Internet Engineering Task Force   MMUSIC WG
Internet Draft   Schulzrinne/Rosenberg
draft-ietf-mmusic-sip-cc-01.txt   Columbia U./Bell Laboratories
June 17, 1999
Expires: December, 1999

SIP Call Control Services

STATUS OF THIS MEMO

Abstract

This document describes a set of extensions to SIP which allow for various call control services. Example services include blind transfer, transfer with consultation, multi-party calls, bridged conferences, and ad-hoc conferencing. The services are supported in a fully distributed manner, so that they can be provided without a central conference server. However, a SIP proxy can act as a conference server to provide these services. For the various services described here, we overview the requirements for the service, and specify the protocol functions needed to support it. We then define a basic set of SIP primitives which can be used to construct these services, and others.

1 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 [1] and indicate requirement levels for compliant SIP implementations.

2 Introduction

The Session Initiation Protocol (SIP) [2] is a signaling protocol used for the initiation of multimedia sessions. SIP also defines mechanisms for termination of sessions using the BYE method. However, SIP has less support for signaling of services that take place during the call itself. These kinds of services can be broken into several classes:

Session Control Services These services relate to the media session. Examples include floor and chair control (which controls who can send/receive data in the session), hold, and mute.

Call Control Services These services relate to participant management. These services are all built on the basic blocks of adding and removing users from a call. Examples include transfer (the simultaneous removal and addition of a member from a call), multi-party calling, call bridging, and ad-hoc bridged conferences.

This document describes extensions to SIP for providing call control services. The services are supported in a fully distributed manner, so that they can be provided without a central conference server. However, a SIP proxy can act as a conference server to provide these services. Our aim is to provide a general set of tools which can be used to construct, at a minimum, a core set of services, but be potentially useful as building blocks for future services. To accomplish this goal, we begin by overviewing the requirements for each of the core services. This includes its basic functional requirements and its security requirements. Then, we overview the technical issues in providing these services, and outline the basic primitives that we have concluded are needed. The next section formally defines these primitives through new headers and UA behavior.

3 Services

We overview the services which we desire to support at a minimum. For each, we define the requirements for the service, with a particular focus on security. Security is a primary concern for many of these services. As such, the following general principles apply:

- o Parties involved in some service should be able to cyryptographically verify the identity of the other parties in

the service

- o Parties involved in some service should have a choice about their participation in the service

- o Parties involved in some service should know what the service being invoked is

These three basic requirements are a natural consequence of an architecture where endpoints are assumed to be intelligent. Note, however, that just because the protocol provides information and gives choices, does not mean all implementations need to take advantage of this. Thin and dumb endpoints can choose not to provide information to the end users, and can choose not to provide choices to them. This has the advantage of enabling one common protocol for smart and dumb endpoints alike.

3.1 Blind Transfer

In the blind transfer service, two parties are in an existing call. One party, the transferring party , wishes to terminate the call with the other party the transferred pary , and at the same time, transfer them to another party, the transferred-to party

```
                    ----------------
                 -> | Transferred-to |
                /   |     party      |
               /    ----------------
              /
             /
  --------------                          ----------------
 | Transferred  |<------------------------| Transferring   |
 |    party     |                         |    party       |
  --------------                          ----------------
```

Figure 1: Transfer Service

Some of the requirements for this service include:

- o The original call terminates regardless of whether the

transfer succeeds or not.

- o The transferring party does not know whether the transfer succeeds or not.
- o The transferred party should be able to know that they are being transferred
- o The transferred party should be able to know to whom they are being transferred
- o The transferred party should be able to decide whether to accept or reject the transfer
- o If the transferred party rejects the transfer, the call with the transferring party still terminates
- o The transferred party should be able to verify that they are being transferred by the transferring party
- o The transferred-to party should know that they are being transferred-to
- o The transferred-to party should be able to know the identity of the transferring party
- o The transferred-to party should be able to accept or reject the transferred call just like any other call

3.2 Transfer and Hold

This service is a variation on blind transfer. The difference is that the transferring party does not leave the call with the transferred party. If the service is successful, the transferred party is involved in two calls - one with the transferring party, and one with the transferred to party. Many of the requirements are similar. The requirements for the service are:

- o The call between the transferring and transferred parties remains active, regardless of the status of the new call between the transferred and transferred-to parties.
- o The transferring party does not know whether the transfer succeeds or not.
- o The transferred party should be able to know that they are being transferred

- o The transferred party should be able to know to whom they are being transferred

- o The transferred party should be able to decide whether to accept or reject the transfer

- o If the transferred party rejects the transfer, the call with the transferring party remains unchanged

- o The transferred party should be able to verify that they are being transferred by the transferring party

- o The transferred-to party should know that they are being transferred-to

- o The transferred-to party should be able to know the identity of the transferring party

- o The transferred-to party should be able to accept or reject the transferred call just like any other call

- o The transferred-to party should not be able to ascertain, through signaling messages, that the transferring party is still communicating with the transferred party. In other words, blind transfer and transfer and hold appear identical to the transferred-to party.

3.3 Transfer with Consultation

This service is similar to blind transfer. However, the transferring party first contacts the transferred-to party to approve the transfer through multimedia communication. Pending approval, the transferring party then simultaneosly disconnects from the transferred-to and transferred parties, and connects the transferred and transferred-to parties. The transferring and transferred parties stay connected if, for some reason, the transfer fails.

The requirements for this service are more complex. They include:

- o The transferring party should not need to know ahead of time that they will transfer the call to the transferred-to party. In other words, it should not be neccesary to know ahead of time that the consultation call (between the transferring and transferred-to parties) is for the purposes of a transfer.

- o The transferred party should be able to know that they are being transferred

- The transferred party should be able to know to whom they are being transferred

- The transferred party should be able to decide whether to accept or reject the transfer

- The transferred party should be able to verify that they are being transferred by the transferring party

- The transferred-to party should know that they are being transferred-to

- The transferred-to party should be able to know the identity of the transferring party

- The transferred-to party should be able to know that the transferred party is being transferred as a result of the consultation call in progress with the transferring party.

