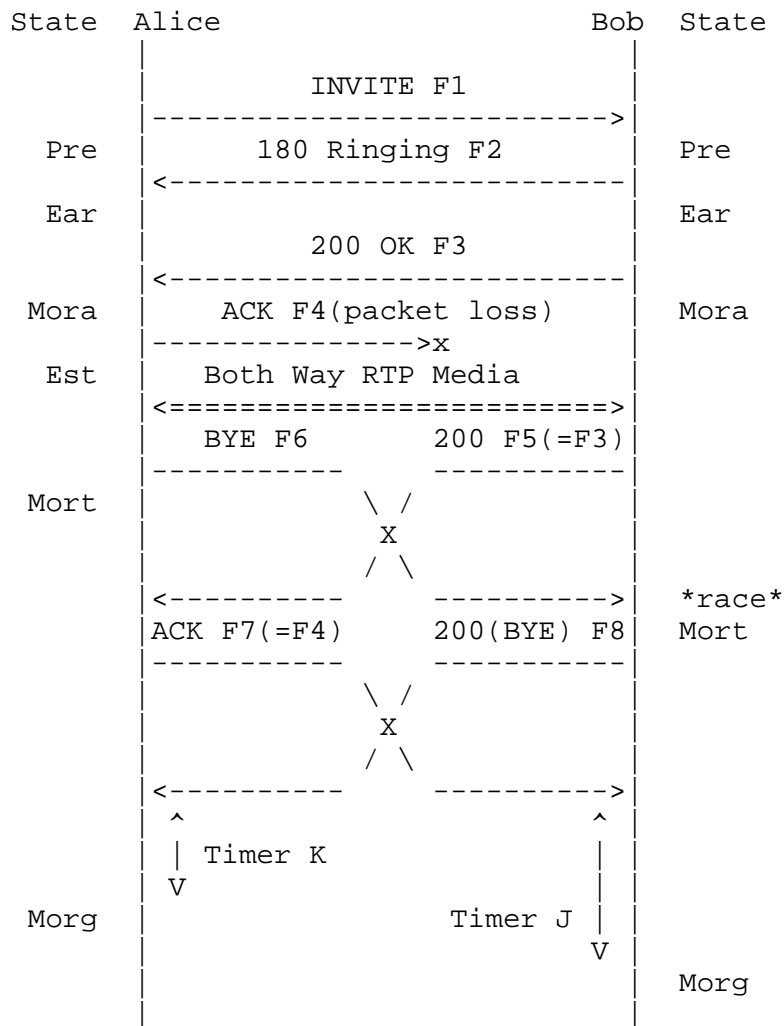
```
   State  Alice                           Bob  State
          |                              |
          |          INVITE F1           |
          |----------------------------->|
    Pre   |        180 Ringing F2        |  Pre
          |<-----------------------------|
    Ear   |                              |  Ear
          |           200 OK F3          |
          |<-----------------------------|
   Mora   |      ACK F4(packet loss)     |  Mora
          |--------------->x             |
    Est   |      Both Way RTP Media      |
          |<============================>|
          |    BYE F6        200 F5(=F3) |
          |-----------      -----------  |
   Mort   |            \ /               |
          |             X                |
          |            / \               |
          |<----------      ---------->| *race*
          |ACK F7(=F4)      200(BYE) F8| Mort
          |-----------      -----------  |
          |            \ /               |
          |             X                |
          |            / \               |
          |<----------      ---------->|
          | ^                      ^   |
          | | Timer K              |   |
          | V                      |   |
   Morg   |                Timer J |   |
          |                      V |   |
          |                        | Morg
          |                        |
          |                        |
```

   This scenario illustrates the race condition that occurs when the UAS
   receives an Established message, BYE, while in the Moratorium state.
   An ACK request for a 200 OK response is lost (or delayed). Bob
   retransmits the 200 OK to the ini-INVITE, and at the same time Alice
   sends a BYE request and terminates the session. Upon receipt of the
   retransmitted 200 OK, Alice's UA might be inclined to reestablish the
   session. But that is wrong -- the session should not be
   reestablished when the dialog is in the Mortal state. Moreover, in
   the case where the UAS sends an offer in a 200 OK, the UAS should not
   start a session again, for the same reason, if the UAS receives a
   retransmitted ACK after receiving a BYE.

   Note: As noted in Section 3.1.4, implementation issues are outside
   the scope of this document, but the following tip is provided for
   avoiding race conditions of this type. The caller can delay sending
   BYE F6 for some period of time (2 seconds, perhaps), after which the
   caller can reasonably 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 that the
   UA can behave like this. In this case, it is the implementor's
   choice as to how long to wait. In most cases, such an implementation
   may be useful to prevent the type of race condition shown in this
   section. This document expresses no preference about whether or not
   they should wait for an ACK to be delivered. After considering the
   impact on user experience, implementors should decide whether or not
   to wait for a while, because the user experience depends on the
   implementation and has no direct bearing on protocol behavior.

   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

   /* The 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 transitions 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

