Internet Engineering Task Force SIPPING WG
Internet Draft J.Rosenberg,H.Schulzrinne
draft-rosenberg-sip-conferencing-models-01.txt dynamicsoft,Columbia U.
July 20, 2001
Expires: February 2002

Models for Multi Party Conferencing in SIP

STATUS OF THIS MEMO

Abstract

The Session Initiation Protocol (SIP) can support multi-party conferencing in many different ways. In this draft, we define the various multi-party conferencing models, and for each, discuss how they are used and then analyze their relative benefits and drawbacks.

1 Introduction

The Session Initiation Protocol (SIP) [1] has been defined for the establishment, maintenance, and termination of calls between one or more users. However, despite its origins as a large scale multiparty conferencing protocol, SIP is used today primarily for point to point calls. This configuration is the focus of the SIP specification and most of its extensions. As a result, there is a lot of confusion about how SIP supports multi-party conferencing.

We seek to remedy this problem by describing, in a consistent and complete fashion, the various multi-party conferencing models supported by standard SIP. For each model, we discuss:

- How the model works.
- How users are invited to join.
- How users can join an existing conference without being invited
- How well the model scales.
- Which entities need to be aware of the model.
- How participants learn about each other.

We also identify missing pieces and reccomend standard activity to fill them in. This document itself does not define any new extensions of any kind. However, several scenarios discussed in the draft make use of existing extensions to SIP.

2 End System Mixing

The first model we call "end system mixing". In this model, user A calls user B, and they have a conversation. At some point later, A decides to conference in user C. To do this, A calls C, using a completely separate SIP call. This call uses a different Call-ID, different tags, etc. There is no call set up directly between B and C. A receives media streams from both B and C, and mixes them. A sends a stream containing A's and C's streams to B, and a stream stream containing A's and B's streams to C.

This model is depicted graphically in Figure 1.

Basically, user A handles both signaling and media mixing. B and C are unaware of the multi-party call, from a SIP perspective at least. From an RTP perspective, A is a mixer, and so the RTCP reports from A will contain SDES information that indicates the existence of an additional party in the media stream.

Note that this model has the serious drawback that the conference ends when the mixing UA leaves the call.

OPEN ISSUE: Another problem with this approach is that there is no specific way for A to determine when a

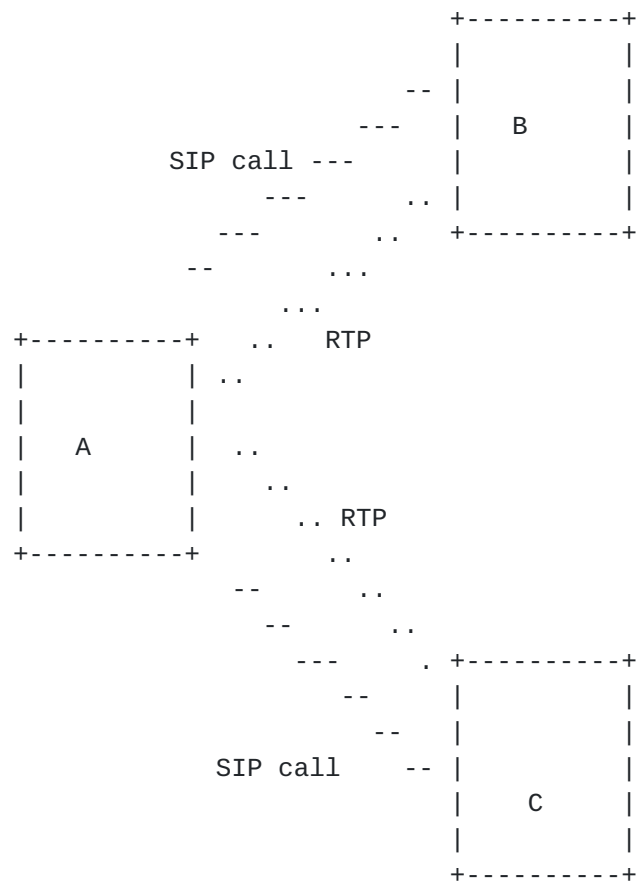

Figure 1: Three Way Calling using End System Mixing

signaling message it receives was meant just for it, or for the entire conference. For example, if B sends a REFER to A, pointing to user D, was this REFER meant for A alone, or for A and C? If it was meant for A and C, presumably A would act upon the REFER and send it to C as well. C too would act on the REFER. This would cause two separate REFER-triggered INVITEs to get routed to D. How would D know that both INVITEs need to be mixed together as a conference? What if it cannot support this capability?