- The transferred-to party should be able to accept or reject the transferred call just like any other call

- If the transferred-to party accepts the transfer, the transferring party should be able to know this

- If the transferred-to party rejects the transfer, the transferring party should be able to know this

- The call between the transferring and transferred-to party terminates at the same time as the call between the transferring and transferred party, should the transfer be successful

This service is harder to implement. To be done in a distributed manner requires that information on the success of the call between transferred and transferred-to parties is communicated back to the transferring party.

3.4 Multi-party Conferencing

Multiparty conferencing allows multiple participants to simultaneously exchange media so that each party hears media from every other one. There are many flavors of this service.

3.4.1 Loosely Coupled Multicast Conference

In this flavor, there is a very large conference, for which multicast is being used to distribute the media. The conference is large enough

so that there is not a direct signaling relationship between all parties. Session participants simply join the multicast group, and learn about each other through RTCP [3]. This kind of conference model is often referred to as a loosely coupled conference

The main requirement is to be able to invite another participant to join in this conference. In fact, this kind of conference is readily supported by baseline SIP, as it was the initial application for it. The only new requirement is that the called party needs some way to know that there will not be an actual SIP session - no BYE will ever arrive (nor should one be sent). The INVITE delivers the session invitation, and thats it. Relying on session parameters for this is undesirable, since it leads to a dependency between SIP behavior and the specific session type. Furthermore, it may not be possible to ascertain from the media session whether an actual SIP session is needed.

3.4.2 Distributed Full Mesh

In this conference model, each participant has a SIP signaling session open with each other participant. The media session may be multi-unicast or multicast. To support these conferences, the signaling must provide support for:

- o Transitioning gracefully from a normal two-party call to a conference without knowing apriori this will happen

- o Adding parties to the conference

- o Leaving the conference

The requirements for the service are:

- o Any member of an existing conference can add another party to the conference.

- o The new party should know they are being asked to join an existing conference.

- o The new party should be able to accept or reject the invitation to join the existing call.

- o If the new party rejects the invitation to the conference, no other participant should have received any messages which indicates they were ever asked to join the conference

- o The new party should be able to know, within the limits of synchronization of state across participants, the current set

of participants in the call before they decide whether to reject or accept the invitation.

- o Each participant in the call should learn that a new party is being added.
- o Each participant in the call should be able to cryptographically verify that the new party has been invited by a specific participant.
- o Each participant in the call should be able to decide whether to accept or reject the new participant.
- o If any existing participant in the call rejects the new participant, the new participant is not added to the call at all.
- o The inviting party can learn the success or failure of the addition of the new party.
- o Each participant should be able to know whether the new party was successfully added or not.
- o Any participant should be able to leave the conference at any time.
- o Each participant should know within a short period of time when some other participant has left
- o A participant who leaves the conference should have its SIP signaling relationship terminated with all other participants.
- o It must be possible for two participants to simultaneously add a new party to the conference.
- o It must be possible for a participant to add another to the conference while some other participant leaves the conference.
- o The existence of the conference does not depend on the presence of any single user in the conference.
- o The conference terminates when the last two parties terminate their signaling relationships.

It is important to note that this kind of conference does not require the use of a centralized conference controller.

3.4.3 Dial-in Bridge

Another conferencing application is the "dial-up bridge". In this scenario, a media bridge is used, and this device also acts as a centralized signaling server. Users join the conference by "dialing-in", which means they try and initiate a SIP session with the conference bridge directly. Participants do not maintain a signaling session with each other. Rather, each participant maintains a single SIP session with the conference bridge.

The requirements for this kind of conference are:

- o It should not be neccesary for a participant to know apriori that they are contacting a dial-up bridge - it should take place as a regular SIP call.
- o Participants should be able to join the conference at any time by dialing in.
- o Participants should be able to invite another participant to join the conference call.
- o Participants should still be able to learn, through some means, the identity of the other participants in the call.
- o Participants should be able to leave the conference at any time.
- o When a participant leaves or joins, this information should be propagated to all other conference participants through some means besides tones or announcements in the media stream.
- o It must be possible for the conference bridge to authenticate the identity of participants.

3.4.4 Ad-hoc Bridge

This service is not so much another conferencing model, as a transition mechanism between conferencing models. A conference starts out as a fully distributed mesh. These conferences become unwieldy as this number of participants approaches tens to hundreds. Someone in the conference then decides to transition the call to a conference bridge. The bring a conference bridge into the call, and then instruct each participant to drop their signaling relationships with the other participants in favor of a single signaling relationship with the bridge. After the transition is complete, the conference runs similar to the dial-in bridge case. However, there are some distinctions. In the dialup conference, any participant can join in without being invited if they know a conference code of some sort. In the ad-hoc bridge case, participants must still be actively invited.

The requirements for this service are:

- o The transition must be at the behest of one of the participants.
- o Any participant can cause the transition to take place.
- o It is not necessary for the protocol to detect and resolve simultaneous transitions. It is assumed that the persons in the conference would coordinate this themselves.
- o The conference should continue to be operable during the transition
- o Participants should be informed of the transition, but it must be possible for the perception to be that there has been no change.
- o It should be possible for some participants to accept the transition, and appear through the bridge, and for others to remain in full mesh.
- o Participants should be able to leave the conference at any time, including the transition period.
- o Participants should be able to invite others to the conference, even during the transition period. The mechanism for inviting them should not depend on the fact that a transition is taking place.

3.4.5 Conference out of Consultation

In this service, a user A has a call in progress with B, and a separate call in progress with C. These calls are unrelated, with different Call-ID's. From this double call scenario, the conference out of consultation service allows the calls to be merged, resulting in a single, full-mesh conference, as described above.