   /* "(=F4)" of ACK F7 shows that it is equivalent to the F4 in that it
      is an ACK for F3. This doesn't mean that F4 and F7 must be equal
      in Via-branch value. Although it is ambiguous in RFC 3261 whether
      the Via-branch of ACK F7 differs from that of F4, it doesn't
      affect the UAS's behavior. */

```
   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 race conditions when
   receiving messages from other states while in the Mortal state.

3.2.1. UA Receives BYE (Established State) While in the Mortal State
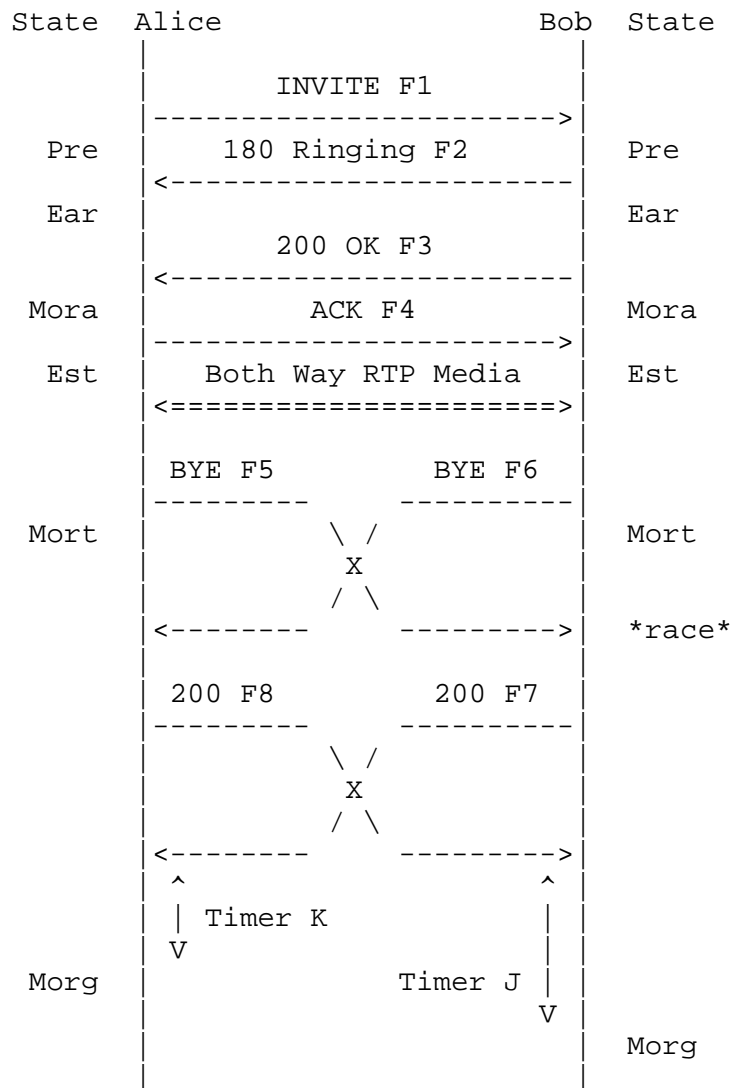
```
   State  Alice                          Bob  State
          |                               |
          |            INVITE F1          |
          |------------------------------>|
    Pre   |        180 Ringing F2         |  Pre
          |<------------------------------|
    Ear   |                               |  Ear
          |            200 OK F3          |
          |<------------------------------|
   Mora   |             ACK F4            |  Mora
          |------------------------------>|
    Est   |        Both Way RTP Media      |  Est
          |<=============================>|
          |                               |
          |  BYE F5          BYE F6       |
          |---------      ----------|
   Mort   |          \ /                   |  Mort
          |           X                    |
          |          / \                   |
          |<--------    --------->|  *race*
          |                               |
          |  200 F8          200 F7       |
          |---------      ----------|
          |          \ /                   |
          |           X                    |
          |          / \                   |
          |<--------    --------->|
          | ^                    ^ |
          | | Timer K            | |
          | V                    | |
   Morg   |               Timer J | |
          |                     V |
          |                       |  Morg
          |                       |
```

This scenario illustrates the race condition that occurs when the UAS
receives an Established message, BYE, while 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, the UA remains in the Mortal state.

UAs in the Mortal state return error responses to the requests that
operate within a dialog or session, such as re-INVITE, UPDATE, or
REFER. However, the UA shall return a 200 OK to the BYE taking the
use case into consideration where a caller and a callee exchange
reports about the session when it is being terminated. (Since the
dialog and the session both terminate when a BYE is sent, the choice
of sending a 200 or an error response upon receiving a BYE while 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 transitioned from the Established state to the
      Mortal state by sending a BYE, Alice responds with a 200 to the
      BYE request. */

3.2.2. UA Receives re-INVITE (Established State) While in the Mortal
        State
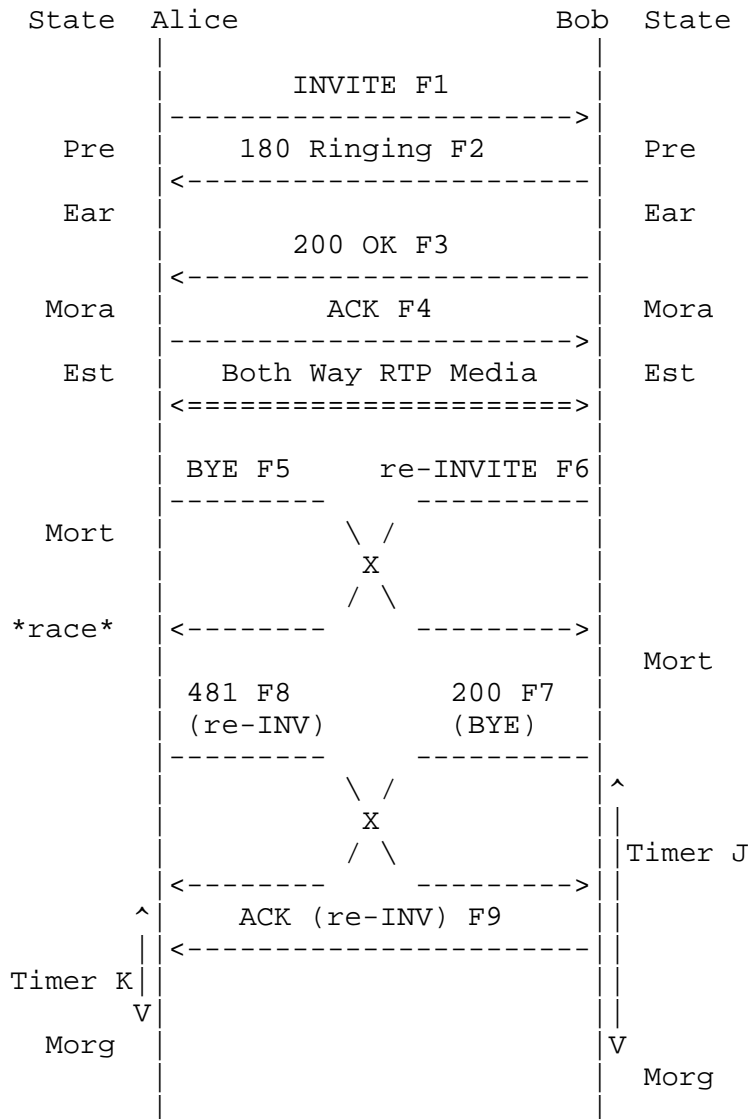
```
     State  Alice                       Bob State
            |                         |
            |        INVITE F1        |
            |------------------------>|
      Pre   |     180 Ringing F2      | Pre
            |<----------------------- |
      Ear   |                         | Ear
            |        200 OK F3        |
            |<----------------------- |
      Mora  |         ACK F4          | Mora
            |------------------------>|
      Est   |    Both Way RTP Media   | Est
            |<=======================>|
            |                         |
            |  BYE F5     re-INVITE F6|
            |---------    ----------  |
      Mort  |         \ /             |
            |          X              |
            |         / \             |
    *race*  |<--------    --------->  |
            |                         | Mort
            |  481 F8       200 F7    |
            |  (re-INV)     (BYE)     |
            |---------    ----------  |
            |         \ /           |^
            |          X            ||
            |         / \           ||Timer J
            |<--------    --------->||
          ^ |    ACK (re-INV) F9    ||
          | |<---------------------||
    Timer K| |                      ||
          V |                       ||
      Morg  |                       |V
            |                       | Morg
            |                         |
```

   This scenario illustrates the race condition that occurs when the UAS
   receives an Established message, re-INVITE, while in the Mortal
   state. Bob sends a re-INVITE, and Alice sends a BYE at the same

time. The re-INVITE receives a 481 response since the TU of Alice
has transitioned from the Established state to the Mortal state by
sending BYE. Bob sends an 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 transitions 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 transitions to the Mortal state at the moment
   Alice sends the BYE, but Bob does not know this until he receives
   the BYE. Therefore, the dialog is in the Terminated state from
   Alice's point of view, but in 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. UA Receives 200 OK for re-INVITE (Established State) While in
        the Mortal State
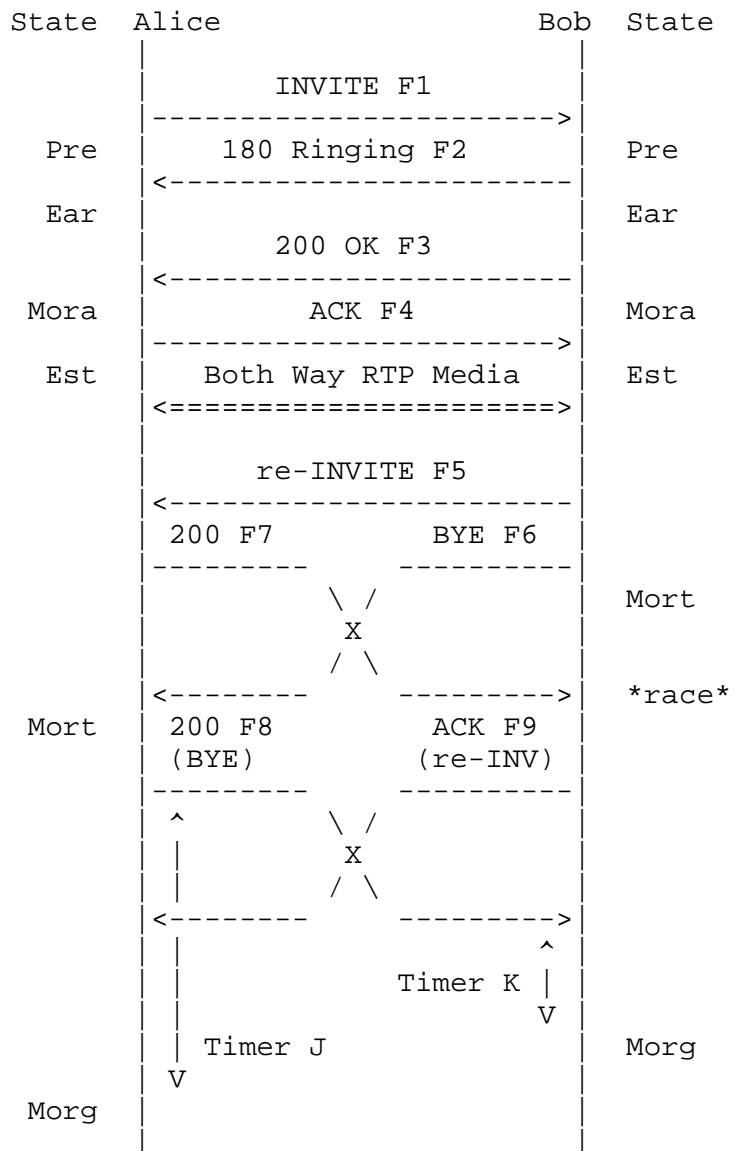
```
   State  Alice                         Bob  State
          |                           |
          |          INVITE F1        |
          |-------------------------->|
    Pre   |      180 Ringing F2       |  Pre
          |<--------------------------|
    Ear   |                           |  Ear
          |          200 OK F3        |
          |<--------------------------|
    Mora  |           ACK F4          |  Mora
          |-------------------------->|
    Est   |     Both Way RTP Media    |  Est
          |<=========================>|
          |                           |
          |        re-INVITE F5       |
          |<--------------------------|
          |  200 F7          BYE F6   |
          |---------      ---------   |
          |           \ /             |  Mort
          |            X              |
          |           / \             |
          |<--------      -------->   |  *race*
    Mort  |  200 F8          ACK F9   |
          |  (BYE)          (re-INV)  |
          |---------      ---------   |
          | ^          \ /            |
          | |           X             |
          | |          / \            |
          |<--------      -------->   |
          | |                    ^    |
          | |        Timer K     |    |
          | |                    V    |
          | |  Timer J           |    |  Morg
          | V                         |
    Morg  |                           |
          |                           |
```

   This scenario illustrates the race condition that occurs when the UAS
   receives an Established message, 200 to a re-INVITE, while in the
   Mortal state. Bob sends a BYE immediately after sending a re-INVITE.
   (For example, in the case of a telephone application, it is possible
   that a user hangs up the phone immediately after refreshing the
   session.) Bob sends an ACK for a 200 response to INVITE while in the
   Mortal state, completing the INVITE transaction.

      Note: As noted in Section 3.1.4, implementation issues are outside
      the scope of this document, but the following tip is provided for
      avoiding race conditions of this type. The UAC can delay sending a
      BYE F6 until the re-INVITE transaction F5 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 that the UA can
      behave like this. In this case, it is the implementor's choice as to
      how long to wait. In most cases, such an implementation may be
      useful in preventing the type of race condition described in this
      section. This document expresses no preference about whether or not
      they should wait for an ACK to be delivered. After considering the
      impact on user experience, implementors should decide whether or not
      to wait for a while, because the user experience depends on the
      implementation and has no direct bearing on protocol behavior.

      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: 38482762982201885511@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 transitions 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. Callee Receives ACK (Moratorium State) While in the Mortal State
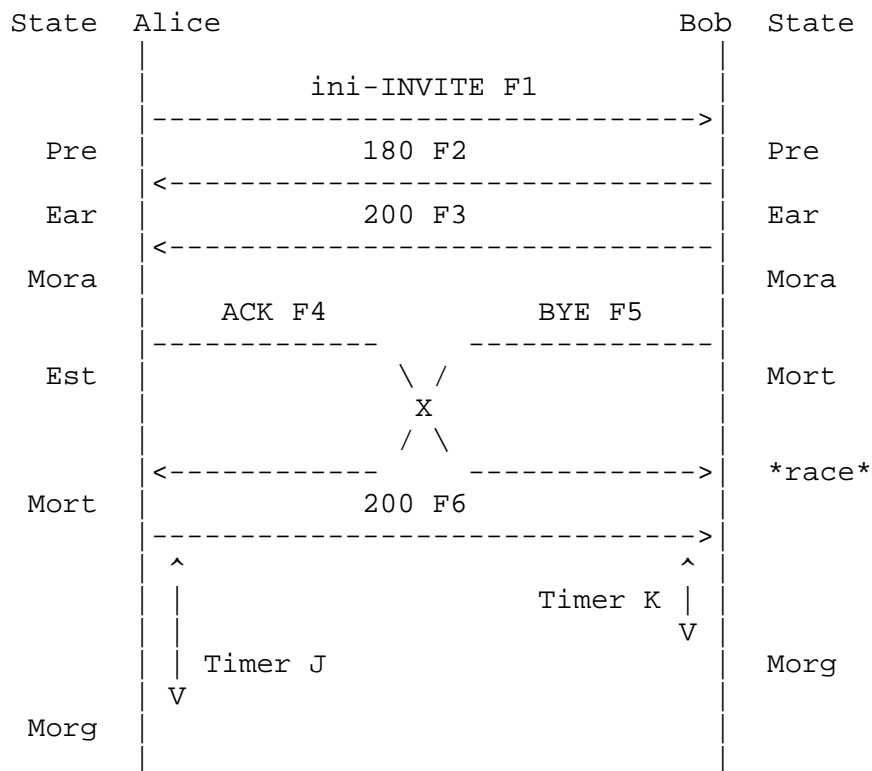
```
  State  Alice                               Bob  State
           |                                   |
           |           ini-INVITE F1           |
           |---------------------------------->|
    Pre    |             180 F2                |  Pre
           |<----------------------------------|
    Ear    |             200 F3                |  Ear
           |<----------------------------------|
   Mora    |                                   |  Mora
           |      ACK F4            BYE F5      |
           |------------       -------------   |
    Est    |            \ /                    |  Mort
           |             X                     |
           |            / \                    |
           |<-----------       ------------->  |  *race*
   Mort    |             200 F6                |
           |---------------------------------->|
           | ^                               ^ |
           | |                    Timer K |   |
           | |                             V   |
           | |  Timer J                    |   |  Morg
           | V                             |   |
   Morg    |                                   |
           |                                   |
```

This scenario illustrates the race condition that occurs when the UAS
receives an Established message, ACK to 200, while 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, there is a
race condition. A session is not started when the ACK is received
because Bob has already terminated the session by sending a BYE. The
answer in the ACK request is just ignored.

Note: As noted in Section 3.1.4, implementation issues are outside
the scope of this document, but the following tip is provided for
avoiding 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 that the UA can behave like
this. In this case, it is the implementor's choice as to how long to
wait. In most cases, such an implementation may be useful in
preventing the type of race condition described in this section.
This document expresses no preference about whether or not they
should wait for an ACK to be delivered. After considering the impact
on user experience, implementors should decide whether or not to wait
for a while, because the user experience depends on the
implementation and has no direct bearing on protocol behavior.

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

3.3.1. Re-INVITE Crossover

