Network Working Group
Request for Comments: 3725
BCP: 85
Category: Best Current Practice

J. Rosenberg
dynamicsoft
J. Peterson
Neustar
H. Schulzrinne
Columbia University
G. Camarillo
Ericsson
April 2004

Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)

Status of this Memo

Copyright Notice

Abstract

Third party call control refers to the ability of one entity to create a call in which communication is actually between other parties. Third party call control is possible using the mechanisms specified within the Session Initiation Protocol (SIP). However, there are several possible approaches, each with different benefits and drawbacks. This document discusses best current practices for the usage of SIP for third party call control.

Table of Contents

1. Introduction

In the traditional telephony context, third party call control allows one entity (which we call the controller) to set up and manage a communications relationship between two or more other parties. Third party call control (referred to as 3pcc) is often used for operator services (where an operator creates a call that connects two participants together) and conferencing.

Similarly, many SIP services are possible through third party call control. These include the traditional ones on the PSTN, but also new ones such as click-to-dial. Click-to-dial allows a user to click on a web page when they wish to speak to a customer service representative. The web server then creates a call between the user and a customer service representative. The call can be between two phones, a phone and an IP host, or two IP hosts.

Third party call control is possible using only the mechanisms specified within RFC 3261 [1]. Indeed, many different call flows are possible, each of which will work with SIP compliant user agents. However, there are benefits and drawbacks to each of these flows. The usage of third party call control also becomes more complex when aspects of the call utilize SIP extensions or optional features of SIP. In particular, the usage of RFC 3312 [2] (used for coupling of signaling to resource reservation) with third party call control is non-trivial, and is discussed in Section 9. Similarly, the usage of early media (where session data is exchanged before the call is accepted) with third party call control is not trivial; both of them specify the way in which user agents generate and respond to SDP, and it is not clear how to do both at the same time. This is discussed further in Section 8.

This document serves as a best current practice for implementing third party call control without usage of any extensions specifically designed for that purpose. Section 4 presents the known call flows that can be used to achieve third party call control, and provides guidelines on their usage. Section 9 discusses the interactions of RFC 3312 [2] with third party call control. Section 8 discusses the interactions of early media with third party call control. Section 10 provides example applications that make usage of the flows recommended here.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [3] and indicate requirement levels for compliant implementations.

3. Definitions

The following terms are used throughout this document:

3pcc: Third Party Call Control, which refers to the general ability to manipulate calls between other parties.

Controller: A controller is a SIP User Agent that wishes to create a session between two other user agents.

4. 3pcc Call Establishment

The primary primitive operation of third party call control is the establishment of a session between participants A and B. Establishment of this session is orchestrated by a third party, referred to as the controller.

This section documents three call flows that the controller can utilize in order to provide this primitive operation.

4.1. Flow I

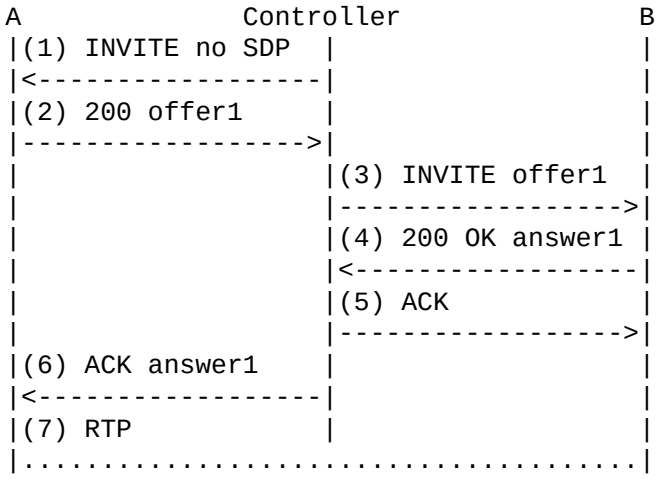


Figure 1

The call flow for Flow I is shown in Figure 1. The controller first sends an INVITE A (1). This INVITE has no session description. A's phone rings, and A answers. This results in a 200 OK (2) that contains an offer [4]. The controller needs to send its answer in the ACK, as mandated by [1]. To obtain the answer, it sends the offer it got from A (offer1) in an INVITE to B (3). B's phone rings. When B answers, the 200 OK (4) contains the answer to this offer, answer1. The controller sends an ACK to B (5), and then passes answer1 to A in an ACK sent to it (6). Because the offer was generated by A, and the answer generated by B, the actual media session is between A and B. Therefore, media flows between them (7).

This flow is simple, requires no manipulation of the SDP by the controller, and works for any media types supported by both endpoints. However, it has a serious timeout problem. User B may not answer the call immediately. The result is that the controller cannot send the ACK to A right away. This causes A to retransmit the 200 OK response periodically. As specified in RFC 3261 Section 13.3.1.4, the 200 OK will be retransmitted for 64*T1 seconds. If an ACK does not arrive by then, the call is considered to have failed. This limits the applicability of this flow to scenarios where the controller knows that B will answer the INVITE immediately.

4.2. Flow II

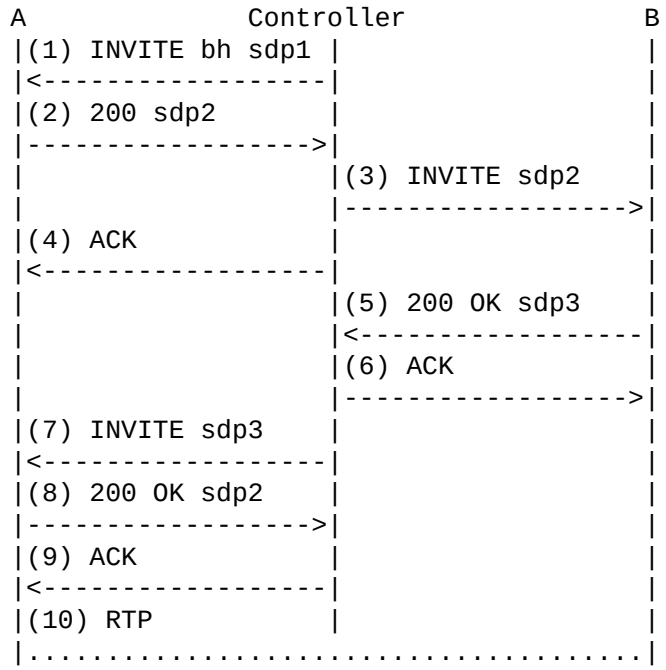


Figure 2

An alternative flow, Flow II, is shown in Figure 2. The controller first sends an INVITE to user A (1). This is a standard INVITE, containing an offer (sdp1) with a single audio media line, one codec, a random port number (but not zero), and a connection address of 0.0.0.0. This creates an initial media stream that is "black holed", since no media (or RTCP packets [8]) will flow from A. The INVITE causes A's phone to ring.

> Note that the usage of 0.0.0.0, though recommended by RFC 3264, has numerous drawbacks. It is anticipated that a future specification will recommend usage of a domain within the .invalid DNS top level domain instead of the 0.0.0.0 IP address. As a result, implementors are encouraged to track such developments once they arise.

When A answers (2), the 200 OK contains an answer, sdp2, with a valid address in the connection line. The controller sends an ACK (4). It then generates a second INVITE (3). This INVITE is addressed to user B, and it contains sdp2 as the offer to B. Note that the role of sdp2 has changed. In the 200 OK (message 2), it was an answer, but in the INVITE, it is an offer. Fortunately, all valid answers are valid

initial offers. This INVITE causes B's phone to ring. When it answers, it generates a 200 OK (5) with an answer, sdp3. The controller then generates an ACK (6). Next, it sends a re-INVITE to A (7) containing sdp3 as the offer. Once again, there has been a reversal of roles. sdp3 was an answer, and now it is an offer. Fortunately, an answer to an answer recast as an offer is, in turn, a valid offer. This re-INVITE generates a 200 OK (8) with sdp2, assuming that A doesn't decide to change any aspects of the session as a result of this re-INVITE. This 200 OK is ACKed (9), and then media can flow from A to B. Media from B to A could already start flowing once message 5 was sent.