Because the three-way calling approach works only for the most basic case, we do not recommend it as a general solution.

2.1 Inviting Users to Join

Any user in the conference can invite another user to join, so long as they are capable of performing the required mixing and signaling functions. To invite a new user to join, a user in the conference simply calls them using normal SIP procedures. The only difference is that the stream sent to that new user contains the streams received from the other parties in the call.

In fact, it is acceptable for complex connectivity graphs to be constructed, as a result of different users inviting other users to join. For example, take our case of A calling B, and then calling C. If, later on, C calls D, C will performing the mixing of the streams it gets from A (which actually contain media from A and B), along with its own stream, and send that to D. This results in a connectivity graph that looks like Figure 2.

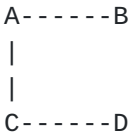

Figure 2: Connectivity Graph

Note, however, that there is a possibility of loops. From here, if D calls B, and brings that stream into the conference, a loop is created. This loop can be detected using the mechanisms described in the RTP specification [2]. However, we expect these conditions to be extremely rare. Presumably, D knows B is in the conference already, and so would not likely call B and invite them in.

A serious problem with the more complex topologies is that the departure of a participant might cause a partition of the conference into several sub-conferences which cannot easily be healed.

2.2 Users Joining

In this model, there is not any explicit conference "identifier" that can be used to join. This conference model, by its nature, is built around ad-hoc conferences. However, it is still possible for a user to join in the following way.

Lets say a new user, E, simply calls B, unaware even, that B is in a

conference (E might actually be aware, but the SIP messaging is no different). B's softphone, recognizing that B is already in a conference, asks B if E should be brought into the conference right away. If B clicks "yes", the call to E is answered. The media stream sent to E contains media from B, along with the media B is already receiving from A.

If B had instead clicked no, E can easily be added to the conference later. No SIP signaling at all is needed to do this. B simply starts sending the mixed media to E.

2.3 Scalability

A drawback of this model is its scalability. Viewing the conference from a graph perspective, if the number of edges touching a vertex (its degree) equals N, the user corresponding to that vertex has to perform up to N separate media stream encodings. We say "up to", as it depends on the number of paricipants who are talking at once. If only one participant is talking, the non-talking "mixer" endpoints don't need to do any additional encoding. If everyone is talking, it is N encodes. Since encoding is generally a complex process, a typical workstation these days can handle two or three simultaneous encodes using a low rate codec like G.723.1. The problem can be mitigated somewhat by distributing the mixing responsibilities (making the graph deep rather than wide). However, this requires a conscious effort of the participants regarding who is to make the call to add a new user. This is unlikely to happen in practice.

Another limitation to scalability is bandwidth. If the degree of a vertex is N, the user needs enough bandwidth to send and receive up to N streams, for a total of 2N. On a 56K modem, using a G.723.1 codec, this limits the degree to two (remember RTP overheads). This limitation exists even if only one user is talking. In this case, a mixing host receives the encoded packet stream, and needs to send a copy to each participant it is connected to.

For these reasons, this conferencing model is ideal for three-way conferences (i.e., degrees of two), but doesn't scale up much higher.

2.4 Location of Service Logic

This model does not require any extension to SIP in order to work. It does require knowledge of this mechanism within the UA performing the mixing. Non-mixing participants do not need to know anything special.

2.5 Discovering Participant Identities

The identities of other participants in the conference is NOT known

through SIP. Rather, it is learned through RTP. UAs with degrees greater than one are RTP mixers. As such, they take the RTCP SDES of the streams they mix, and aggregrate them into the RTCP stream sent out. Since RTCP messages are sent infrequently, there may be a delay between when a user joins, and when their presence is known to the other participants.

3 Large-Scale Multicast Conferences

Large-scale multicast conferences were the original motivation for both the Session Description Protocol (SDP) [3] and SIP. In a large-scale multicast conference, one or more multicast addresses are allocated to the conference (more than one may be needed if layered encodings are in use). Each participant joins that multicast groups, and sends their media to those groups. Signaling is not sent to the multicast groups. The sole purpose of the signaling is to inform participants of which multicast groups to join.