```
   Alice                            Bob
     |                               |
     |            INVITE F1           |
     |------------------------------>|
     |         180 Ringing F2         |
     |<------------------------------|
     |            200 OK F3           |
     |<------------------------------|
     |             ACK F4             |
     |------------------------------>|
     |        Both Way RTP Media      |
     |<=============================>|
     |                               |
     |re-INVITE F5    re-INVITE F6   |
     |-----------    -------------|
```

```
    |                  \  /                |
    |                   X                  |
    |                  /  \                |
    |<-----------    ------------>|
    |    491 F8          491 F7    |
    |-----------    ------------|
    |                  \  /                |
    |                   X                  |
    |                  /  \                |
    |<-----------    ------------>|
    |   ^ ACK F9         ^ ACK F10|
    --|---------    ----|-------|
    |  |                \  /     |        |
    |  |                 X       |        |
    |  |                / \      |        |
    |<-|----------    ---|------->|
    |  |                         |        |
    |  |0-2.0 sec                |        |
    |  |                         |        |
    |  v   re-INVITE F11(=F6)    |        |
    |<-----------------|--------|
    |    200 OK F12            |        |
    |-----------------|------->|
    |      ACK F13            |        |
    |<-----------------|--------|
    |                         |        |
    |                         |2.1-4.0 sec
    |                         |        |
    |re-INVITE F14(=F5) v     |        |
    |-------------------------->|
    |      200 OK F15          |
    |<--------------------------|
    |        ACK F16          |
    |-------------------------->|
    |                         |
    |                         |
```
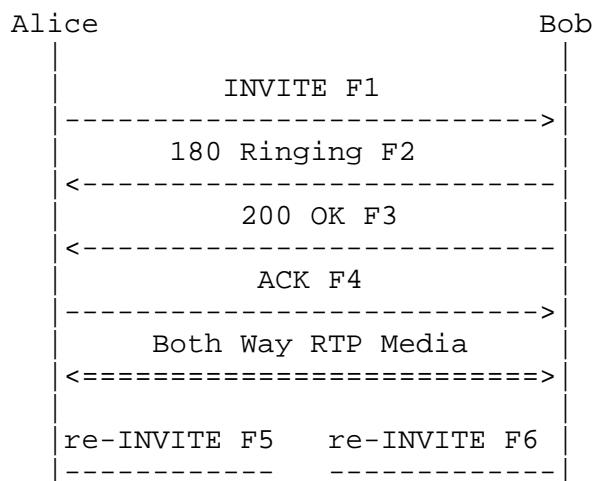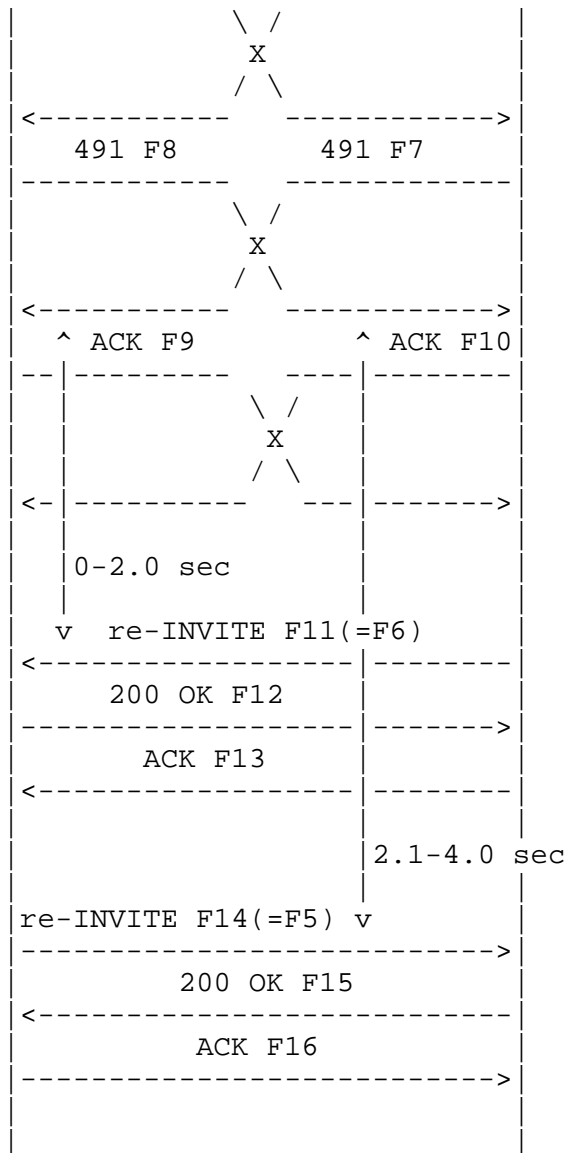
In this scenario, Alice and Bob send re-INVITEs at the same time.
When two re-INVITEs cross in the same dialog, they are retried, each
after a different interval, according to Section 14.1 of RFC 3261
[1]. When Alice sends the re-INVITE and it crosses with Bob's, the
re-INVITE will be retried after 2.1-4.0 seconds because she owns the
Call-ID (she generated it). Bob will retry his 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 or not it has
generated the Call-ID of the dialog, 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 retries the re-INVITE for session refresh earlier than
Alice. If Alice was to retry 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 does not need
to retry his re-INVITE to refresh the session.

In another instance, where two re-INVITEs for session modification
cross over, retrying the same re-INVITE again after a 491 by the
Call-ID owner (the UA that retries its re-INVITE after the other UA)
may result in unintended behavior, so the UA must decide if the retry
of the re-INVITE is necessary. (For example, when a call hold and an
addition of video media cross over, mere retry 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 another 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