This flow has the advantage that all final responses are immediately ACKed. It therefore does not suffer from the timeout and message inefficiency problems of flow 1. However, it too has troubles. First off, it requires that the controller know the media types to be used for the call (since it must generate a "blackhole" SDP, which requires media lines). Secondly, the first INVITE to A (1) contains media with a 0.0.0.0 connection address. The controller expects that the response contains a valid, non-zero connection address for A. However, experience has shown that many UAs respond to an offer of a 0.0.0.0 connection address with an answer containing a 0.0.0.0 connection address. The offer-answer specification [4] explicitly tells implementors not to do this, but at the time of publication of this document, many implementations still did. If A should respond with a 0.0.0.0 connection address in sdp2, the flow will not work.

However, the most serious flaw in this flow is the assumption that the 200 OK to the re-INVITE (message 8) contains the same SDP as in message 2. This may not be the case. If it is not, the controller needs to re-INVITE B with that SDP (say, sdp4), which may result in getting a different SDP, sdp5, in the 200 OK from B. Then, the controller needs to re-INVITE A again, and so on. The result is an infinite loop of re-INVITEs. It is possible to break this cycle by having very smart UAs which can return the same SDP whenever possible, or really smart controllers that can analyze the SDP to determine if a re-INVITE is really needed. However, we wish to keep this mechanism simple, and avoid SDP awareness in the controller. As a result, this flow is not really workable. It is therefore NOT RECOMMENDED.

4.3. Flow III

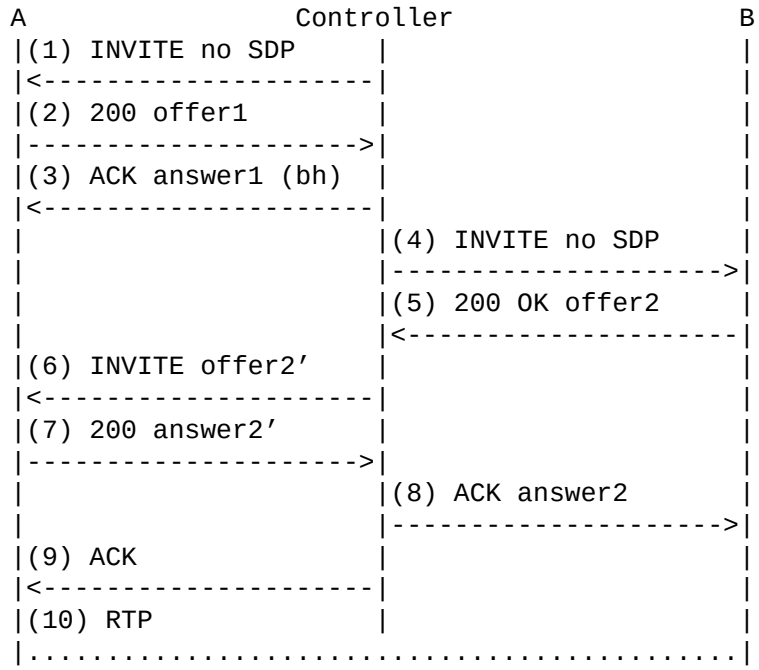


Figure 3

A third flow, Flow III, is shown in Figure 3.

First, the controller sends an INVITE (1) to user A without any SDP (which is good, since it means that the controller doesn't need to assume anything about the media composition of the session). A's phone rings. When A answers, a 200 OK is generated (2) containing its offer, offer1. The controller generates an immediate ACK containing an answer (3). This answer is a "black hole" SDP, with its connection address equal to 0.0.0.0.

The controller then sends an INVITE to B without SDP (4). This causes B's phone to ring. When they answer, a 200 OK is sent, containing their offer, offer2 (5). This SDP is used to create a re-INVITE back to A (6). That re-INVITE is based on offer2, but may need to be reorganized to match up media lines, or to trim media lines. For example, if offer1 contained an audio and a video line, in that order, but offer2 contained just an audio line, the controller would need to add a video line to the offer (setting its port to zero) to create offer2'. Since this is a re-INVITE, it should complete quickly in the general case. That's good, since user B is retransmitting their 200 OK, waiting for an ACK. The SDP in the

200 OK (7) from A, answer2', may also need to be reorganized or trimmed before sending it an the ACK to B (8) as answer2. Finally, an ACK is sent to A (9), and then media can flow.

This flow has many benefits. First, it will usually operate without any spurious retransmissions or timeouts (although this may still happen if a re-INVITE is not responded to quickly). Secondly, it does not require the controller to guess the media that will be used by the participants.

There are some drawbacks. The controller does need to perform SDP manipulations. Specifically, it must take some SDP, and generate another SDP which has the same media composition, but has connection addresses equal to 0.0.0.0. This is needed for message 3. Secondly, it may need to reorder and trim SDP X, so that its media lines match up with those in some other SDP, Y. Thirdly, the offer from B (offer2) may have no codecs or media streams in common with the offer from A (offer 1). The controller will need to detect this condition, and terminate the call. Finally, the flow is far more complicated than the simple and elegant Flow I (Figure 1).

4.4. Flow IV

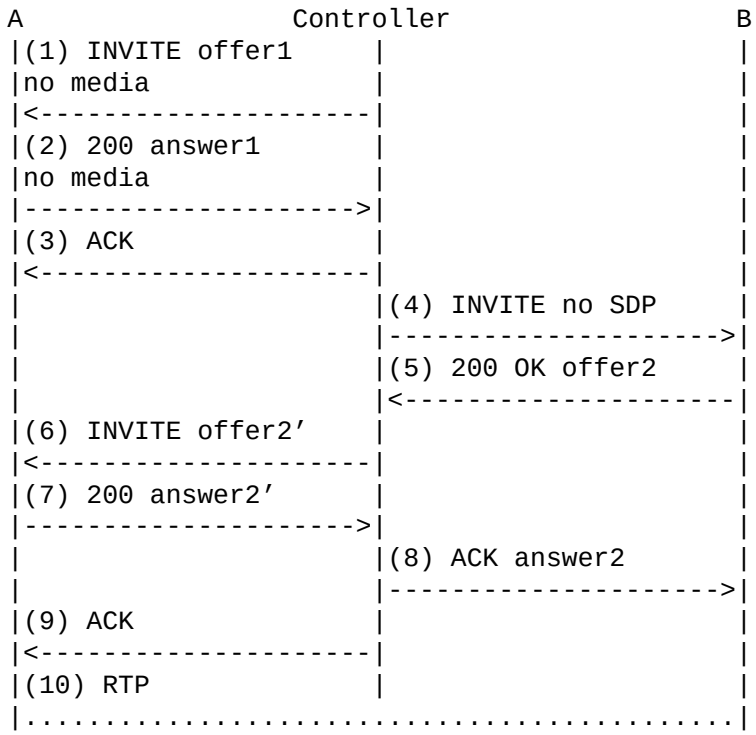


Figure 4

Flow IV shows a variation on Flow III that reduces its complexity. The actual message flow is identical, but the SDP placement and construction differs. The initial INVITE (1) contains SDP with no media at all, meaning that there are no m lines. This is valid, and implies that the media makeup of the session will be established later through a re-INVITE [4]. Once the INVITE is received, user A is alerted. When they answer the call, the 200 OK (2) has an answer with no media either. This is acknowledged by the controller (3). The flow from this point onwards is identical to Flow III. However, the manipulations required to convert offer2 to offer2', and answer2' to answer2, are much simpler. Indeed, no media manipulations are needed at all. The only change that is needed is to modify the origin lines, so that the origin line in offer2' is valid based on the value in offer1 (validity requires that the version increments by one, and that the other parameters remain unchanged).

There are some limitations associated with this flow. First, user A will be alerted without any media having been established yet. This means that user A will not be able to reject or accept the call based on its media composition. Secondly, both A and B will end up answering the call (i.e., generating a 200 OK) before it is known whether there is compatible media. If there is no media in common, the call can be terminated later with a BYE. However, the users will have already been alerted, resulting in user annoyance and possibly resulting in billing events.

5. Recommendations

Flow I (Figure 1) represents the simplest and the most efficient flow. This flow SHOULD be used by a controller if it knows with certainty that user B is actually an automata that will answer the call immediately. This is the case for devices such as media servers, conferencing servers, and messaging servers, for example. Since we expect a great deal of third party call control to be to automata, special casing in this scenario is reasonable.