Large-scale multicast conferences are usually pre-arranged, with specific start and stop times (which is why this information exists in SDP). Protocols such as the Session Announcement Protocol (SAP) [4] are used to announce these conferences. However, multicast conferences do not need to be pre-arranged, so long as a mechanism exists to dynamically obtain a multicast address. SAP itself was originally used for this purpose; this has been supplanted by the malloc architecture [5], still under development.

So, if there are N participants, there will be point-to-point SIP relationships with pairs of participants. Each participant sends a single media stream to the group, and receives up to N-1 streams at any time. Note that the number of streams that a user will receive depends on who is actually sending at any given time. If the stream is audio, and silence suppression is utilized, the number of streams a user will receive at any given time is equal to the number of users talking at any given time. Even for very large conferences, this is usually just a small number of users.

3.1 Inviting Users to Join

Inviting users to join is simple. Any user may invite any other user to join. The SIP INVITE request contains SDP that indicates multicast addresses for each media line. The SDP in the 200 OK response may actually be empty. From Section B.3 of RFC2543:

> For multicast, receive and send multicast addresses are the same and all parties use the same port numbers to receive media data. If the session description provided by the

caller is acceptable to the callee, the callee can choose not to include a session description or MAY echo the description in the response.

The called party then joins the multicast groups indicated in the SDP, using multicast protocols such as IGMP [6]. Note that it is not even necessary for users to send each other BYE messages when the conference is over, especially for large-scale, pre-arranged conferences that have explicit end times indicated in SDP.

OPEN ISSUE: Do we need to specify a SIP mechanism for indicating that no BYE is needed?

SDP aside, a participant can simply leave the conference at any time by leaving the multicast groups. No SIP signaling is needed to accomplish this.

3.2 Users Joining

Users can join a conference of this type without being invited. All they need is the multicast addresses, ports, and codecs being used. These can be obtained through any number of means, including SAP. SDP conference descriptions can even be obtained from web pages, for example.

Once the addresses are obtained, the user simply joins the appropriate multicast groups. Note that absolutely no SIP signaling is required in this case.

3.3 Scalability

The scalability of conferences of this type is can be excellent, especially for audio conferences. However, it is scalable under the assumption that multicast itself can scale to very large groups. Indeed, in local networks, protocols like DVMRP [7] and PIM-DM have tremendous scalability for conferences with very large numbers of members (the so called dense modes). Given the existence of scalable multicast, the primary bottleneck to scalability of this conference type is the periodicity of RTCP reporting. Work has been done on improving the problematic cases [8] so that conferences with well over a million members are possible.

Scaling is a bit harder for video conferences. Unlike voice, where silence suppression allows for no data to be sent during periods of inactivity, the same is not the case for video. This makes it hard to scale without flooding users with lots of video packets.

Security is also hard for multicast conferences. Group key management, especially when users leave the group, is very complex.

Unfortunately, multicast has not been widely deployed across backbones (some do, like Internet2, but they are the exception rather than the rule). The MBone has collapsed, for all intents and purposes. Very few ISPs support multicast. As a result, wide area conferences are not really viable using multicast. However, these conferences are very suitable for LAN or enterprise conferences, where multicast is often deployed.

3.4 Location of Service Logic

This conferencing model does not require any SIP extensions. It does require that SIP UAs are prepared to receive SIP invitations with multicast addresses in the SDP. These UAs need to be prepared to mirror the SDP in the response. They should also be prepared to never receive a BYE for the conference.

3.5 Discovering Participant Identities

The identity of the participants in the session is learned entirely through RTCP. Each user a group multicasts RTCP packets with their name, email address, and so on. Note, however, that in large conferences, there may be significant amounts of time between a participant joining, and sending of their first RTCP SDES packet (this is for receivers only; senders will become known much faster).

4 Dial-In Conference Servers

Dial-In conference servers closely mirror dial-in conference bridges in the traditional PSTN.

A dial-in conference server acts as a normal SIP UA. Users call it, and the server maintains point to point SIP relationships with each user that calls in. The server takes the media from the users who dial into the same conference, mixes them, and sends out the appropriate mixed stream to each participant separately.