```
   Alice                                 Bob
      |                                   |
      |              INVITE F1            |
      |---------------------------------->|
      |           180 Ringing F2          |
      |<----------------------------------|
      |                                   |
      |               200 OK F3           |
      |                                   |
```

```
               |<------------------------|
               |          ACK F4         |
               |------------------------>|
               |     Both Way RTP Media   |
               |<=======================>|
               |                         |
               |  UPDATE F5    re-INVITE F6 |
               |-----------    ------------|
               |         \ /             |
               |          X              |
               |         / \             |
               |<----------    ---------->|
               |  491 F8        491 F7   |
               |  (re-INVITE)   (UPDATE) |
               |-----------    ------------|
               |         \ /             |
               |          X              |
               |         / \             |
               |<----------    ---------->|
               |   ^        ACK F9    ^   |
               |<-|---------------|--------|
               |  |              |  |      |
               |  |0-2.0 sec     |  |      |
               |  |              |  |      |
               |  v   re-INVITE F10 |      |
               |<----------------|--------|
               |    200 OK F11   |        |
               |----------------|-------->|
               |     ACK F12     |        |
               |<----------------|--------|
               |                |  |      |
               |                |2.1-4.0 sec |
               |                |  |      |
               |    UPDATE F13   v        |
               |------------------------>|
               |     200 OK F14          |
               |<------------------------|
               |                         |
               |                         |
```

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

Note: A 491 response is sent because an SDP offer is pending, and 491
is an error that is related to matters that impact 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
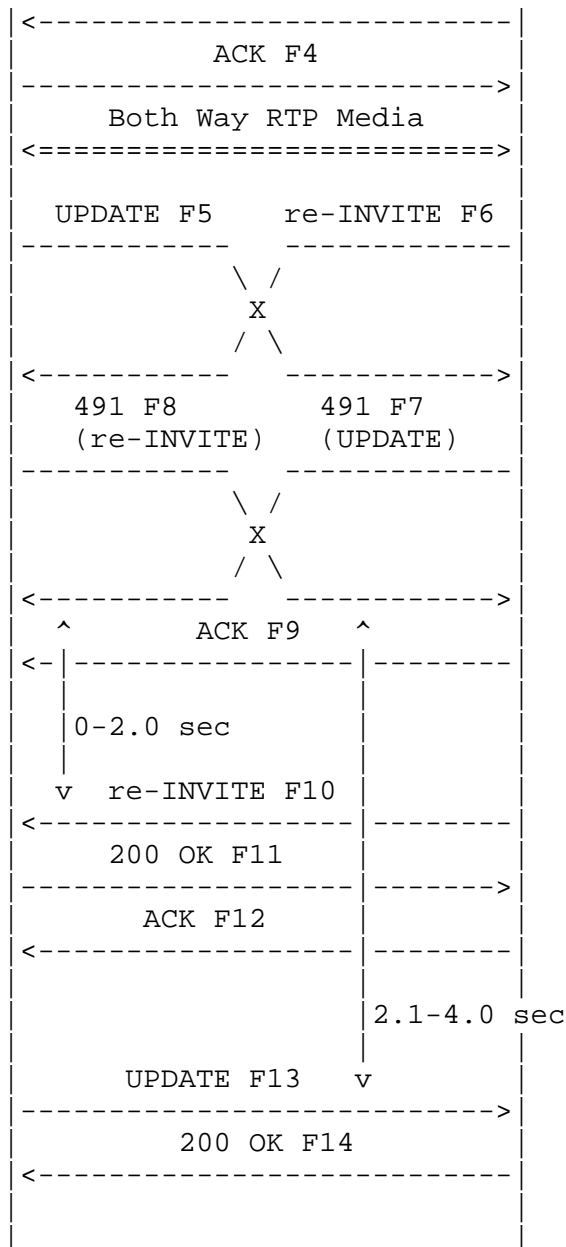m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=sendonly

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

   /* This is a case where a re-INVITE for a session refresh and an
      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 is 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 the 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 (Established State) While in the Mortal State
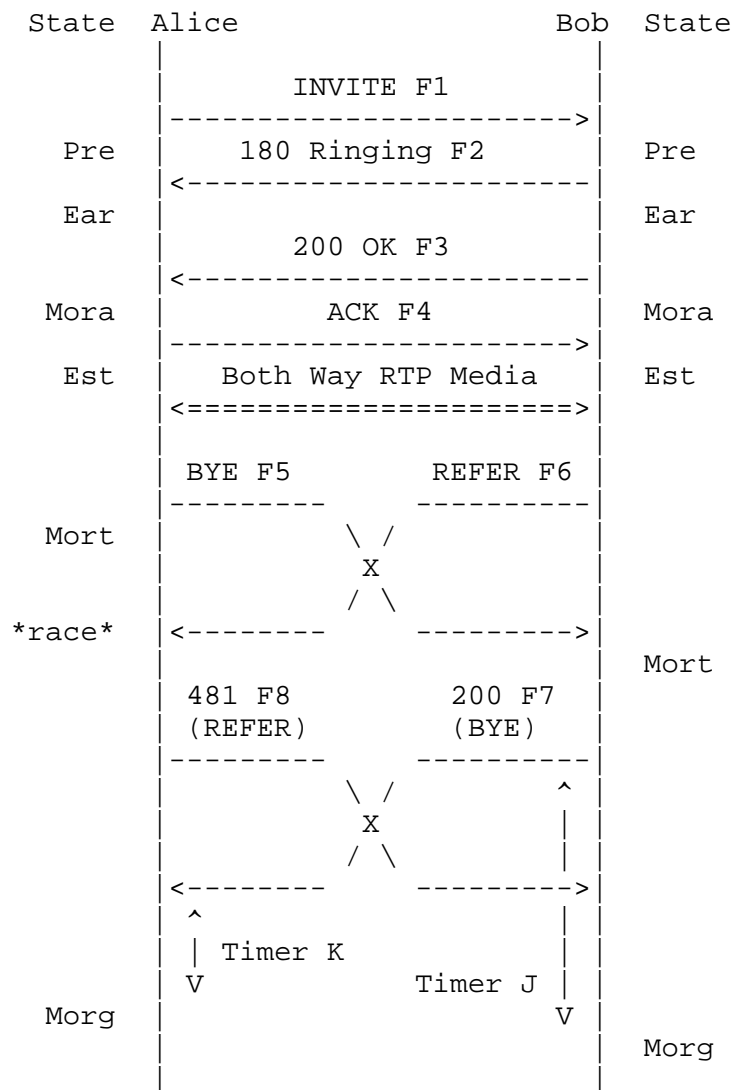
```
    State  Alice                      Bob  State
             |                         |
             |         INVITE F1       |
             |------------------------>|
      Pre  |       180 Ringing F2    | Pre
             |<------------------------|
      Ear  |                         | Ear
             |         200 OK F3       |
             |<------------------------|
     Mora  |          ACK F4         | Mora
             |------------------------>|
      Est  |      Both Way RTP Media | Est
             |<======================>|
             |                         |
             | BYE F5        REFER F6 |
             |---------    ----------|
     Mort  |          \ /            |
             |           X             |
             |          / \            |
    *race*  |<--------    --------->|
             |                         | Mort
             | 481 F8        200 F7   |
             | (REFER)       (BYE)    |
             |---------    ----------|
             |          \ /      ^   |
             |           X       |   |
             |          / \      |   |
             |<--------    --------->|
             | ^                 |   |
             | | Timer K         |   |
             | V        Timer J  |   |
     Morg  |                   V   |
             |                         | Morg
             |                         |
```

   This scenario illustrates the race condition that occurs when the UAS
   receives an Established message, REFER, while in the Mortal state.
   Bob sends a REFER, and Alice sends a BYE at the same time. Bob sends
   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, the
   UA possesses dialog information for an internal process but the
   dialog shouldn't exist outwardly. Therefore, the UA sends an error
   response to the 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 procedures in RFC 5057 [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
   transitions from the Confirmed state to the 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
   transitions to the Mortal state at the moment Alice sends the BYE,
   but Bob doesn't know this 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. Security Considerations

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

5. 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.

6. References

6.1. 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
         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 Session Initiation Protocol (SIP)", RFC 3262,
         June 2002.

6.2. Informative References

   [6]   Sparks, R., "Multiple Dialog Usages in the Session Initiation
         Protocol", RFC 5057, November 2007.

   [7]   Sparks, R., "Correct transaction handling for 200 responses to
         Session Initiation Protocol INVITE requests", Work in Progress,
         July 2008.

Appendix A.   BYE in the Early Dialog

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