For calls to unknown entities, or to entities known to represent people, it is RECOMMENDED that Flow IV (Figure 4) be used for third party call control. Flow III MAY be used instead, but it provides no additional benefits over Flow IV. However, Flow II SHOULD NOT be used, because of the potential for infinite ping-ponging of re-INVITEs.

Several of these flows use a "black hole" connection address of 0.0.0.0. This is an IPv4 address with the property that packets sent to it will never leave the host which sent them; they are just

discarded. Those flows are therefore specific to IPv4. For other network or address types, an address with an equivalent property SHOULD be used.

In most cases, including the recommended flows, user A will hear silence while the call to B completes. This may not always be ideal. It can be remedied by connecting the caller to a music-on-hold source while the call to B occurs.

6. Error Handling

There are numerous error cases which merit discussion.

With all of the call flows in Section 4, one call is established to A, and then the controller attempts to establish a call to B. However, this call attempt may fail, for any number of reasons. User B might be busy (resulting in a 486 response to the INVITE), there may not be any media in common, the request may time out, and so on. If the call attempt to B should fail, it is RECOMMENDED that the controller send a BYE to A. This BYE SHOULD include a Reason header [5] which carries the status code from the error response. This will inform A of the precise reason for the failure. The information is important from a user interface perspective. For example, if A was calling from a black phone, and B generated a 486, the BYE will contain a Reason code of 486, and this could be used to generate a local busy signal so that A knows that B is busy.

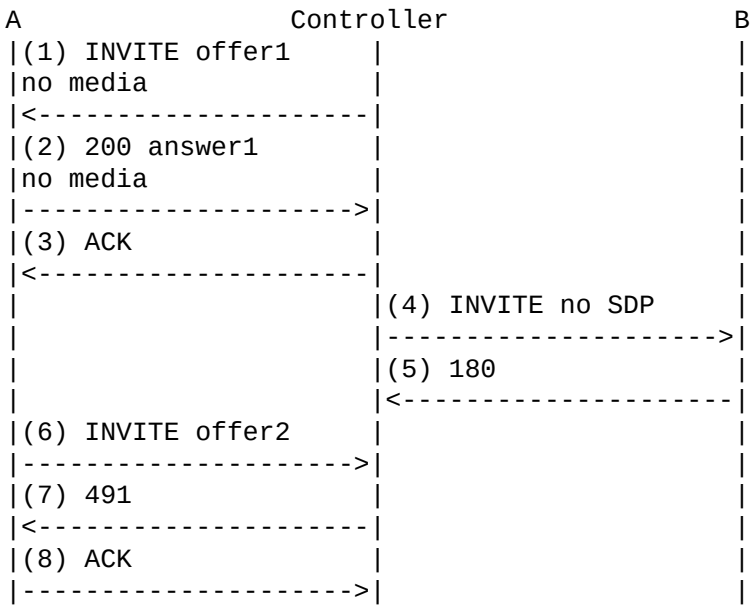


Figure 5

Another error condition worth discussion is shown in Figure 5. After the controller establishes the dialog with A (messages 1-3) it attempts to contact B (message 4). Contacting B may take some time. During that interval, A could possibly attempt a re-INVITE, providing an updated offer. However, the controller cannot pass this offer on to B, since it has an INVITE transaction pending with it. As a result, the controller needs to reject the request. It is RECOMMENDED that a 491 response be used. The situation here is similar to the glare condition described in [1], and thus the same error handling is sensible. However, A is likely to retry its request (as a result of the 491), and this may occur before the exchange with B is completed. In that case, the controller would respond with another 491.

7. Continued Processing

Once the calls are established, both participants believe they are in a single point-to-point call. However, they are exchanging media directly with each other, rather than with the controller. The controller is involved in two dialogs, yet sees no media.

Since the controller is still a central point for signaling, it now has complete control over the call. If it receives a BYE from one of the participants, it can create a new BYE and hang up with the other participant. This is shown in Figure 6.

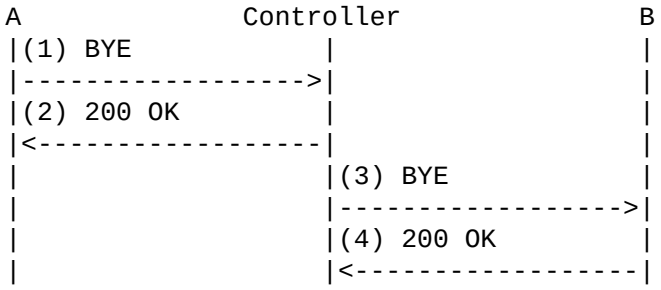


Figure 6

Similarly, if it receives a re-INVITE from one of the participants, it can forward it to the other participant. Depending on which flow was used, this may require some manipulation on the SDP before passing it on.

However, the controller need not "proxy" the SIP messages received from one of the parties. Since it is a Back-to-Back User Agent (B2BUA), it can invoke any signaling mechanism on each dialog, as it sees fit. For example, if the controller receives a BYE from A, it can generate a new INVITE to a third party, C, and connect B to that

participant instead. A call flow for this is shown in Figure 7, assuming the case where C represents an end user, not an automata. Note that it is just Flow IV.

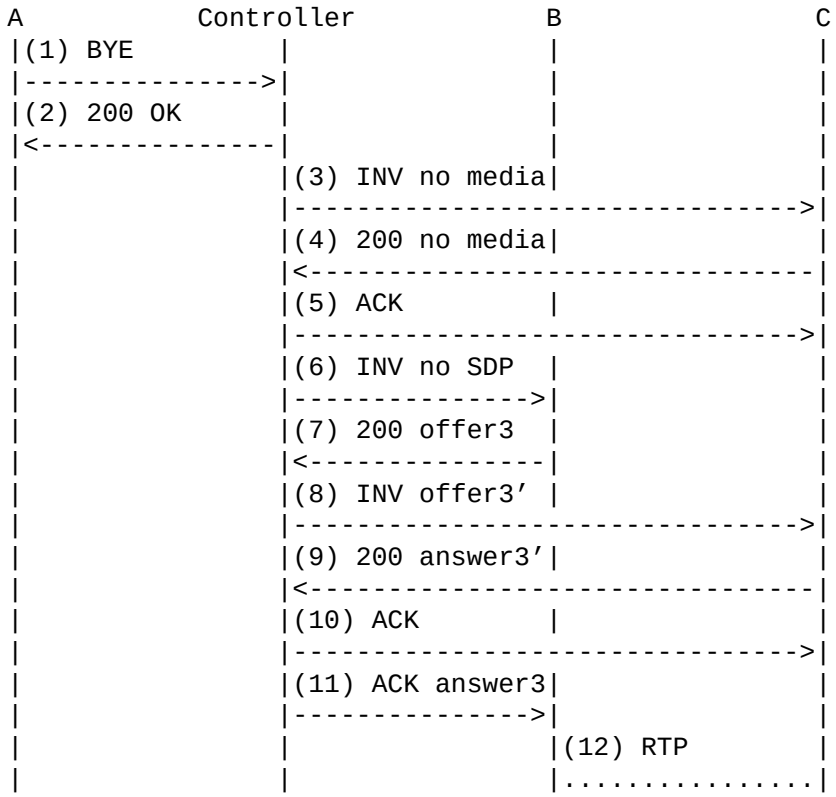


Figure 7

From here, new parties can be added, removed, transferred, and so on, as the controller sees fit. In many cases, the controller will be required to modify the SDP exchanged between the participants in order to affect these changes. In particular, the version number in the SDP will need to be changed by the controller in certain cases. If the controller should issue an SDP offer on its own (for example, to place a call on hold), it will need to increment the version number in the SDP offer. The other participant in the call will not know that the controller has done this, and any subsequent offer it generates will have the wrong version number as far as its peer is concerned. As a result, the controller will be required to modify the version number in SDP messages to match what the recipient is expecting.

It is important to point out that the call need not have been established by the controller in order for the processing of this section to be used. Rather, the controller could have acted as a B2BUA during a call established by A towards B (or vice versa).

8. 3pcc and Early Media

Early media represents the condition where the session is established (as a result of the completion of an offer/answer exchange), yet the call itself has not been accepted. This is usually used to convey tones or announcements regarding progress of the call. Handling of early media in a third party call is straightforward.