The requirements for this service are:

- o Only participant A can invoke the service
- o It must not be neccesary for A to know that he will merge the two calls before any or either of them is made
- o It must not be neccesary for A to have been the initiator of the calls that are being merged

- It must be possible to merge an arbitrary number of calls

- The participants being merged must be informed that the merging is taking place

- A participant must be able to reject the merge, in which case they are disconnected with all parties

- A participant must be able to verify that A was the party that initiated the merge.

4 Discussion of Implementation Options

This section discusses some of the technical issues in designing a protocol mechanism to support the above requirements.

4.1 Transfer

For the discussion which follows, we assume the transferring party is A, the transferred party is B, and the transferred-to party is C.

The nature of the transfer service is that the transferred party (B) must be informed about the transfer and accept it before C (the transferred-to party) is contacted. This implies that the messaging flow for the service must consist of a message from A to B, and then B to C.

The message from A to B must simultaneously disconnect A and B, and alert B about the transfer. This is most readily accomplished by including some kind of header in the BYE message which indicates that B should initiate a call to C. This header is the Also header, which is described in greater detail in section 5.1. It contains the address of a participant, along with a signed token. This token is the signature over the sender of the message (the From field), the address in the Also header, and the Call-ID. Since C needs to know that he is being contacted as a result of a transfer, the INVITE from B to C must contain some kind of header indicating that it was A who asked for the transfer. This header needs to contain A's name along with the authorization token from the Also header. This token allows C to verify that A requested the transfer to C for this particular call. This header is the Requested-By header, described in greater detail in section 5.3.

Therefore, the basic transfer messaging flow is simple. A sends a BYE to B, containing an Also header listing C. The BYE causes A and B to be disconnected. User B is alerted about the transfer. If accepted, B sends an INVITE to C, including a Requested-By header in the INVITE.

4.2 Full mesh conferences

We assume the conference starts as a standard two party call in SIP. One of the parties wishes to add a third to the conference. Based on the requirements, the new party needs to first be asked if they wish to join the conference. This implies that messaging begins with the inviting party (party A) sending a message to the new participant (party B). This message must contain a list of the other participants. If the invitation is acceptable to B, B can begin to join the conference. To join the conference, a signaling relationship must be established between B and all other participants. This can be done by having existing participants contact B, or B contacting existing participants. Since B has the list of participants in the initial INVITE from A, the most efficient approach is to have B contact each participant directly.

Thus, in the simplest scenario, A (who is in a call with C), sends an INVITE to B. This INVITE contains an Also header, indicating C. B sends an INVITE to C, containing a Requested-By header naming A. C accepts, and then B sends a 200 OK to A. Now, there is a signaling relationship between all parties. Adding additional parties is done in a similar fashion.

On the surface, this simple mechanism appears sufficient. However, it is not. Consider the following problematic cases (assume A,B, and C are already in a conference):

- o While A is adding D, B adds E. Since A did not tell D about E (as it didn't know about E), D may not know of E's existence. This results in a partially connected conference.

- o While A is adding D, B sends a BYE to the group. If this BYE is sent by B before the INVITE from D arrives at B, B should respond to the INVITE with an error. As far as B is concerned, the INVITE has failed, and it responds with an error to A. What should A do now? It cannot tell whether the add party failed because someone left the group, or because someone refused to add that party. In one case, the add should be tried again, and in the other, it should not. Even worse, should B accept the call from D, a partially disconnected conference will occur.

- o What happens if a transfer takes place at the same time as an add party?

- o A participant leaves the conference, but fails to send a BYE to all the other participants (either on purpose or by accident). The result is a partially disconnected conference.

The problems can all be categorized as difficulties in synchronizing a distributed database. The database, in this case, is the set of participants. This database is replicated at each participant. The database is dynamic, with each participant owning the entry in the database corresponding to itself. As changes occur, everyone must be quickly synchronized to achieve a consistent view of the conference participants.

4.2.1 Approach I: Caretaker

In this approach, the party (A) that invites another (D) to the conference is its caretaker. When A adds D, it informs D of the other participants it knows about. D then sends an INVITE to each of these in turn, establishing a signaling relationship. Should the participant list (at A) change during the time D is being addded (until a 200 OK arrives from D), A makes note of these changes, and then propagates them to D.

The difficulty with this approach is there is no easy way for A to know when it can cease being caretaker for D. Lets say A invited D, and told it to contact B and C, which it did. After receiving the 200 OK from D, A receives an INVITE from E, a new party added by B. Now, does A need to inform D about E? If B had invited E after knowing about D, A does not have to inform E, but if B invited E before knowing about D, A does have to inform D.

Furthermore, should the caretaker itself leave the conference, the mechanism ceases to work. As a result, we don not believe this approach is viable.

4.2.2 Approach II: Flooding

We make the following important observation:

> synchronization of the set of participants in a fully meshed multiparty conference is similar to the problem of database synchronization in link state routing protocols, like OSPF.

Based on this, we can develop mechanisms for SIP based on the same synchronization, flooding, and adjacency notions in OSPF. We further observe that this approach has already been used as the basis for existing conferencing mechanisms [4].

To solve the first problem above, we introduce additional semantics and behavior into the Also header. When A invites D into the conference, the INVITE includes an Also header listing B and C. This

prompts D to send an INVITE to both B and C. In OSPF terminology, this effectively establishes an adjacency between D and B, and D and C. These INVITEs contain Also headers as well, listing the set of participants the D believes is in the call.

When B and C receive this INVITE, they compare the set of participants in the Also header with the set of participants they believe are in the call. Note that this is effectively the same operation as database synchronization in OSPF. The result is three sets for each pair (assume B below):

S1: S1 is BD - the intersection of the set of participants B and D both believe to be in the conference.