```
   Alice                  Proxy                  Bob    Carol
   |                        |                      |      |
   |    INVITE F1           |                      |      |
   |--------------->|    INVITE F2           |      |
   |     100 F3     |---------------->|      |
   |<---------------|  180(To tag=A) F4 |      |
   |    180(A) F5   |<----------------|      |
   |<---------------|                      |      |
   |                        |    INVITE(Fork) F6   |      |
   |                        |----------------------->|      |
   |                        |           100 F7   |      |
   |    BYE(A) F8   |<-----------------------|      |
   |--------------->|     BYE(A) F9         |      |
   |                        |---------------->|      |
   |                        |   200(A,BYE) F10 |      |
   | 200(A,BYE) F11 |<----------------|      |
   |<---------------|   487(A,INV) F12 |      |
   |                        |<----------------|      |
   |                        |    ACK(A) F13    |      |
   |                        |---------------->|      |
   |                        |                      |      |
   |                        |                      |      |
   |                        |   200(To tag=B) F13  |      |
   |    200(B) F14  |<-----------------------|      |
   |<---------------|                      |      |
   |   ACK(B) F15   |                      |      |
   |--------------->|    ACK(B) F16        |      |
   |                        |----------------------->|      |
   |   BYE(B) F17   |                      |      |
   |--------------->|    BYE(B) F18        |      |
   |                        |----------------------->|      |
   |                        |    200(B) F19        |      |
   |    200(B) F20  |<-----------------------|      |
   |<---------------|                      |      |
   |                        |                      |      |
   |                        |                      |      |
```
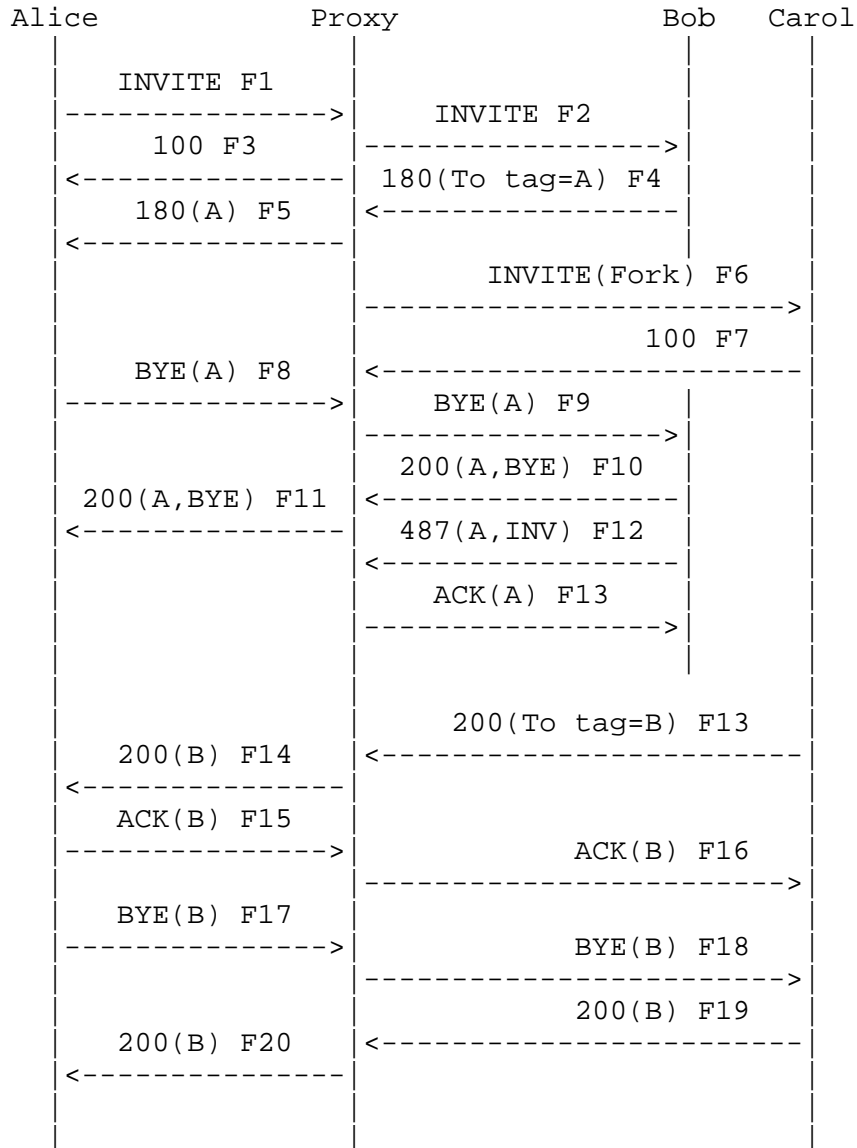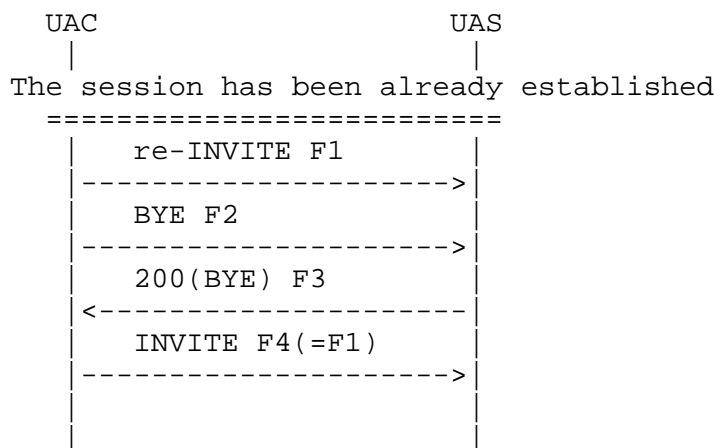
   Care is advised in sending BYE in the Early state when forking by a
   proxy is expected. In this example, the BYE request progresses
   normally, and it succeeds in correctly terminating the dialog with
   Bob. After Bob terminates the dialog by receiving the BYE, he sends
   a 487 to the ini-INVITE. According to Section 15.1.2 of RFC 3261

[1], it is RECOMMENDED for the UAS to generate a 487 to any pending
requests after receiving a BYE. In this example, Bob sends a 487 to
the ini-INVITE since he receives the BYE while the ini-INVITE is in
pending state.

However, Alice receives a final response to the 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 a BYE in the Early state to terminate a
specific early dialog -- it may satisfy the intent of some callers.
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 Overlapping with re-INVITE

```
     UAC                       UAS
      |                         |
The session has been already established
      =========================
      |     re-INVITE F1        |
      |------------------------>|
      |     BYE F2              |
      |------------------------>|
      |    200(BYE) F3          |
      |<------------------------|
      |    INVITE F4(=F1)       |
      |------------------------>|
      |                         |
      |                         |
      |                         |
```

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 when
there is no response to the INVITE for some reason. The appendix
explains the behavior in this case and its rationale, 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 two
behaviors are independent of each other, even though both layers may
be triggered to change their states by sending or receiving the same
SIP messages. (A dialog can be terminated even though a transaction
still remains, and vice versa.)

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

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

```
BYE    INV  dialog UAC                        UAS
              :     |                          |
              :     |                          |
              |     |   re-INVITE F1           |
       o      |     |-------------------->|
              |     |   BYE F2                 |
   o    |  (Mortal) |-------------------->|
   |    |     |     |   200(BYE) F3            |
   |    |     |     |<--------------------|
   |    |     |     |   INVITE F4(=F1)         |
   |    |     |     |-------------------->|
   |    |     |     |   481(INV) F5            |
   |    |     |     |<--------------------|
   |    |     |     |   ACK(INV) F6            |
   |    |     |     |-------------------->|
   |    |     |     |                          |
   o    |     o     |                          |
        |           |                          |
        o           |                          |
                    |                          |
```
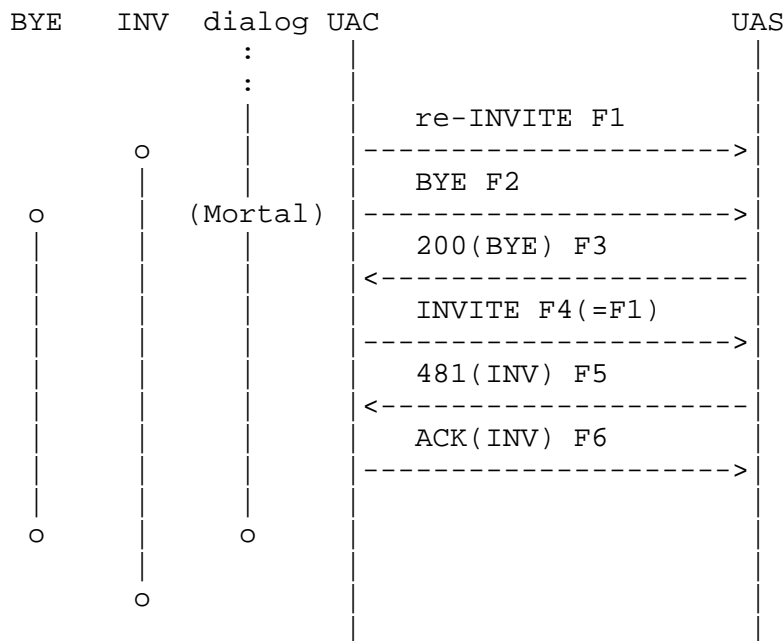
For the UAC, the INVITE client transaction begins at the point F1 is
sent. The UAC sends BYE (F2) immediately after F1. This is a
legitimate behavior. (Usually, the usage of each SIP method is
independent, for BYE and others. However, it should be noted that it
is prohibited to send a request with an SDP offer while the previous
offer is in progress.)

After that, F2 triggers the BYE client transaction. At the same
time, the dialog state transitions to the Mortal state and then only
a BYE or a response to a BYE 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 transitioned to the
Terminated state. As is mentioned above, the dialog and the
transaction behave independently each other. Therefore, the
transaction handling has to be continued even though the dialog has
moved to the Terminated state.

Note: As noted in Section 3.1.4, implementation issues are outside
the scope of this document, but the following tip is provided for
avoiding race conditions of this type. The UAC can delay sending BYE
F2 until the re-INVITE transaction F1 completes. Implementors can
decouple the actions of the user (e.g., hanging up) from the actions
of the protocol (the sending of BYE F2), so that the UA can behave
like this. In this case, it is the implementor's choice as to how
long to wait. In most cases, such an implementation may be useful to
prevent this case. This document expresses no preference about
whether or not they should wait for an ACK to be delivered. After
considering the impact on user experience, implementors should decide
whether or not to wait for a while, because the user experience
depends on the implementation and has no direct bearing on protocol
behavior.

Next, the UAS's state is shown below.