The model is depicted in Figure 3. Note that each UA (A,B,C,D) has a point to point SIP and RTP relationship with the conference server. Each call has a different Call-ID. Each user sends their own media to the server. The media delivered to user A by the server is the media mixed from users B,C and D. The media delivered to user B by the server is the media mixed from users A, C and D. The media delivered to user C by the server is the media mixed from users A, B and D. The media delivered to user D is the media mixed from users A, B and C

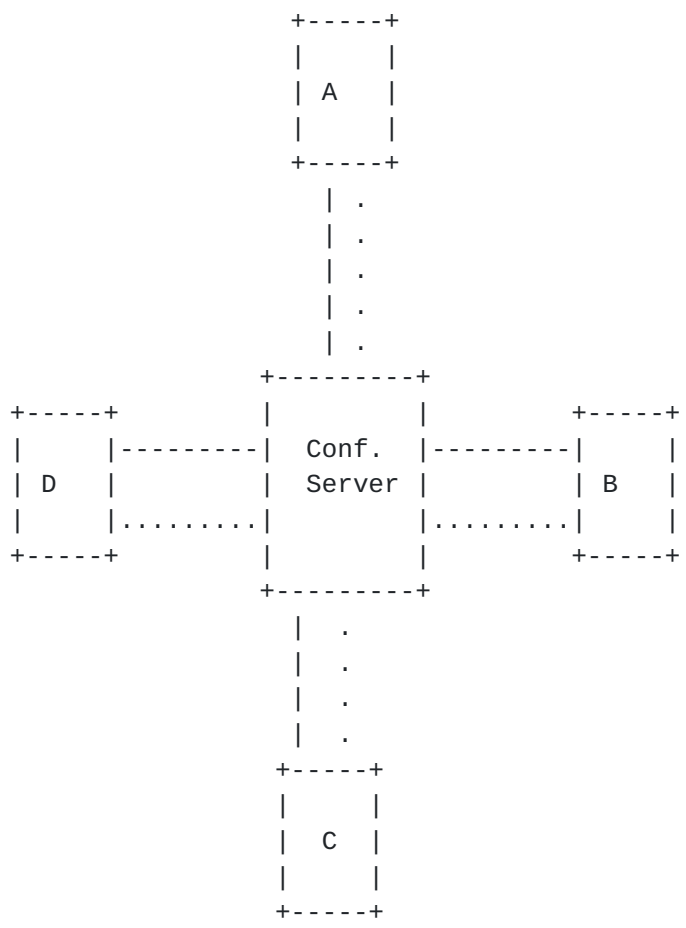

Figure 3: Dial-In Conference Servers

(this is also known as a mix-minus configuration).

The conference is identified by the request URI of the calls from each participant. This provides numerous advantages from a services and routing point of view [9]. For example, one conference on the server might be known as sip:conference34@servers.com. All users who call sip:conference34@servers.com are mixed together.

Dial-In conference servers are usually associated with pre-arranged conferences. However, the same model applies to ad-hoc conferences. An ad-hoc conference server creates the conference state when the first user joins, and destroys it when the last one leaves. The SIP

and RTP interfaces are identical to the pre-arranged case.

Since conferencing servers are nothing more than SIP UASes, they can use any of the procedures SIP allows a UAS to use. This includes authentication. So, for example, a specific conference may have a password associated with it. Users who join are challenged (with a 401) using digest authentication. The realm, in this case, would identify the conference. The INVITE that comes back would have an Authorization header that includes the response to the challenge - the name of the user trying to join the conference, and the conference password, hashed as defined in [10].

Conferences can also limit the number of participants. When a new user tries to join, but the conference is full, the conference server can just reject the request with a "500 Conference Full" response.

4.1 Inviting Users to Join

Inviting users to join is done using the SIP REFER message [11]. If user A wishes to ask user B to join, A would send B a REFER that looks like:

```
REFER sip:B@example.com SIP/2.0
From: sip:A@example.com
To: sip:B@example.com
Refer-To: sip:conference34@servers.com
```

This would cause B to send an INVITE message to the conference server:

```
INVITE sip:conference34@servers.com
From: sip:B@example.com
To: sip:conference34@servers.com
Referred-By: sip:A@example.com
```

Since the request URI identifies the conference, this will cause B to get added to conference 34.

An additional mechanism for inviting a user to join is to send REFER from A to the conference server, with a Refer-To containing the address of B. This REFER would look like:

```
REFER sip:conference34@servers.com SIP/2.0
From: sip:A@example.com
To: sip:B@example.com
Refer-To: sip:B@example.com
```

This approach has the advantage that it doesn't require REFER support from B, only from the conference server.

> OPEN ISSUE: A problem with the mechanisms for adding a user is that they assume that the UA for user A (the one who adds another user to the conference) knows that it is indeed talking to a conference server. If the mechanisms in this section were applied to a UA which was not a conference server, the result would be the creation of additional call legs, but not a conference. This means that we require some mechanism for identifying that a URL is a conference URL.