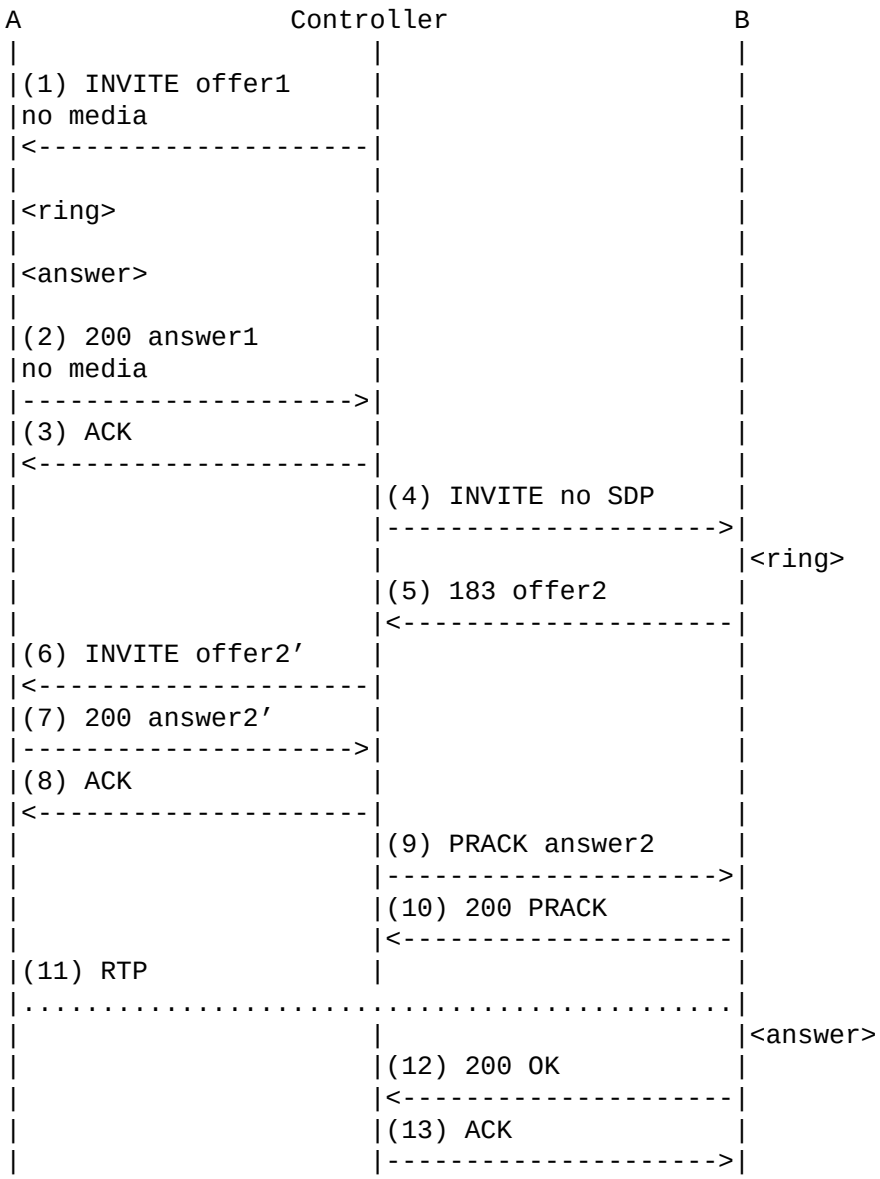


Figure 8

Figure 8 shows the case where user B generates early media before answering the call. The flow is almost identical to Flow IV from Figure 4. The only difference is that user B generates a reliable provisional response (5) [6] instead of a final response, and answer2 is carried in a PRACK (9) instead of an ACK. When party B finally does accept the call (12), there is no change in the session state, and therefore, no signaling needs to be done with user A. The controller simply ACKs the 200 OK (13) to confirm the dialog.

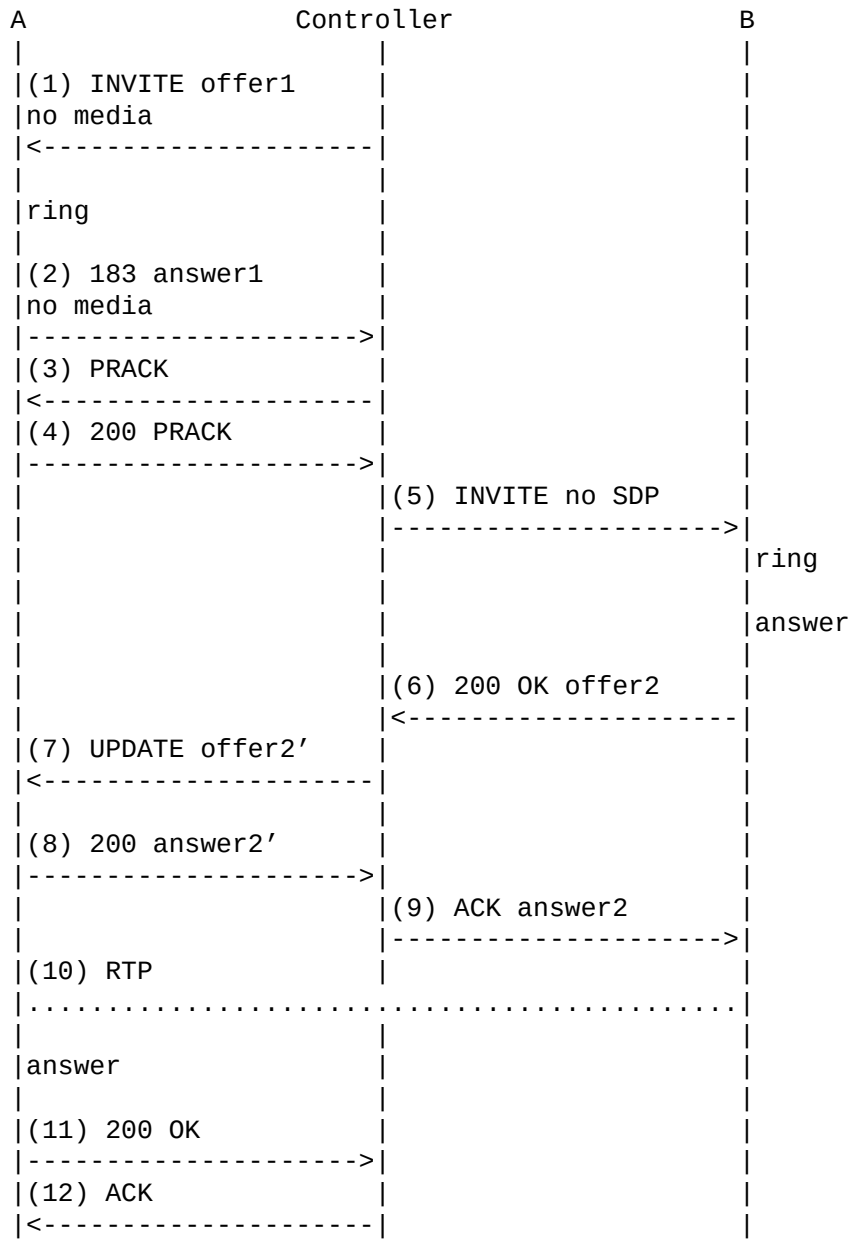


Figure 9

The case where user A generates early media is more complicated, and is shown in Figure 9. The flow is based on Flow IV. The controller sends an INVITE to user A (1), with an offer containing no media streams. User A generates a reliable provisional response (2) containing an answer with no media streams. The controller PRACKs this provisional response (3). Now, the controller sends an INVITE

without SDP to user B (5). User B's phone rings, and they answer, resulting in a 200 OK (6) with an offer, offer2. The controller now needs to update the session parameters with user A. However, since the call has not been answered, it cannot use a re-INVITE. Rather, it uses a SIP UPDATE request (7) [7], passing the offer (after modifying it to get the origin field correct). User A generates its answer in the 200 OK to the UPDATE (8). This answer is passed to user B in the ACK (9). When user A finally answers (11), there is no change in session state, so the controller simply ACKs the 200 OK (12).