S2: S2 is B - BD - the set of participants B believes to be in the conference, but D is not aware of

S3: S3 is D - BD - the set of participants D has been asked to contact, which are not known to B

First consider S2. There are only two ways this inconsistency can happen. The first way is that B has learned of a new participant before A issued the add party to D. The second is that A has learned the party has left the call before the INVITE from D arrives at B. Unfortunately, the desired behavior is different in each case. If B is correct, and a new party has joined, B should return the address of the party in the 200 OK to the INVITE from D. This would prompt D, in turn, to add those parties. On the other hand, if B is wrong, and the party has left the conference, B should say nothing in the 200 OK about this participant.

To enable these differing cases, we can add two additional pieces of information to the addresses in the Also header. These are the participant state (either active or inactive), and the version number. When a participant receives a BYE from another, they mark that participant as inactive, and hold onto the state for a short duration (time TBD). This member is included in Also headers as other participants, but they are marked as inactive. Based on this, in the case above, B can ascertain the right behavior.

The version number satisfies a different need. What happens if the participant that left, comes back because they are re-INVITEd? In this case, some of the participants will think this participant is inactive, and others will consider them active. To determine which piece of state is correct, the version number increments each time the state changes. The version with the highest value is always the most recent. (TBD: who sets this? Can't always be the originator). This is identical to the use of sequence numbers in LSA's in OSPF.

Consider now the set S3. When B receives the INVITE, this represents the set of users D claims is in the conference, but B does not know about. Since B keeps a cache of users who have left the conference, B can be sure these are new participants that it has not learned of yet. B should then send an INVITE to these users to establish signaling relationships with them. As with other INVITEs' the Also field contains B's perspective on the set of conference participants. This is effectively the same process as flooding of new LSA's in OSPF.

TBD: How is requested-by handled in these various cases?

We believe the flooding approach to be robust and well-proven from many other protocols.

4.3 Dial-up Bridges

Dial up bridges are easily supported. We model them as virtual users. When a user wants to join a dial-up conference, they send an INVITE to the conference bridge. The bridge answers the call, and establishes a point to point signaling relationship with the new participant. The bridge performs the mixing locally, and sends the mixed stream to each participant separately. As far as each participant is concerned, they have a single signaling relationship with one other entity - the conference server.

Fortunately, this does not prohibit each party from learning the identity of the others in the call. The bridge is effectively an RTP mixer. As such, it can use contributing sources (CSRC) in the RTP and RTCP packets to identify the other participants in the call.

A user leaves the conference by hanging up with the bridge, as they would hang up with any other user in a normal two party call.

An important issue is how conferences are identified. In the telephone network, there is usual a dial-in number and a passcode that the participant must know. In SIP, there are many more possibilities:

o The conference is identified by a single URL - sip:conference332@conferences.com, for example. A user sends an INVITE to this address. The bridge identifies the conference by looking at the URI in the Request-URI.

o There is a single URI for each bridge - sip:bridge3@conferences.com. The specific conference is identified by a passcode sent as the password in the URI: sip:bridge3:9987097@conferences.com.

- o The conference is identified by a single URL, as in the first case. However, participants must also have a passcode. When the server receives an INVITE for this URI, it responds with a 401 demanding digest authorization. The shared secret used for authenticating the caller is the passcode.

- o The conference is identified by a single URL, as in the first case. The server is programmed with the public keys of those participants allowed to join. When a participant tries to join the conference by sending an INVITE to its address, the server uses PGP authentication to verify the user is one of those permitted. This allows for tight, per user controls on conference participation.

Some have suggested identifying the conference by Call-ID. We do not believe this is the right approach. The Call-ID represents a SIP signaling relationship shared among two or more users. Since, in the conference bridge case, each user has a separate signaling relationship with the bridge, using a common Call-ID is not appropriate.

Note that, based on this description, dial-in conferences are readily supported in baseline SIP without any extensions. However, the situation is more complex when a participant wishes to add another to the conference.

We believe it is essential that the act of adding a party to a bridged conference is no different than the act of adding a party to a fully meshed one. Consider a bridged conference with participants A, B, and C. Each has a signaling relationship with the bridge, X. A wishes to bring D into the conference. Using the same mechanisms as for fully meshed conferences, A sends an INVITE to D, with the Also header indicating X. D then sends an INVITE to X, which accepts. The result is that D has a signaling relationship with the bridge, but is still maintaining its signaling relationship with A.

To resolve this, the bridge needs to step up and instruct D to effectively abandon its signaling relationship with A (and vice a versa). This does not mean the bridge wants A to send an BYE to D. Rather, the bridge wants another one call leg to subsume another. For D, this means that the D-X call leg should subsume the D-A call leg. To accomplish this, the bridge sends an INVITE to D with a header called Replaces. Replaces indicates that the call leg the INVITE arrived on is subsuming the one identified in the header. The Replaces contains the address of A. The request must also be authenticated, since the Replaces header presents a powerful DOS attack. Users should accept an INVITE with a Replaces header only after either requesting confirmation from the user, or if the request

is signed by an authorized bridging service.

4.4 Conference out of Consultation

In this service, A is in a call with B, and separately, A is in a call with C. These are two separate calls, and thus have identical Call-IDs. Transitioning to a full mesh multiparty conference is relatively straightforward. A can simply send an INVITE to B, with an Also listing C. As far as B is concerned, the process is a normal add party.

The only difference is that the Call-ID is different in both calls. Thus, the INVITE to C from B would not appear to be for the same call. To resolve this, A must effectively change the Call-ID with B, and then perform an add party. The change in Call-ID is accomplished by having A send an INVITE to B (using the Call-ID from the A-C call), with a Replaces header containing the A-B Call-ID and A's address. The Replaces header has the same semantic here as in the bridged conference case above. The call leg identified in the Replaces header is subsumed by the call-leg of the INVITE.

Once this transition has taken place, A can send an INVITE to B, containing Also:C, as discussed above.