4.2 Users Joining

It is easy for users to join the conference. The participant that wishes to join simply sends an INVITE to the conference server, with the conference ID in the request URI. The conference ID (which is a SIP URL), can be learned by any number of means, including having it on a web page, receiving it in an email, etc.

For example, if B wishes to join sip:conference34@servers.com, B would send the following request:

```
INVITE sip:conference34@servers.com
From: sip:B@example.com
To: sip:conference34@servers.com
```

4.3 Scalability

The scalability of this model is limited by the bandwidth and processing power of the conference server. If there are N participants in a conference, M of which are sending media streams, the server will need to manage N signaling relationships, perform M RTP stream decodes, and N RTP stream encodes (assuming M > 0). The encoding is the primary processing bottleneck, and the sending of the N media streams is the primary bandwidth bottleneck. However,

conference servers can be built using heavy duty hardware, and have high bandwith access.

Furthermore, since we are using the request URI to name the conferences, we can use standard SIP techniques for distributing conferences across servers [9].

4.4 Location of Service Logic

The SIP UA of the conference participants does not require any special processing. The RTP implementation in those clients, however, should support RTCP and be prepared to receive contributing sources.

All of the new logic for providing this service resides in the conferencing server. No SIP extensions are needed, simply logic that resides above the SIP stack to manage the conferencing service.

4.5 Discovering Participant Identities

The identities of other participants in the conference are NOT known through SIP. Rather, it is learned through RTP. THe conference server is an RTP mixer. As such, it takes the RTCP SDES of the streams it mixes, and aggregrates them into the RTCP stream sent out. This will allow participants to gradually (over a few seconds), learn the identities of the other participants.

As an implementation choice, the conference server can generate the RTCP SDES of its participants, rather than using those provided by the participants. The reason for this is authenticity. A conference server can use SIP authentication mechanisms to identify the participants in the conference. This may allow it to validate the RTCP SDES provided by the participants. A conference server could remove any false information, and regenerate the SDES using the correct user identity as validated through SIP.

5 Ad-hoc Centralized Conferences

In an ad-hoc centralized conference, two users A and B start with a normal SIP call. At some point later, they decide to add a third party. Instead of using end system mixing, they would prefer to use a conference server, as defined in Section 4.

The call flow for starting this kind of conference is shown in Figure 4. Initially, A calls B (1-3). At some point, B decides to add a user, C, to the call, and begins the transition to a conference server. The first step in this process is the discovery of a conference server that supports ad-hoc conferences. This can be done

through static configuration, or through any of a number of standard service discovery protocols, such as the Service Location Protocol [12].

Once the server is discovered, a conference ID is chosen. This ID must be globally unique. The conference ID is then prepended to the server, and a SIP URL for the ad-hoc conference is formed. For example, if the server "a.servers.com" is used, and the unique ID is "a7hytaskp09878a", the SIP URL for this conference is sip:a7hytaskp09878a@a.servers.com.

B then sends an INVITE to this URL (4). This creates the initial conference state in the server. The conference server accepts the call (5) and B sends an ACK (6). B then sends a REFER to A (7), referring them to sip:a7hytaskp09878a@a.servers.com. A accepts the referral (8) and this triggers an INVITE to this address (9). This causes A to be added to the conference. The conference server accepts the INVITE (10), and an ACK is generated (11). Once the NOTIFY request (indicating successful completion of the referred call) is sent from A to B (12), A responds with a 200 OK. Since B is now assured that A is connected through the conference server, B hangs up to A with a BYE (14).

> OPEN ISSUE: Its not clear that this is the best flow. An alternative flow is for B to REFER the conference server to A, using a call replacement mechanism. This is probably more correct, since this is not so much a transfer as a call leg replacement.

Finally, B can add C to the call. This is identical to the procedures described in Section 4 for adding userst to the conference. First, B generates a REFER (16) to C. The Refer-To header contains the conference URL, sip:a7hytaskp09878a@a.servers.com. C responds to the referral with a 200 OK (17). C then INVITEs itself to the conference (18-20). C then generates a NOTIFY informing B that the REFER has completed (21).

It is also possible to transition from a end system mixed conference (even one with a complex connection topology), to a centralized conference server. Consider a end-system mixed conference with the topology of Figure 2. User A wishes to transition to a centralized conference server in order to add another participant. The transition is shown in Figures 5 and 6.