Note that it is likely that there will be clipping of media in this call flow. User A is likely a PSTN gateway, and has generated a provisional response because of early media from the PSTN side. The PSTN will deliver this media even though the gateway does not have anywhere to send it, since the initial offer from the controller had no media streams. When user B answers, media can begin to flow. However, any media sent to the gateway from the PSTN up to that point will be lost.

9. Third Party Call Control and SDP Preconditions

A SIP extension has been specified that allows for the coupling of signaling and resource reservation [2]. This specification relies on exchanges of session descriptions before completion of the call setup. These flows are initiated when certain SDP parameters are passed in the initial INVITE. As a result, the interaction of this mechanism with third party call control is not obvious, and worth detailing.

9.1. Controller Initiates

In one usage scenario, the controller wishes to make use of preconditions in order to avoid the call failure scenarios documented in Section 4.4. Specifically, the controller can use preconditions in order to guarantee that neither party is alerted unless there is a common set of media and codecs. It can also provide both parties with information on the media composition of the call before they decide to accept it.

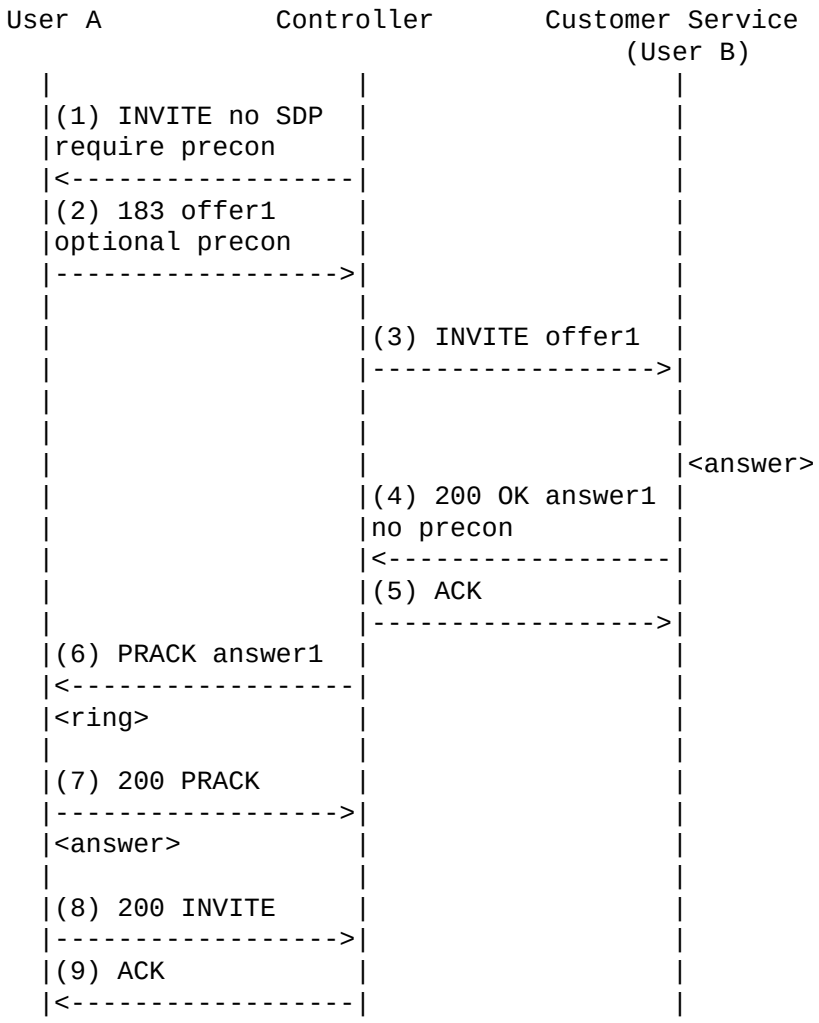


Figure 10

The flow for this scenario is shown in Figure 10. In this example, we assume that user B is an automata or agent of some sort which will answer the call immediately. Therefore, the flow is based on Flow I. The controller sends an INVITE to user A containing no SDP, but with a Require header indicating that preconditions are required. This specific scenario (an INVITE without an offer, but with a Require header indicating preconditions) is not described in [2]. It is RECOMMENDED that the UAS respond with an offer in a 1xx including the media streams it wishes to use for the call, and for each, list all preconditions it supports as optional. Of course, the user is not alerted at this time. The controller takes this offer and passes it to user B (3). User B does not support preconditions, or does, but

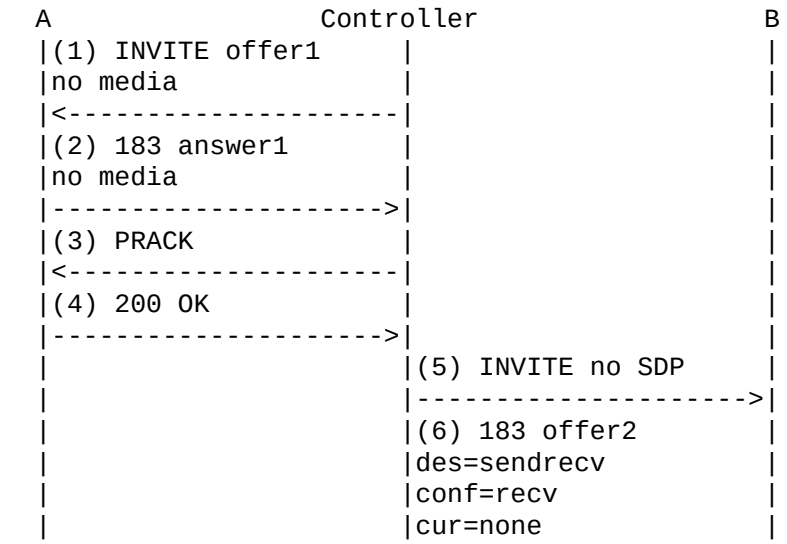
is not interested in them. Therefore, when it answers the call, the 200 OK contains an answer without any preconditions listed (4). This answer is passed to user A in the PRACK (6). At this point, user A knows that there are no preconditions actually in use for the call, and therefore, it can alert the user. When the call is answered, user A sends a 200 OK to the controller (8) and the call is complete.