```
UAC                           UAS dialog  INV    BYE
 |                             |      :
 |                             |      :
 |    re-INVITE F1             |      |
 |-------------->x             |      |
 |    BYE F2                   |      |
 |---------------------->|     |               o
 |    200(BYE) F3        | (Mortal)             |
 |<----------------------|     |               |<-Start Timer J
 |    INVITE F4(=F1)     |     |      |         |
 |---------------------->|     |      o         |
 |    4xx/5xx(INV) F5    |      o     |         o
 |<----------------------|     |      |
 |    ACK(INV) F6        |     |      |
 |---------------------->|     |      |<-Start Timer I
 |                       |     |      |
 |                       |     |      |
 |                       |     |      o
 |                       |     |
```
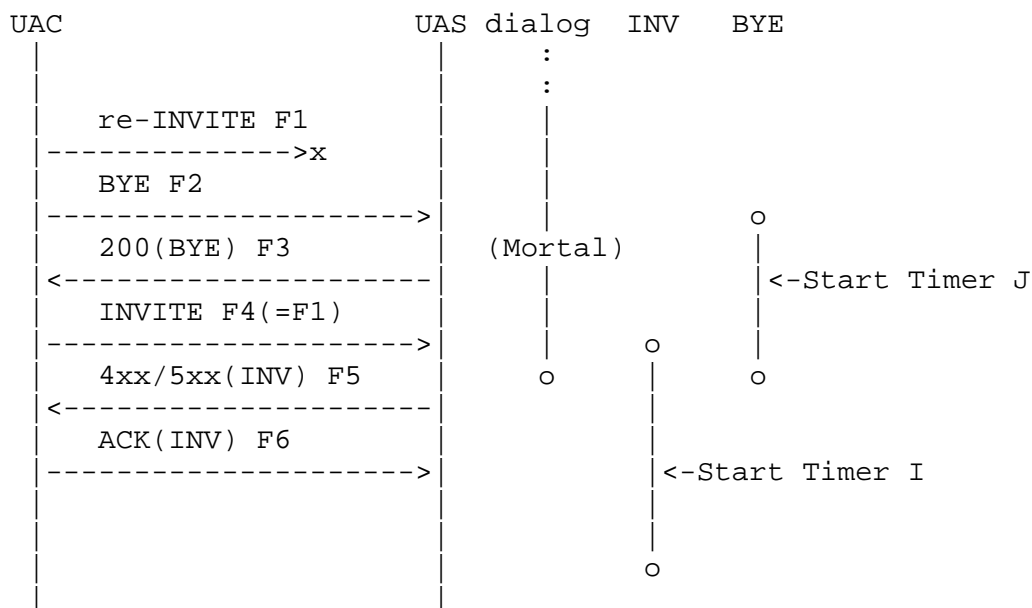
For the UAS, it can be considered that packet F1 is lost or delayed
(here, the behavior is explained for the case that the UAS receives
F2 BYE before F1 INVITE). Therefore, F2 triggers the BYE transaction

for the UAS, and simultaneously the dialog moves to the Mortal state.
Then, upon the reception of F4, the INVITE server transaction begins.
(It is permitted to start the INVITE server transaction in the Mortal
state. The INVITE server transaction begins to handle the received
SIP request regardless of the dialog state.) The UAS's TU sends an
appropriate error response for the F4 INVITE, either 481 (because the
TU knows that the dialog that matches 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 INVITE message F4 (and F1) is a mid-dialog
request. Mid-dialog requests have a To tag. It should be noted that
the 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 that indirectly impact the
   dialog state transition in the Early state. CANCEL does not have any
   influence on the UAC's dialog state. However, the request has an
   indirect influence on the dialog state transition because it has a
   significant effect on ini-INVITE. For the UAS, the CANCEL request
   has more direct effects on the dialog than on the sending of a CANCEL
   by the UAC, because it can be a trigger to send the 487 response.
   Figure 3 explains the 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 Figures 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 handled virtually just for
   the ease of understanding of the UA's behavior when it sends and
   receives CANCEL.

```
                      +-------------+
                      | Preparative |---+
                      +-------------+   |
                        :   | 1xx(s)    |
                        :   V           |
                        : +-------+     | 2xx(s)
                        : | Early |-----+------+
                        : +-------+            |
                        :    :                 V
                        :    :           +-----------+
                        :    :           | Confirmed |<...
                        :....:           +-----------+   :
                         :                    | :        :
                         :             BYE(r) | :        :
                         : CANCEL(r)          | :.......:
                         V                    |    CANCEL(r)
                       .............          |
                       : Cancelled :          |
                       :...........:          |
                         | 487(s)             |
                         |                     |
                      +--------------------+   |
                             |
                             V
                    +------------+
                    | Terminated |
                    +------------+
```
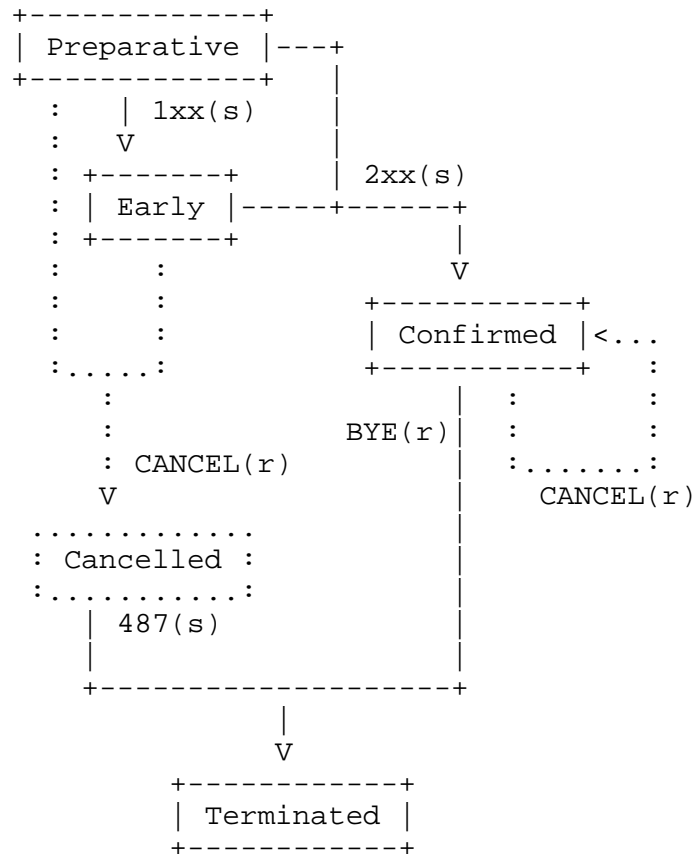
                 Figure 3: CANCEL flow diagram for UAS

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

   The first behavior is when the UAS receives a CANCEL in the Early
   state. In this case, the UAS immediately sends a 487 for the INVITE,
   and the dialog transitions to the Terminated state.

   The other is the case in which the UAS receives a CANCEL while in the
   Confirmed state. In this case, the dialog state transition does not
   occur, because the UAS has already sent a final response to the
   INVITE to which the CANCEL is targeted. (Note that, because of the
   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 transition until it
   actually receives a BYE.)

Appendix D.   Notes on the Request in the Mortal State

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

   In the Mortal state, only a BYE can be accepted, and the other
   messages in the INVITE dialog usage are responded to 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 RFC 5057 [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 a TU. However, the ACK for 2xx and the ACK for error responses
   are both 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, the UA needs to keep the INVITE
   dialog usage until the Mortal state finishes, so that it is able to
   send 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 receipt of the 2xx,
   to cope with the possible 2xx retransmission. (The duration of the
   2xx retransmission is 64*T1, so the UA needs to be prepared to handle
   the retransmission for this duration.) However, the UA shall send an
   error response to other requests, since the INVITE dialog usage in
   the Mortal state is kept only for the sending of ACK for 2xx.

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

Appendix E.   Forking and Receiving New To Tags

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

   When an INVITE is forked inside a SIP network, there is a possibility
   that the TU receives multiple responses to the ini-INVITE with
   differing To tags (see Sections 12.1, 13.1, 13.2.2.4, 16.7, 19.3,

etc., of RFC 3261 [1]). If the TU receives multiple 1xx responses
with different To tags, 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 transitions 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 that do
not transition to the Established state are terminated (see Section
13.2.2.4 of RFC 3261 [1]). 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
different To tags are received, and then a 200 response for one of
the early dialogs (dialog A) is received. Dotted lines (..) in the
sequences are auxiliary lines to represent the influence on dialog B.