If the calls being connected are multi-party calls, the situation is more complex. (TBD: does this mechanism work for bridging two full mesh calls?)

4.5 Ad-hoc conference bridging

To support an ad-hoc conference bridge, the following operations must take place:

- o One of the parties in the call must contact a bridge, informing it of the set of participants

- o The bridge must contact those participants, and cause them to replace their signaling relationship with the other parties with the relationship with the bridge

To support the first, the initiator sends a message to the bridge, containing the list of participants. We use an INVITE method for this, and the participants are listed in the Also headers. It is not clear if this is the right approach. The semantics of INVITE with Also are not the same here. The bridge is not being asked to join the call, rather, its being asked to take over the the signaling and media connectivity for the call. For this reason, it might be appropriate to define a new method to indicate this, or perhaps a new

header or parameter to Also.

Once the bridge has been contacted with the list of participants, it must send an INVITE to each (using the same Call-ID as the current call) to establish a relationship with them. This call leg must eventually replace the call legs the user has with all the other users. However, the user should not subsume a call leg with some other user until the bridge has succesfully contacted that other user.

For this to work, the initial INVITE with each user is treated as a normal add-party. The Also list contains those users the bridge knows about (initially, those the initiator told the bridge about). As far as the contacted user is concerned, a normal add party is taking place. The response is (under normal cases) a 200 OK containing those additional parties the contacted user knows about. This way, if a user was in the process of an add party while someone else transitioned to a bridge, the bridge can learn about the new party. Should the user add parties after being contacted by the bridge, the user will tell the new party about the bridge. This allows the bridge to learn about all users that come (and go) during the transition period.

Once the bridge has completed contacting all participants in the party, it attempts to subsume the various call legs into its own call leg. To do this, it sends another INVITE to each participant, listing those call legs which must be subsumed. In the case where a participant has added another user after the response to the bridges initial INVITE was sent, but before the the "subsuming INVITE" arrives, things still work. The new party will be informed about the bridge, contact the bridge, and the bridge accepts. The bridge can then send another INVITE to each user subsuming this particular new call leg.

4.6 Transfers to Multiparty Conferences

This situation is more complex than normal transfers. We first consider the case of a full mesh signaling relatioship. Assume A, B, and C are already in a call. A wishes to transfer both B and C to Z.

Extending the mechanism for a single party call is the ideal choice. In this case, A would ask B to contact Z, and A would ask C to contact Z. Everything works fine so long as (1) both B and C perform the transfer (i.e., both contact Z), and (2) Z accepts both B and C's invitations. However, if these assumptions fail to hold, the resulting transfer will only partially complete. For example, it is possible that only A gets transferred to Z.

Whether this behavior is acceptable or not is a good question. We believe that since the blind transfer mechanism has no guarantees on success (the transferring party disconnects in either case), this behavior is acceptable.

Another issue that arises for multiparty conference transfers is a flooding effect at the transferred-to party. If a large number of participants where transferred, Z would receive, in rapid succession, an INVITE from each. To facilitate a usable application, Z should not really prompt the user about accepting each of these parties. Rather, it should accept them all if it accepts the first. So, we therefore have the rule: if a user accepts a transfer, it must accept all other parties which have been transferred.

The specific mechanism is the same for multiparty conferences. A sends a BYE to B and C containing an Also header listing Z. B and C send a 200 OK to the BYE, and then send an INVITE to Z. This INVITE contains a Requested-By header listing A. When Z gets the first of these, it alerts the user and accepts the call. (TBD: should these triggered INVITEs contain Also's? Probably. But, in this case, Z is going to get the first INVITE with lots of Also's. Many of these (but perhaps not all) will eventually contact Z directly. So, should Z send an INVITE to those in the Also headers it doesn't know about already? Perhaps it should wait a while to see who contacts it first. As an alternative, the BYE from A can contain Z's address, PLUS those it send the BYE to. As a result, the INVITE from B or C to Z would only contain those users in the Also which Z did not list in the BYE. What is the right approach here?)

5 Header Syntax

This section defines the syntax for the three new headers defined here - Also, Replaces, and Requested-By.

5.1 Also

The Also header is used to list other participants in a call. It is a request and response header, and contains a list of SIP URI's, along with some special parameters.

```
Also           = ``Also'' ``:'' 1#Also-Values
Also-Values    = name-addr [parameters]
parameters     = 1*parameter
parameter      = ``;'' (status-param | version-param | crypto-param)
status-param   = ``status'' ``='' (``active'' | ``inactive'')
version-param  = ``version'' ``='' 1*3digit
```

```
crypto-param   = ``token'' ``='' token
```

The crypto-param is a token which is copied into the Requested-By header for requests that are "triggered" as a result of an Also header. The token is a signature over the URI of the entity generating the Also header, the address in the Also header itself, and the Call-ID. See section 6.1 for details on its computation.

5.2 Replaces

The Replaces header is used to indicate that the call leg identified in the header is to be subsumed by the one initiated by this INVITE. It is a request header only, valid only in INVITE messages. The syntax is:

```
Replaces       = ``Replaces'' ``:'' 1#Replaces-Values
Replaces-Values=  SIP-URI [call-id-param]
call-id-param  =  ``;'' ``call-id'' ``='' token
```

When the call-id parameter is not present, it is presumed to be the same as the Call-ID of the INVITE itself.

5.3 Requested-By

The Requested-By header is a request header only. It identifies the participant who asked the UAC to send the request. The syntax is:

```
Requested-By   = ``Requested-By'' ``:'' name-addr [req-params]
req-params     = ``;'' token-param
token-param    = ``token'' ``='' token
```

6 Also and Requested-By Header Semantics

This section overviews the detailed semantics associated with the Also and Requested-By headers.

6.1 Sending an Untriggered INVITE

When a UAC sends an INVITE containing an Also header, without having been asked by some other UAC to do so, it is called an untriggered

INVITE. Untriggered INVITEs are sent when a user wishes to add another user to a call, or to perform a transfer and hold service. Other uses may exist.