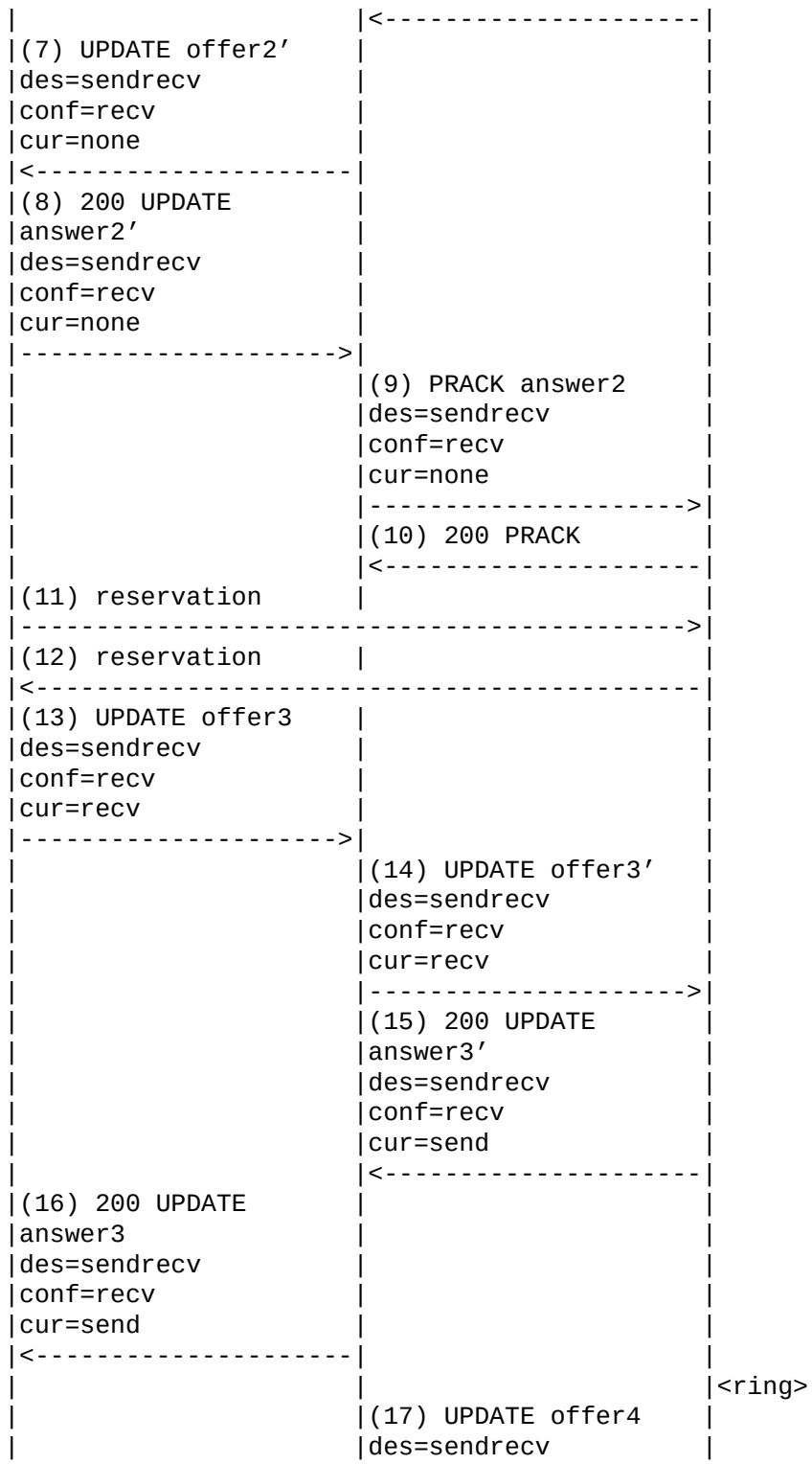
In the event that the offer generated by user A was not acceptable to user B (because of non-overlapping codecs or media, for example), user B would immediately reject the INVITE (message 3). The controller would then CANCEL the request to user A. In this situation, neither user A nor user B would have been alerted, achieving the desired effect. It is interesting to note that this property is achieved using preconditions even though it doesn't matter what specific types of preconditions are supported by user A.

It is also entirely possible that user B does actually desire preconditions. In that case, it might generate a 1xx of its own with an answer containing preconditions. That answer would still be passed to user A, and both parties would proceed with whatever measures are necessary to meet the preconditions. Neither user would be alerted until the preconditions were met.

9.2. Party A Initiates

In Section 9.1, the controller requested the use of preconditions to achieve a specific goal. It is also possible that the controller doesn't care (or perhaps doesn't even know) about preconditions, but one of the participants in the call does care. A call flow for this case is shown in Figure 11.




(7) UPDATE offer2'
des=sendrecv
conf=recv
cur=none
(8) 200 UPDATE
answer2'
des=sendrecv
conf=recv
cur=none
(9) PRACK answer2
des=sendrecv
conf=recv
cur=none
(10) 200 PRACK
(11) reservation
(12) reservation
(13) UPDATE offer3
des=sendrecv
conf=recv
cur=recv
(14) UPDATE offer3'
des=sendrecv
conf=recv
cur=recv
(15) 200 UPDATE
answer3'
des=sendrecv
conf=recv
cur=send
(16) 200 UPDATE
answer3
des=sendrecv
conf=recv
cur=send
<ring>
(17) UPDATE offer4
des=sendrecv

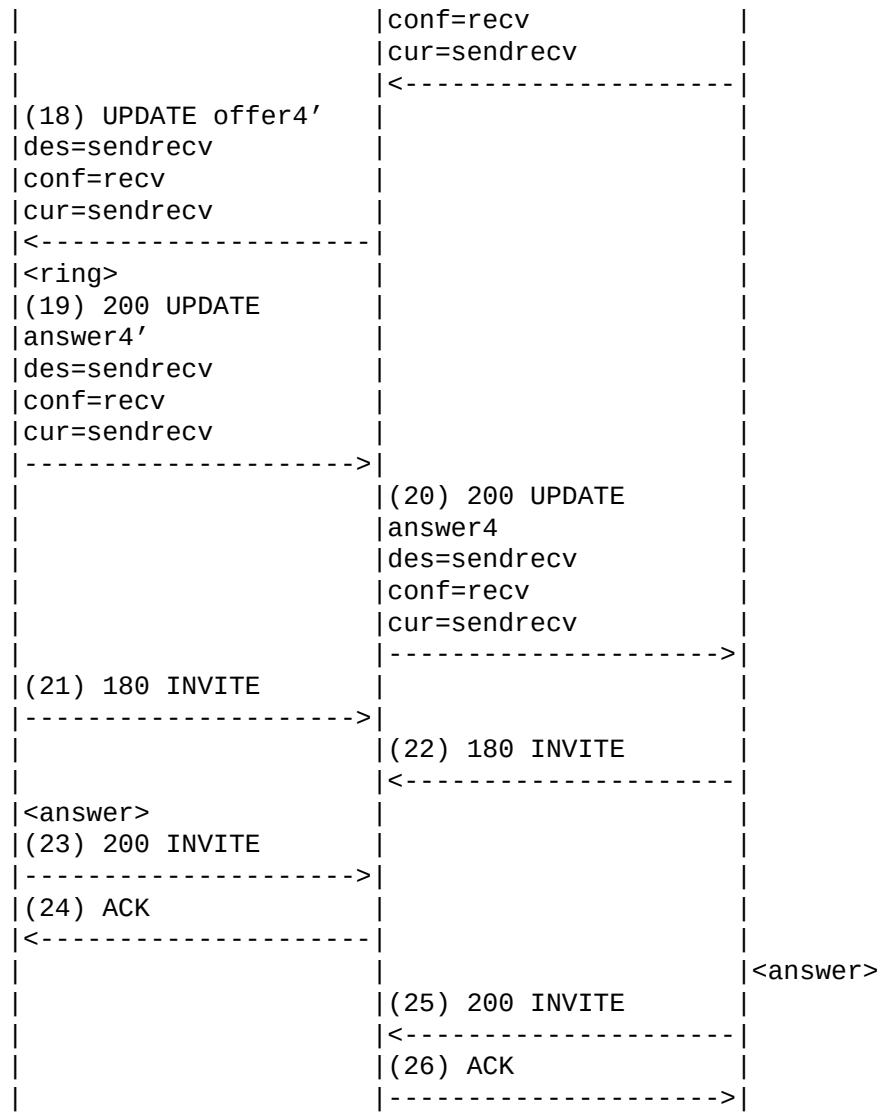


Figure 11

The controller follows Flow IV; it has no specific requirements for support of the preconditions specification [2]. Therefore, it sends an INVITE (1) with SDP that contains no media lines. User A is interested in supporting preconditions, and does not want to ring its phone until resources are reserved. Since there are no media streams in the INVITE, it can't reserve resources for media streams, and therefore it can't ring the phone until they are conveyed in a subsequent offer and then reserved. Therefore, it generates a 183 with the answer, and doesn't alert the user (2). The controller PRACKs this (3) and A responds to the PRACK (4).

At this point, the controller attempts to bring B into the call. It sends B an INVITE without SDP (5). B is interested in having preconditions for this call. Therefore, it generates its offer in a 183 that contains the appropriate SDP attributes (6). The controller passes this offer to A in an UPDATE request (7). The controller uses UPDATE because the call has not been answered yet, and therefore, it cannot use a re-INVITE. User A sees that its peer is capable of supporting preconditions. Since it desires preconditions for the call, it generates an answer in the 200 OK (8) to the UPDATE. This answer, in turn, is passed to B in the PRACK for the provisional response (9). Now, both sides perform resource reservation. User A succeeds first, and passes an updated session description in an UPDATE request (13). The controller simply passes this to A (after the manipulation of the origin field, as required in Flow IV) in an UPDATE (14), and the answer (15) is passed back to A (16). The same flow happens, but from B to A, when B's reservation succeeds (17-20). Since the preconditions have been met, both sides ring (21 and 22), and then both answer (23 and 25), completing the call.

What is important about this flow is that the controller doesn't know anything about preconditions. It merely passes the SDP back and forth as needed. The trick is the usage of UPDATE and PRACK to pass the SDP when needed. That determination is made entirely based on the offer/answer rules described in [6] and [7], and is independent of preconditions.