```
                                  UAC
                    dialog(A)      |    INVITE F1
                      Pre o        |------------------------>
                          |        |    100 F2
                          |        |<------------------------
                          |        |    180(To tag=A) F3
                    Ear   |        |<------------------------
              dialog(B)   |        |
            forked new DSM|        |    180(To tag=B) F4
              Ear o.........|..........|<------------------------
                  |        |        |
                  |        |        |    200(A) F5
    terminate->|.....Mora |..........|<------------------------
       early   |        ^ |        |    ACK(A) F6
       media   |    Est | |        |------------------------>
                  |        | |        |
                  |        | |64*T1   |
                  |        | |(13.2.2.4 of RFC 3261 [1])
                  |        | |        |
                  |        | |        |
                  |        | V        |
            o.........|.(terminate INVITE transaction)
            terminated   |        |
             dialog(B)   |        |
                          |        |
```
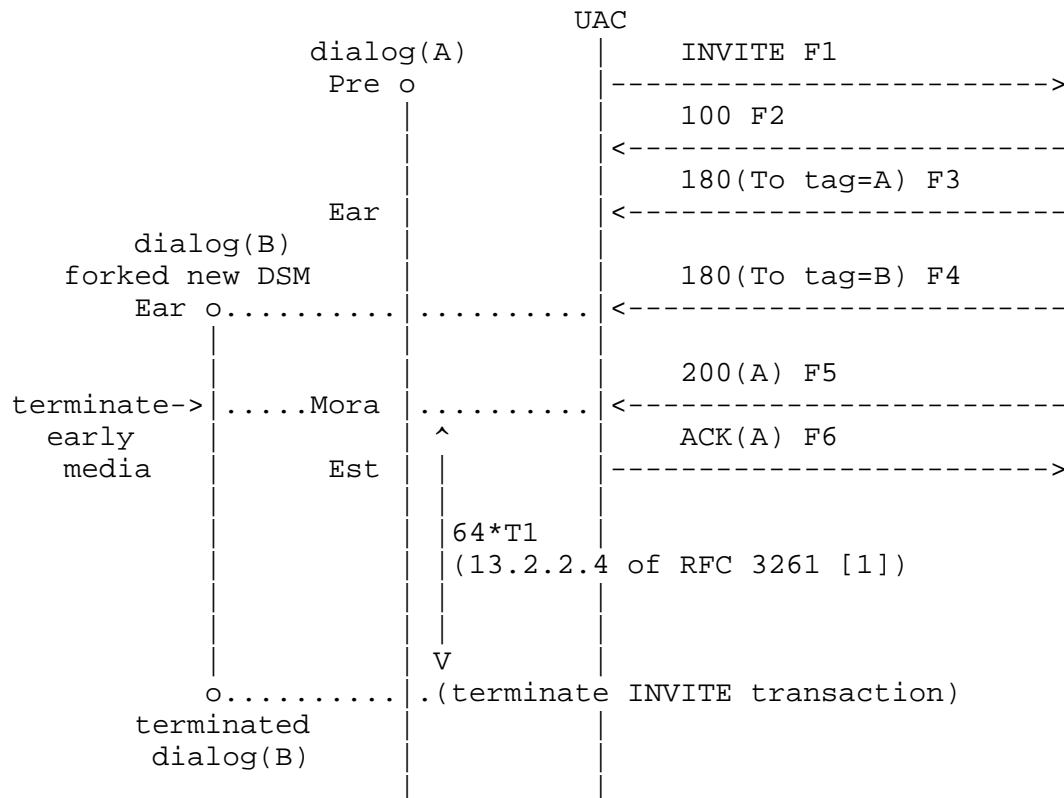
Figure 4: Receiving 1xx responses with different To tags

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)), the DSM forks and the early dialog B is generated.
64*T1 seconds later, dialog A receives a 200 OK response. Dialog B,
which does not transition to the Established state, terminates.

Next, the behavior of a TU that receives multiple 2xx responses with
different To tags is explained. When a mortal early dialog that did
not match the first 2xx response that the TU received receives
another 2xx response that matches its To tag before the 64*T1 INVITE
transaction timeout, its DSM transitions 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 sends an ACK and,
immediately after, the TU usually sends a BYE to terminate the DSM.
(In special cases, e.g., if 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 typical
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.

```
                                     UAC
                   dialog(A)          |   INVITE F1
                     Pre o            |---------------------->
                         |            |   100 F2
                         |            |<---------------------
                         |            |   180(To tag=A) F3
        dialog(B)    Ear |            |<---------------------
      forked new DSM     |            |   180(To tag=B) F4
           Ear o.........|.........|  |<---------------------
              |          |            |
              |          |            |   200(A) F5
  terminate->|.....Mora  |.........|  |<---------------------
     early    |          |     ^      |   ACK(A) F6
      media   |      Est |     |      |---------------------->
              |          |     |      |
              |          |    |64*T1  |
              |          |     |      |   200(B) F7
       Mora  |..........|.|........|  |<---------------------
              |          |  |         |   ACK(B) F8
        Est  |..........|.|........|  |---------------------->
              |          |  |         |   BYE(B) F9
       Mort  |..........|.|........|  |---------------------->
        ^     |          |  |         |   200(B) F10
        |     |          |  |         |<---------------------
        |     |          |  |         |
        |Timer K         |  |         |
        |     |          |  | V       |
        |     |          |  | (terminate INVITE transaction)
        V     |          |            |
      Morg o            |            |
              |            |            |
```
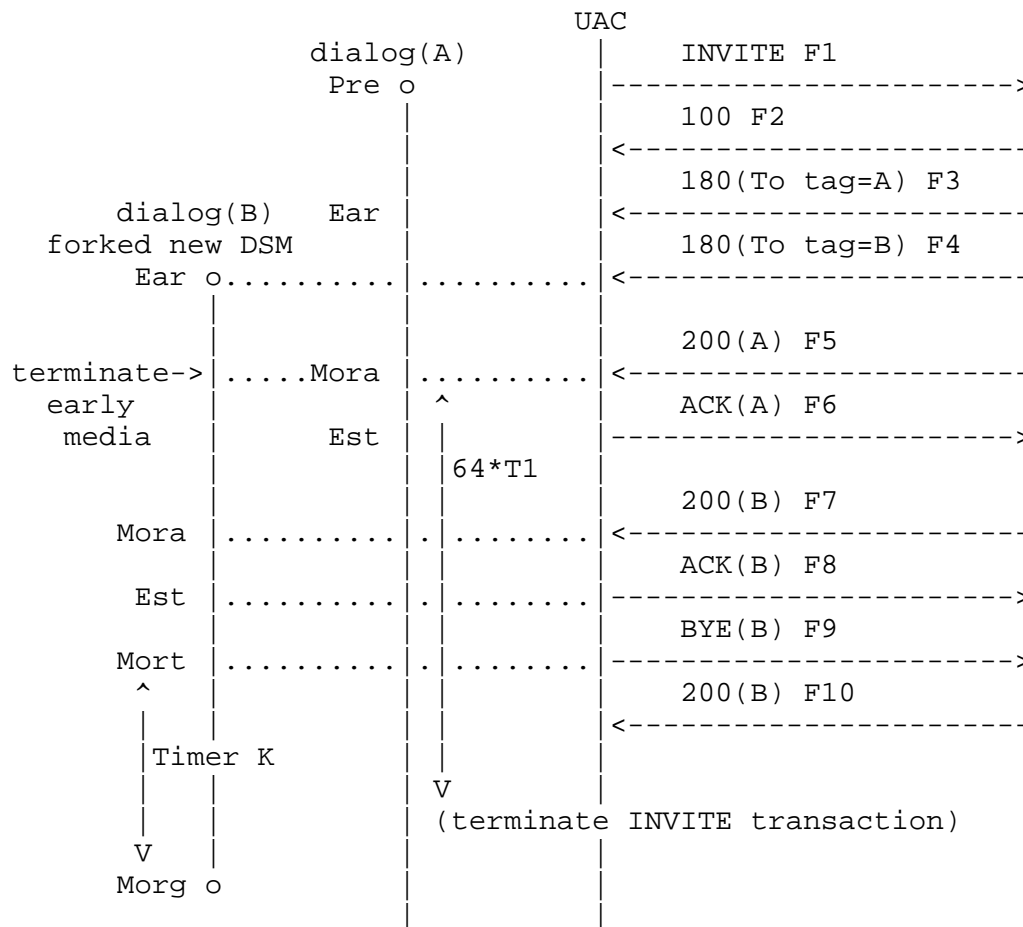
         Figure 5: Receiving 1xx and 2xx responses with different To tags

Below is an example sequence when a TU receives multiple 200
responses with different To tags before the 64*T1 timeout of the
INVITE transaction in the absence of a provisional response. Even
though a TU does not receive a provisional response, the TU needs to

   process the 2xx responses (see Section 13.2.2.4 of RFC 3261 [1]). 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., if a UA intends to
   establish multiple dialogs, the TU may not send the BYE.)