An untriggered INVITE MUST NOT contain a Requested-By header. This header is used to determine whether an INVITE is triggered or not.

When a UAC sends an untriggered INVITE containing an Also header, it implies that the UAC wishes the recipient to send an INVITE to those parties listed in the Also headers. If sent to a party not already in the call, the INVITE effects an add party operation. If sent to a party already in a call, it affects a transfer and hold operation. To ensure fully connected conferences, it is RECOMMENDED that a UAC include a URI for each participant it is aware of.

Each element in the Also list should additionally contain a status and a version parameter. If the UAC believes the participant is no longer in the call, the status parameter is set to inactive, otherwise its active. The version parameter contains the version of the status for each participant that the UAC is currently aware of.

The Also header SHOULD contain a token parameter for each URI listed. This parameter is computed in the following fashion:

1. Initialize a string to the value of the Call-ID.

2. Append the URI from the Also, not including any displaynames, but otherwise including all URI parameters. Also append the Also parameters status and version.

3. Append the URI that will be included in the From field of the INVITE.

4. Append the URI that will be included in the To field of the INVITE.

5. Compute a signature over this field, using a XXX hash and encryption using XXX.

6. The signature is then base64 encoded. The result is the token.

The response to the INVITE is a non-200 value if the UAS failed to establish a call leg with all the participants listed in the Also fields, or if the UAS was unwilling or unable to execute the request.

A 200 OK response means that the UAS successfully established the call with those participants which have not already left the call. In

other words, if A sends an untriggered INVITE to B, containing C in the Also header, B will send an INVITE to C. If C has left the call (a fact which A did not know yet), C will respond with a specific error code indicating this. In this fashion, B will know that it may still respond with a 200 OK to A should all other call legs become established. Furthermore, if other participants have joined the call since A sent the INVITE to B, B may have established call legs to them as well. The triggered INVITE will fail if B fails to establish a call leg with those participants, even if they are not listed in the Also header.

Thus, a UAC SHOULD NOT treat a 200 OK to an untriggered INVITE as an indication that a call leg was established with all (and only) the participants listed in the Also header.

6.2 Receiving an Untriggered INVITE

A UAS can determine whether or not an INVITE was triggered or untriggered based on the presence of the Requested-By header. Presence of this header means that the INVITE was triggered, and its absence implies untriggered.

If the UAS receiving the INVITE is not currently in the call identified by the Call-ID, the UAS is being invited to join an existing call as a new member. The UAS SHOULD alert the user and ask for confirmation.

If the UAS receiving the INVITE is currently in a call identified by the Call-ID, the UAS is being transferred and held. The UAS SHOULD alert the user and ask for confirmation.

6.2.1 New Call

The UAS SHOULD send a 100 Trying response. If the transfer or add party request is not acceptable to the user, a 6XX response SHOULD be sent to the UAC. If the transfer/add-party is acceptable, the UAS MUST NOT respond definitively at this point.

Instead, the UA formulates an INVITE for each participant listed in the Also header. Each INVITE MUST also contain a Requested-By header. This header is formed by attaching the URI in the From field in the INVITE to the Requested-By header. The token from the element in the Also field is copied to the token parameter in the Requested-By header. The URI for the Also field is copied into the To field of the INVITE. The remaining fields are initialized as they would be for any other INVITE sent by this UA. The INVITE's generated by the UA are called triggered INVITEs.

The UA also formulates an internal participant list. This list contains a set of URIs for each user, and for each, a version and status parameter. This list is initialized to the set contained in the Also header in the INVITE. This list is also placed into the Also headers of each triggered INVITE. The token in the Also field is generated as described in section 6.1. Note that this is NOT the same token received for this Also element in the untriggered INVITE. It is regenerated with the UA as the originator.

Each triggered INVITE is then sent. The INVITEs MAY be sent in parallel, or MAY be sent sequentially, or MAY be sent in any groupings deemed appropriate. However, for sake of low latencies, sending the triggered INVITEs all at once is RECOMMENDED.

If the UA receives a response to any of these INVITEs that is not 200 or 6XX (Not in Call), the UA determines that it was not successfully added to the call. It MUST send a BYE to those participants which:

- o responded to a triggered INVITE with a 200
- o have not yet responded
- o sent it a triggered INVITE for the same call

The latter case occurs when another party in the call (who has received an INVITE from the UA) adds a new party as well. This new party is informed of the UA, and sends it a triggered INVITE.

The UA MUST then respond to the original untriggered INVITE with an error code (TBD: what code?).

If a response to a triggered INVITE is a 200, this response may contain additional Also headers. These headers contain additional participants that the recipient of the triggered INVITE knew about, but the UA did not. The 200 may also contain updated status on participants the UA knew about.

The UA uses this list to update its own list of participants. New users learned about from the 200 OK are added to the list. Users listed in the 200 OK, which are known to the UA, but whose version number in the 200 OK is higher, are updated.

If the resulting update generates new active members, the UA MUST generate additional triggered INVITEs for them. The generation of these triggered INVITEs is identical to the above process, with an important difference. The URI in the Requested-By field is copied from the To field in the 200 OK. 200 responses to these triggered INVITEs may cause further triggered invites.

If the resulting update causes members to move from active to inactive, the UA should not send them a triggered INVITE if it has not already done so.

If the response to a triggered INVITE is a 6xx (not in call), the UA changes the status of that member to inactive, and increments the version number (TBD: should this be an increment? Perhaps the 6xx should contain the new version number).

Once responses have been received to all triggered INVITEs, all of which were either 200 or 6xx, the UA responds to the original INVITE (TBD: should this contain an Also list?). The UA is now in the call.

6.2.2 Existing Call

When a UA receives an INVITE containing an Also field, but no Requested-By field, the INVITE is to transfer/hold the UA.