10. Example Call Flows

10.1. Click-to-Dial

The first application of this capability we discuss is click-to-dial. In this service, a user is browsing the web page of an e-commerce site, and would like to speak to a customer service representative. The user clicks on a link, and a call is placed to a customer service representative. When the representative picks up, the phone on the user's desk rings. When the user pick up, the customer service representative is there, ready to talk to the user.

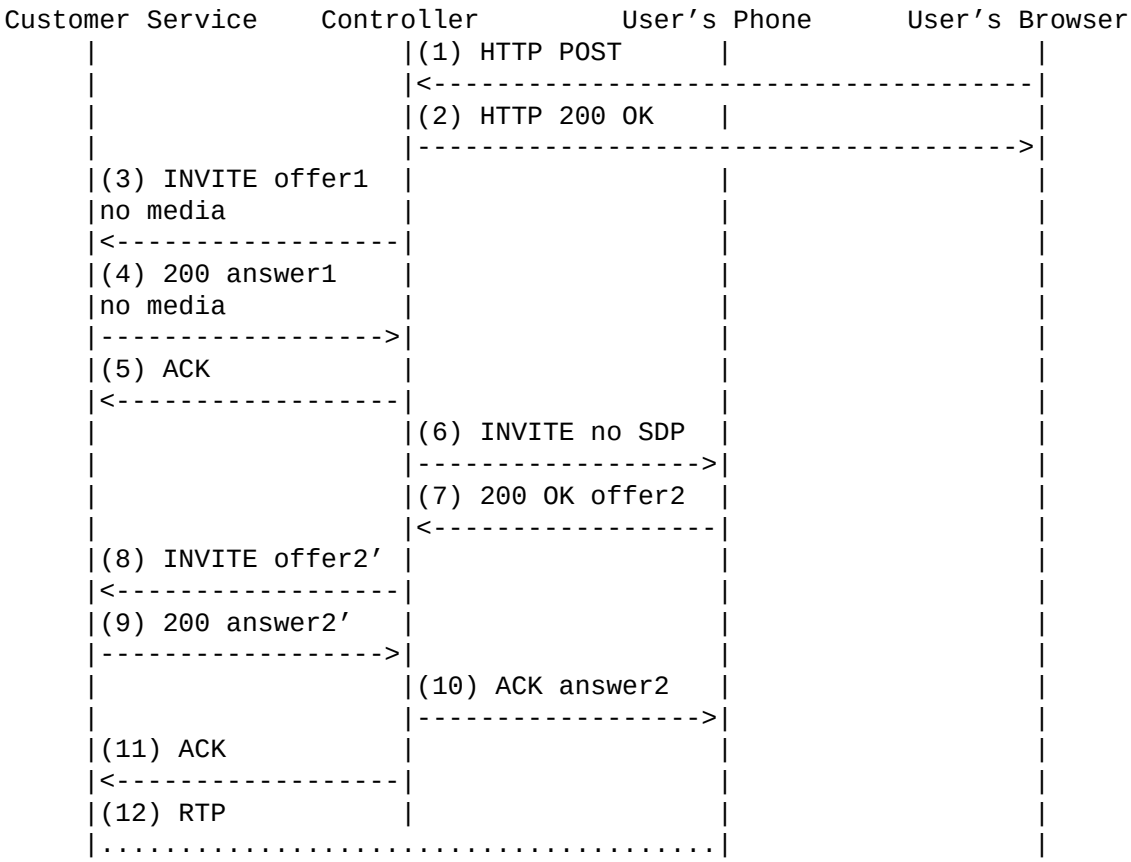


Figure 12

The call flow for this service is given in Figure 12. It is identical to that of Figure 4, with the exception that the service is triggered through an HTTP POST request when the user clicks on the link. Normally, this POST request would contain neither the number of the user or of the customer service representative. The user's number would typically be obtained by the web application from back-end databases, since the user would have presumably logged into the site, giving the server the needed context. The customer service number would typically be obtained through provisioning. Thus, the HTTP POST is actually providing the server nothing more than an indication that a call is desired.

We note that this service can be provided through other mechanisms, namely PINT [9]. However, there are numerous differences between the way in which the service is provided by PINT, and the way in which it is provided here:

- The PINT solution enables calls only between two PSTN endpoints. The solution described here allows calls between PSTN phones (through SIP enabled gateways) and native IP phones.

- When used for calls between two PSTN phones, the solution here may result in a portion of the call being routed over the Internet. In PINT, the call is always routed only over the PSTN. This may result in better quality calls with the PINT solution, depending on the codec in use and QoS capabilities of the network routing the Internet portion of the call.

- The PINT solution requires extensions to SIP (PINT is an extension to SIP), whereas the solution described here is done with baseline SIP.

- The PINT solution allows the controller (acting as a PINT client) to "step out" once the call is established. The solution described here requires the controller to maintain call state for the entire duration of the call.

10.2. Mid-Call Announcement Capability

The third party call control mechanism described here can also be used to enable mid-call announcements. Consider a service for pre-paid calling cards. Once the pre-paid call is established, the system needs to set a timer to fire when they run out of minutes. When this timer fires, we would like the user to hear an announcement which tells them to enter a credit card to continue. Once they enter the credit card info, more money is added to the pre-paid card, and the user is reconnected to the destination party.

We consider here the usage of third party call control just for playing the mid-call dialog to collect the credit card information.