```
                                         UAC
              dialog(A)                   |     INVITE F1
                 Pre o                    |----------------------->
                     |                    |     100 F2
                     |                    |<----------------------
                     |                    |     180(To tag=A) F3
                 Ear |                    |<----------------------
                     |                    |
                     |                    |     200(A) F4
                Mora |..........|<----------------------
                     |    ^               |     ACK(A) F5
                 Est |    |               |----------------------->
                     |    |               |
       dialog(B)     |    |64*T1          |
       forked new DSM|    |               |     200(To tag=B) F6
          Mora o..........|.|........|<----------------------
                 |        | |            |     ACK(B) F7
           Est |..........|.|........|----------------------->
                 |        | |            |     BYE(B) F8
          Mort |..........|.|........|----------------------->
           ^     |        | |            |     200(B) F9
           |     |        | |            |<----------------------
           |     |        | |            |
           |     |        | V            |
           |Timer K       |    (terminate INVITE transaction)
           |     |        |     |        |
           V     |        |     |        |
          Morg o          |     |        |
                          |     |        |
```
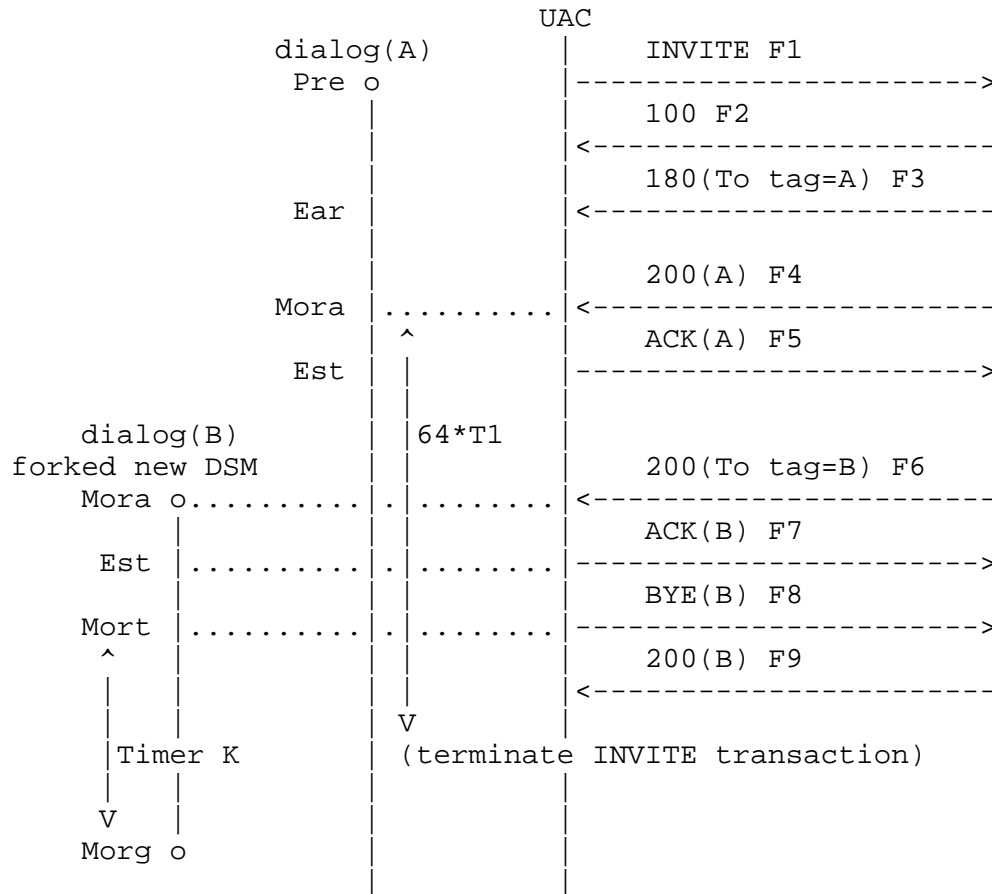
            Figure 6: Receiving 2xx responses with different To tags

   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 that contains a Require header with the
   value of 100rel. Upon receiving a provisional response with 100rel,
   the UAC establishes the early dialog (B) and sends PRACK (Provisional
   Response Acknowledgement). (Here, also, every transaction completes
   independently of others.)

As in 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,
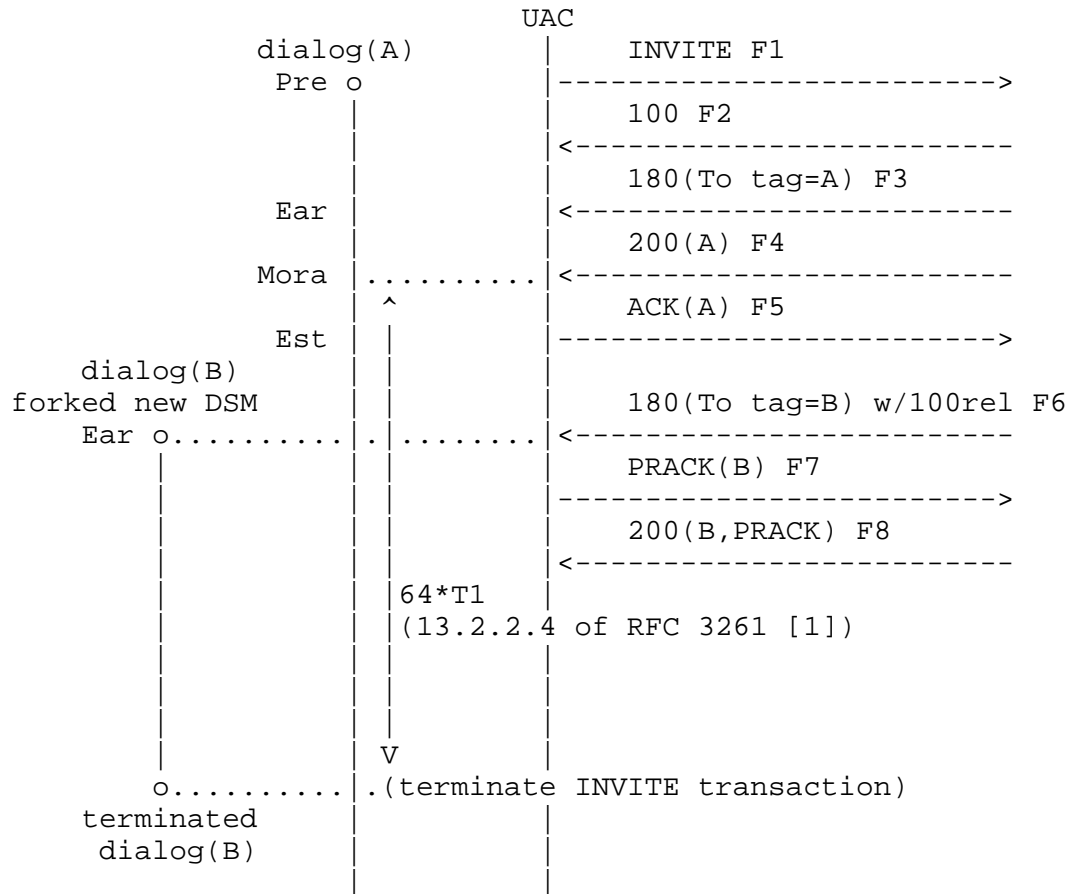not to be transmitted to the UAC).

```
                                      UAC
              dialog(A)                |    INVITE F1
                Pre o                  |------------------------>
                    |                  |    100 F2
                    |                  |<------------------------
                    |                  |    180(To tag=A) F3
                Ear |                  |<------------------------
                    |                  |    200(A) F4
               Mora |.........|<------------------------
                    |    ^     |    ACK(A) F5
                Est |    |     |------------------------>
          dialog(B) |    |     |
      forked new DSM |    |     |    180(To tag=B) w/100rel F6
           Ear o..........|.|........|<------------------------
                |    |  | |    |    PRACK(B) F7
                |    |  | |    |------------------------>
                |    |  | |    |    200(B,PRACK) F8
                |    |  | |    |<------------------------
                |    |  |64*T1 |
                |    |  |(13.2.2.4 of RFC 3261 [1])
                |    |  | |    |
                |    |  | |    |
                |    |  | |    |
                |    |  | V    |
           o..........|.(terminate INVITE transaction)
          terminated      |    |
           dialog(B)       |    |
                           |    |
```

     Figure 7: Receiving 1xx responses with different To tags
  when using the mechanism for reliable provisional responses (100rel)

Authors' Addresses

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

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

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

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

   Yasushi Suzuki
   NTT Corporation
   9-11, Midori-cho 3-Chome
   Musashino-shi, Tokyo 180-8585
   JP

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

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

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

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

   EMail: pkyzivat@cisco.com