If the originator of the INVITE is not already in the call, the INVITE is ignored. A 200 OK response is sent, however. (Transfers can only take place from parties already in the call). Those users in the Also header, who are already in the call, are ignored. If there are no remaining users from the Also list, the INVITE is ignored.

The UA then generates triggered INVITEs to the remaining UA's in the Also list. The behavior from this point forward is identical to processing triggered INVITE responses in the previous section.

6.3 Receiving a Triggered INVITE

When a UA receives an INVITE containing a Requested-By header, it has received a triggered INVITE. If the INVITE is for a new call, the UA has just been transferred-to. If the INVITE is for an existing call, the UA is being informed of a new party in this call.

6.3.1 New Call

The UA has just been transferred-to. The Requested-By header contains the address of the transferring party. The UA SHOULD verify that the token in the Requested-By header is valid. This will verify that the transferring party is, in fact the one listed, and that this party did, in fact, transfer the user listed in the From field. If the token is not verified, the UAS SHOULD respond with a 4xx code, and SHOULD NOT alert the user.

If the token is verified, the UA SHOULD alert the user, and ask for confirmation. If the user rejects the transfer-to, the UAS SHOULD send a 6xx response.

In either case, the UA MUST remember that it rejected the transfer for this Call-ID. Subsequent triggered INVITEs for the same call MUST be responded to with the same error response code. The UA MUST cache its rejection of this transfer (identified by the Call-ID and URI of the transferring party) for at least XX minutes. (TBD - what happens if a very old INVITE arrives after the cache expires, and the user accepts this time - we get a partial disconnect). The UA SHOULD alert the user if it receives a triggered INVITE with a different user listed in the Requested-By header, and MAY respond differently to this transfer.

If the INVITE is acceptable, the UA sends a 200 OK. Processing of subsequent triggered INVITEs (one will likely come from each participant in the call) follows the rules below for an existing call.

6.3.2 Existing Call

When a UA receives a triggered INVITE for an existing call, the INVITE is an attempt to inform the participant of new members for that call.

The UA SHOULD first verify the token. It does so by computing the hash of the Call-ID, To address, Requested-By address, and From address. This is compared to the decrypted value of the token using the public key of the user listed in the Requested-By. If the two match, the token is verified.

If the token is not verified, the INVITE is rejected with a 4xx response. If the token is verified, the UA checks to see if the user listed in the Requested-By is an active call participant. If they are not, the INVITE is rejected with a 4xx response (TBD: is there a case where the UA might not know about this participant yet?). If the user is a participant, the INVITE is accepted. The user SHOULD NOT be alerted.

The list of users in the Also header is then examined. If this list contains users already known to the UA, the local list of participants is updated if the version number is higher. If the list contains users not known to the UA, they are added to the local list of participants.

The UA then computes a difference set between its updated list and the list in the Also header. This set includes any users in its local list and not in the Also list. The set also includes users in both lists, but whose version is higher in the local list. This set is included in the Also header in the 200 OK to the INVITE. The token in the 200 OK is generated as described in 6.1.

The UA then computes a second difference set between its updated list and the list in the Also header. This set includes any users in the Also list not in its local list. The set also includes users in both lists, but whose version is higher than in the local list. The active users from this set are then sent triggered INVITEs. The Requested-By and Also fields in these triggered INVITEs are computed as described above. The inactive users in this set are then sent triggered BYE's. The Requested-By and Also fields in the triggered BYEs are computed in the same fashion as triggered INVITEs, except a triggered BYE contains no Also fields.

6.4 Sending an untriggered BYE

A UA MAY send a BYE, containing Also headers, at any time. This BYE simulataneously terminates a call leg with the recipient, and causes the recipient to attempt to set up a call leg to the parties listed in the Also header. Unlike the INVITE, the BYE response is sent immediately, without first adding the various parties. Sending an untriggered BYE is equivalent to a blind transfer.

The Also headers in the untriggered BYE MUST contain tokens. These tokens are generated in the same way described in section 6.1.

6.5 Receiving an untriggered BYE

If the BYE corresponds to an existing call leg, the UA sends a 200 OK to the BYE. If it does not, it sends a 481.

The UA then generates triggered INVITEs to all participants listed in the Also field. Generation of the triggered INVITEs, and processing of their responses, is done in the same fashion as described in section 6.1. The difference is, of course, that no additional response is sent to the BYE.

6.6 Receiving a triggered BYE

If the BYE doesn't correspond to an existing call leg, the UA sends a 481. The UA then validates the token in the Requested-By header. If it is validated, a 200 OK is sent to the BYE, and the call-leg is torn down.

7 Replaces header semantics

The Replaces header is used to allow one call leg to subsume another. The new call leg is identified by the combination of To, From, and Call-ID in the INVITE carrying the Replaces header. Replaces is a request header only, and MUST appear only in INVITEs. A UAS receiving a Replaces header in a non-INVITE request MUST respond with a 4xx

status code.

The request containing a Replaces header SHOULD be authenticated.

The Replaces header contains a list of call-legs, identified by the URI of the remote party and a Call-ID. If any of these are not valid call-legs as known to the UAS, the INVITE MUST be responded to with a 4xx status code. Otherwise, the UAS "abandons" each call leg listed - acting as if it had never been established. No BYE is sent. A 200 OK is then sent to the client.

If a BYE additionally contains Also headers, the UAS MUST first generate the triggered INVITEs implied by the Also headers. Only if all triggered INVITEs succeed should the UAS act on the Replaces header.

8 Example Call Flows

This section illustrates some example call flows. Messages are of the form:

```
INV B Also:C,D
BYE A Also:Y
```

Where INV implies an INVITE request, and BYE a BYE request. The letter after the method is the Request URI. Also:C,D implies that URI's C and D were in the Also header.