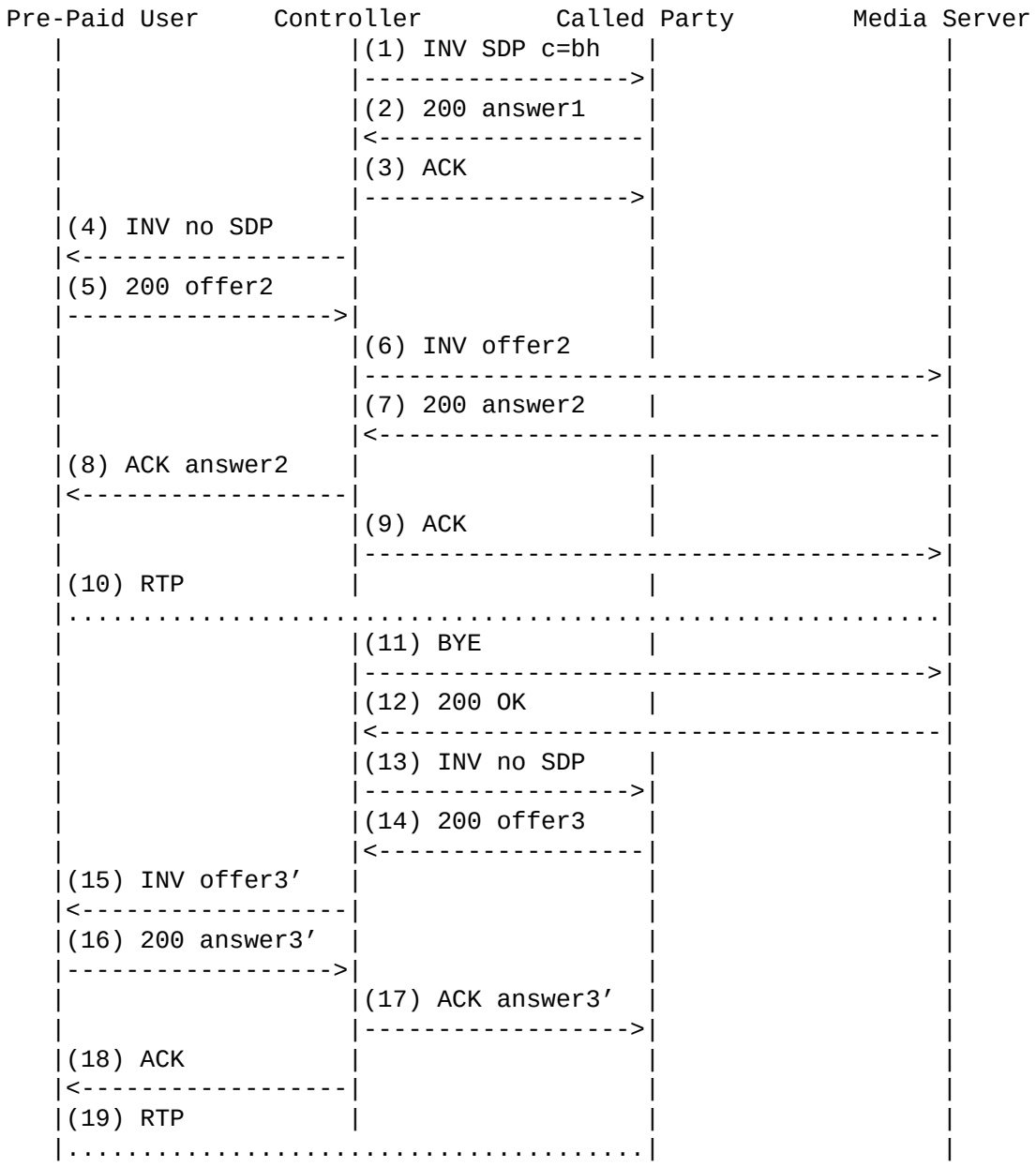


Figure 13

We assume the call is set up so that the controller is in the call as a B2BUA. When the timer fires, we wish to connect the caller to a media server. The flow for this is shown in Figure 13. When the timer expires, the controller places the called party with a connection address of 0.0.0.0 (1). This effectively "disconnects" the called party. The controller then sends an INVITE without SDP to

the pre-paid caller (4). The offer returned from the caller (5) is used in an INVITE to the media server which will be collecting digits (6). This is an instantiation of Flow I. This flow can only be used here because the media server is an automata, and will answer the INVITE immediately. If the controller was connecting the pre-paid user with another end user, Flow III would need to be used. The media server returns an immediate 200 OK (7) with an answer, which is passed to the caller in an ACK (8). The result is that the media server and the pre-paid caller have their media streams connected.

The media server plays an announcement, and prompts the user to enter a credit card number. After collecting the number, the card number is validated. The media server then passes the card number to the controller (using some means outside the scope of this specification), and then hangs up the call (11).

After hanging up with the media server, the controller reconnects the user to the original called party. To do this, the controller sends an INVITE without SDP to the called party (13). The 200 OK (14) contains an offer, offer3. The controller modifies the SDP (as is done in Flow III), and passes the offer in an INVITE to the pre-paid user (15). The pre-paid user generates an answer in a 200 OK (16) which the controller passes to user B in the ACK (17). At this point, the caller and called party are reconnected.

11. Implementation Recommendations

Most of the work involved in supporting third party call control is within the controller. A standard SIP UA should be controllable using the mechanisms described here. However, third party call control relies on a few features that might not be implemented. As such, we RECOMMEND that implementors of user agent servers support the following:

- Offers and answers that contain a connection line with an address of 0.0.0.0.
- Re-INVITE requests that change the port to which media should be sent
- Re-INVITEs that change the connection address
- Re-INVITEs that add a media stream
- Re-INVITEs that remove a media stream (setting its port to zero)
- Re-INVITEs that add a codec amongst the set in a media stream

- SDP Connection address of zero
- Initial INVITE requests with a connection address of zero
- Initial INVITE requests with no SDP
- Initial INVITE requests with SDP but no media lines
- Re-INVITEs with no SDP
- The UPDATE method [7]
- Reliability of provisional responses [6]
- Integration of resource management and SIP [2].

12. Security Considerations

12.1. Authorization and Authentication

In most uses of SIP INVITE, whether or not a call is accepted is based on a decision made by a human when presented information about the call, such as the identity of the caller. In other cases, automata answer the calls, and whether or not they do so may depend on the particular application to which SIP is applied. For example, if a caller makes a SIP call to a voice portal service, the call may be rejected unless the caller has previously signed up (perhaps via a web site). In other cases, call handling policies are made based on automated scripts, such as those described by the Call Processing Language [11]. Frequently, those decisions are also made based on the identity of the caller.

These authorization mechanisms would be applied to normal first party calls and third party calls, as these two are indistinguishable. As a result, it is important for these authorization policies to continue to operate correctly for third party calls. Of course, third party calls introduce a new party - the one initiating the third party call. Do the authorization policies apply based on the identity of that third party, or do they apply based on the participants in the call? Ideally, the participants would be able to know the identities of both other parties, and have authorization policies be based on those, as appropriate. However, this is not possible using existing mechanisms. As a result, the next best thing is for the INVITE requests to contain the identity of the third party. Ultimately, this is the user who is requesting communication, and it makes sense for call authorization policies to be based on that identity.

This requires, in turn, that the controller authenticate itself as that third party. This can be challenging, and the appropriate mechanism depends on the specific application scenario.

In one common scenario, the controller is acting on behalf of one of the participants in the call. A typical example is click-to-dial, where the controller and the customer service representative are run by the same administrative domain. Indeed, for the purposes of identification, the controller can legitimately claim to be the customer service representative. In this scenario, it would be appropriate for the INVITE to the end user to contain a From field identifying the customer service rep, and authenticate the request using S/MIME (see RFC 3261 [1], Section 23) signed by the key of the customer service rep (which is held by the controller).

This requires the controller to actually have credentials with which it can authenticate itself as the customer support representative. In many other cases, the controller is representing one of the participants, but does not possess their credentials. Unfortunately, there are currently no standardized mechanisms that allow a user to delegate credentials to the controller in a way that limits their usage to specific third party call control operations. In the absence of such a mechanisms, the best that can be done is to use the display name in the From field to indicate the identity of the user on whose behalf the call is being made. It is RECOMMENDED that the display name be set to "[controller] on behalf of [user]", where user and controller are textual identities of the user and controller, respectively. In this case, the URI in the From field would identify the controller.

In other situations, there is no real relationship between the controller and the participants in the call. In these situations, ideally the controller would have a means to assert that the call is from a particular identity (which could be one of the participants, or even a third party, depending on the application), and to validate that assertion with a signature using the key of the controller.

12.2. End-to-End Encryption and Integrity

With third party call control, the controller is actually one of the participants as far as the SIP dialog is concerned. Therefore, encryption and integrity of the SIP messages, as provided by S/MIME, will occur between participants and the controller, rather than directly between participants.

However, integrity, authenticity and confidentiality of the media sessions can be provided through a controller. End-to-end media security is based on the exchange of keying material within SDP [10].

The proper operation of these mechanisms with third party call control depends on the controller behaving properly. So long as it is not attempting to explicitly disable these mechanisms, the protocols will properly operate between the participants, resulting in a secure media session that even the controller cannot eavesdrop or modify. Since third party call control is based on a model of trust between the users and the controller, it is reasonable to assume it is operating in a well-behaved manner. However, there is no cryptographic means that can prevent the controller from interfering with the initial exchanges of keying materials. As a result, it is trivially possibly for the controller to insert itself as an intermediary on the media exchange, if it should so desire.

13. Acknowledgements

The authors would like to thank Paul Kyzivat, Rohan Mahy, Eric Rescorla, Allison Mankin and Sriram Parameswar for their comments.

14. References

14.1. 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] Camarillo, G., Ed., Marshall, W., Ed. and J. Rosenberg, "Integration of Resource Management and Session Initiation Protocol (SIP)", RFC 3312, October 2002.

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

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

[5] Schulzrinne, H., Oran, D. and G. Camarillo, "The Reason Header Field for the Session Initiation Protocol (SIP)", RFC 3326, December 2002.

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

[7] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

14.2. Informative References

[8] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[9] Petrack, S. and L. Conroy, "The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services", RFC 2848, June 2000.

[10] Andreasen, F., Baugher, M. and D. Wing, "SDP Security Descriptions for Media Streams", Work in Progress, October 2003.

[11] Lennox, J., Wu, X. and H. Schulzrinne, "CPL: A Language for User Control of Internet Telephony Services", Work in Progress, August 2003.

15. Authors' Addresses

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Jon Peterson
Neustar
1800 Sutter Street
Suite 570
Concord, CA 94520
US

Phone: +1 925 363-8720
EMail: jon.peterson@neustar.biz
URI: http://www.neustar.biz

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027
US

EMail: schulzrinne@cs.columbia.edu
URI: http://www.cs.columbia.edu/~hgs

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

16. Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.