8.1 Basic Transfer

Figure 2 exemplifies the basic transfer in a two party call. A first sets up a call to B, and then transfers B to C.

8.2 Basic Add Party

Figure 3 exemplifies the basic add party. A and B are already in a call. A adds C to the call.

8.3 Add Party during Add Party

In this example (Figure 4), A and B are in a call. A adds another party, C, while B adds a different party, D. In the example, B adds D before learning about C.

```
A                B                    C
   INV
----------------->

   200 OK
<----------------

   ACK
----------------->

   BYE B Also:C
------------------>

   200 OK
<------------------    INV C ReqBy:A
                  --------------------->

                       200 OK
                  <---------------------

                       ACK
                   --------------------->
```

Figure 2: Transfer Message Flow

In the example, C acts on the untriggered INVITE, and sends a triggered INVITE to B. B responds with a 200 OK, also alerting it to D's new presence in the call. D, acting on its untriggered INVITE, sends a triggered INVITE to A, and learns about C. Now, both C and D know about each other. In the example, C sends the INVITE to D first. It is possible in other cases for D to send the INVITE to C first, or for both INVITEs cross each other on the wire (in this case, both sides back off with a 500 and a Retry-After, so that eventually one invitation reaches the other side without an invite in transit in the other direction).

Having received an INVITE from C, D doesn't bother to INVITE C. Both D and C then OK their respective INVITEs.

8.4 Party leaves during add party

In this example (Figure 5), a three party call is in place between A,

```
A              B                  C

         INV C Also:B
---------------------------------->
                INV B Also:C ReqBy:A
               <-------------------
                     200 OK
               -------------------->
                      ACK
               <--------------------
          200 OK
<-----------------------------------
            ACK
----------------------------------->
```

Figure 3: Add Party Message Flow

B and C. A adds another user, D, and shortly thereafer, C leaves the call.

Since D learns from B that C has left the call, D does not bother to contact C, and responds right away to the add party. The result is now a three party call with A,B, and D.

9 A note on multicast media

Another useful service, which we have not discussed so far, is to transition a conference from distributing media through multi-unicast to distribution through multicast. In fact, this is not a SIP issue at all. However, we discuss it here for completeness.

Assume a call between A, B, and C. Media is being distributed through multi-unicast. At some point, A decides its appropriate to transition to multicast. It should send a re-INVITE to B and C, containing an updated SDP with a multicast group (allocated by A by some means, perhaps MADCAP [5]. If the transition to multicast is acceptable, both B and C respond with a 200 OK. No SDP is needed in the response, as per [2].

If B and C decide to switch to multicast, it is in their interest (but not required) to send a re-INVITE to the other participants they

```
A                 B               C                D

           INV C Also:B
----------------------------------->
                        INV D Also:A
                  ---------------------------------->
                  INV B Also:A ReqBy:A
                  <-----------------
                  200 OK Also:D
                  ----------------->
               INV A Also:A,B
<---------------------------------------------------
               200 OK Also:C
---------------------------------------------------->
                                    INV D Also:A,B ReqBy:B
                                  ------------------>
                                     200 OK
                                  <------------------
                                        ACK
                                  ------------------->
                               200 OK
                  <------------------------------------
                                ACK
                  ------------------------------------>
                      200 OK
                  <------------------
                        ACK
                  ------------------>
```

Figure 4: Add Party During Add Party Message Flow

know about, containing the SDP describing the multicast session. The result is that some or all of the sessions on the call-legs transition to multicast. If not all have transitioned, the user may still need to send some packets unicast.

There is no capability for determining the codec parameters for the multicast session based on the intersection of the capabilities of each participant. The model for multicast media distribution in a tightly coupled conference is identical to that for loosely coupled sessions. The initiator of the multicast session chooses a codec, and that is what is used. Note, however, that in the case where the

```
A              B                C                 D

                 INV D Also:B,C
-------------------------------------------------->
                   BYE B
               <-----------------
             BYE A
<--------------------------------
                   200 OK
               ----------------->
        200 OK
-------------------------------->
                       INV B Also:A,B,C ReqBy:A
               <-----------------------------------
                       200 OK Also:C;status=inactive
               ----------------------------------->
                        ACK
               <-----------------------------------
                200 OK
<--------------------------------------------------
             ACK
-------------------------------------------------->
```

Figure 5: Party Leaves During Add Message Flow

sessions start as multi-unicast, the originator will know the capabilities of all the other parties, and thus can intelligently choose the codecs for the session.

10 Security Considerations

Security issues are addressed throughout this document.

The call control mechanisms have serious security issues. An INVITE with an Also cause the recipient to add or drop other parties, possibly without user interaction. This implies that authorization of the requests is critical.

11 Open Issues

There are many, many open issues:

1. How to do this with shared secrets rather than public keys?

2. If the transferred-to party in a transfer accepts some, but not all (or rejects some, but not all) of the INVITEs for it, we end up with a partially disconnected conference.

3. Should we use a session timer to refresh things and periodically re-flood the participant list, in an attempt to keep things synchronized?

4. The version/status concept is still very vague. Does it work? Is it needed?

5. Conference out of consultation for multi-party calls - not clear the Replaces mechanism works here.

12 Acknowledgements

The authors would like to especially thank Jonathan Lennox for his many insightful comments and contributions to this work.

13 Bibliography

[1] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments (Best Current Practice) 2119, Internet Engineering Task Force, Mar. 1997.

[2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments (Proposed Standard) 2543, Internet Engineering Task Force, Mar. 1999.

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments (Proposed Standard) 1889, Internet Engineering Task Force, Jan. 1996.

[4] C. Elliott, "A 'sticky' conference control protocol," vol. 5, pp. 97--119, 1994.

[5] S. Hanna, B. Patel, and M. Shah, "Multicast address dynamic client allocation protocol (MADCAP)," Internet Draft, Internet Engineering Task Force, May 1999. Work in progress.

Table of Contents