First, user A discovers a conference server, and creates a new

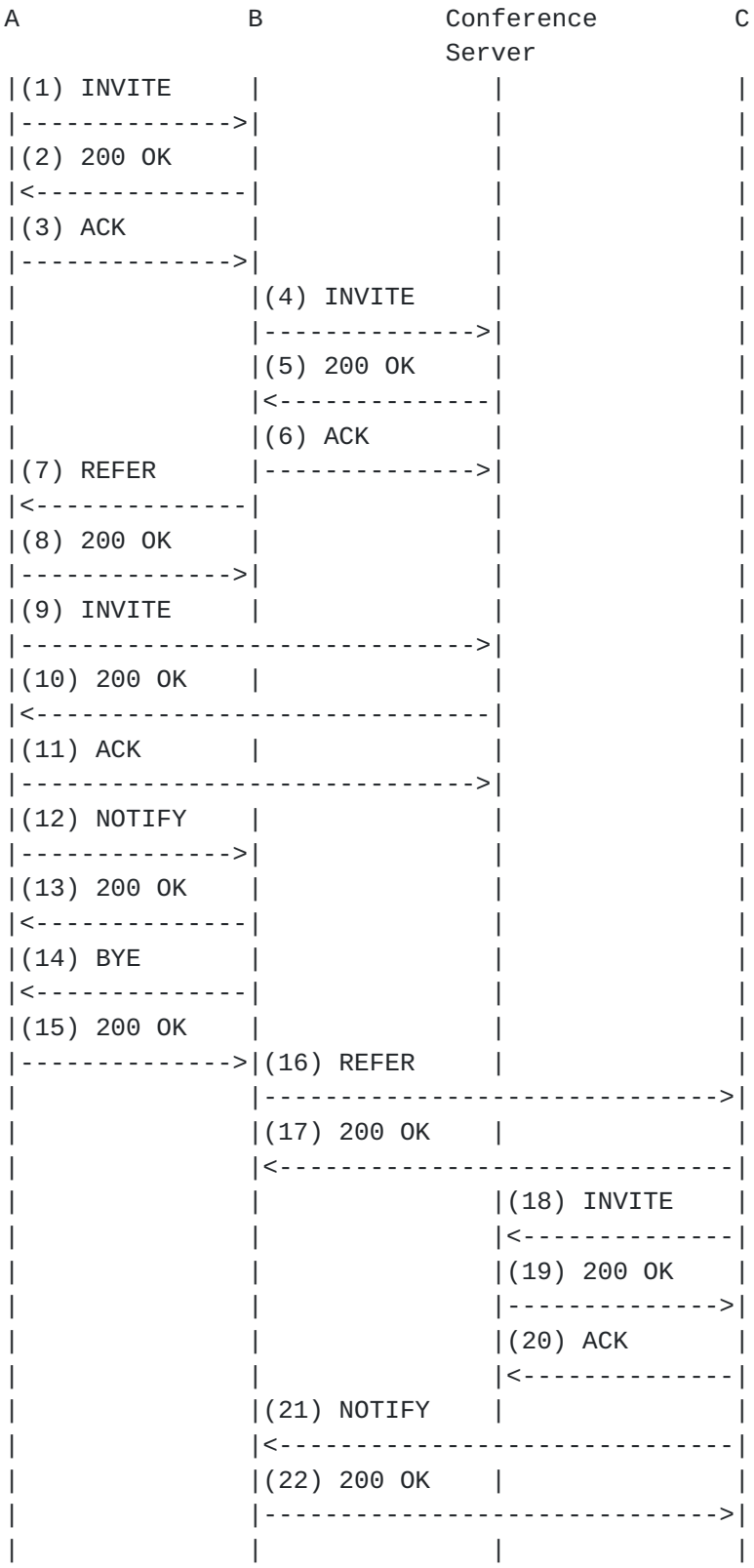
A
B
Conference Server
C
(1) INVITE
(2) 200 OK
(3) ACK
(4) INVITE
(5) 200 OK
(6) ACK
(7) REFER
(8) 200 OK
(9) INVITE
(10) 200 OK
(11) ACK
(12) NOTIFY
(13) 200 OK
(14) BYE
(15) 200 OK
(16) REFER
(17) 200 OK
(18) INVITE
(19) 200 OK
(20) ACK
(21) NOTIFY
(22) 200 OK

Figure 4: Transitioning to ad-hoc

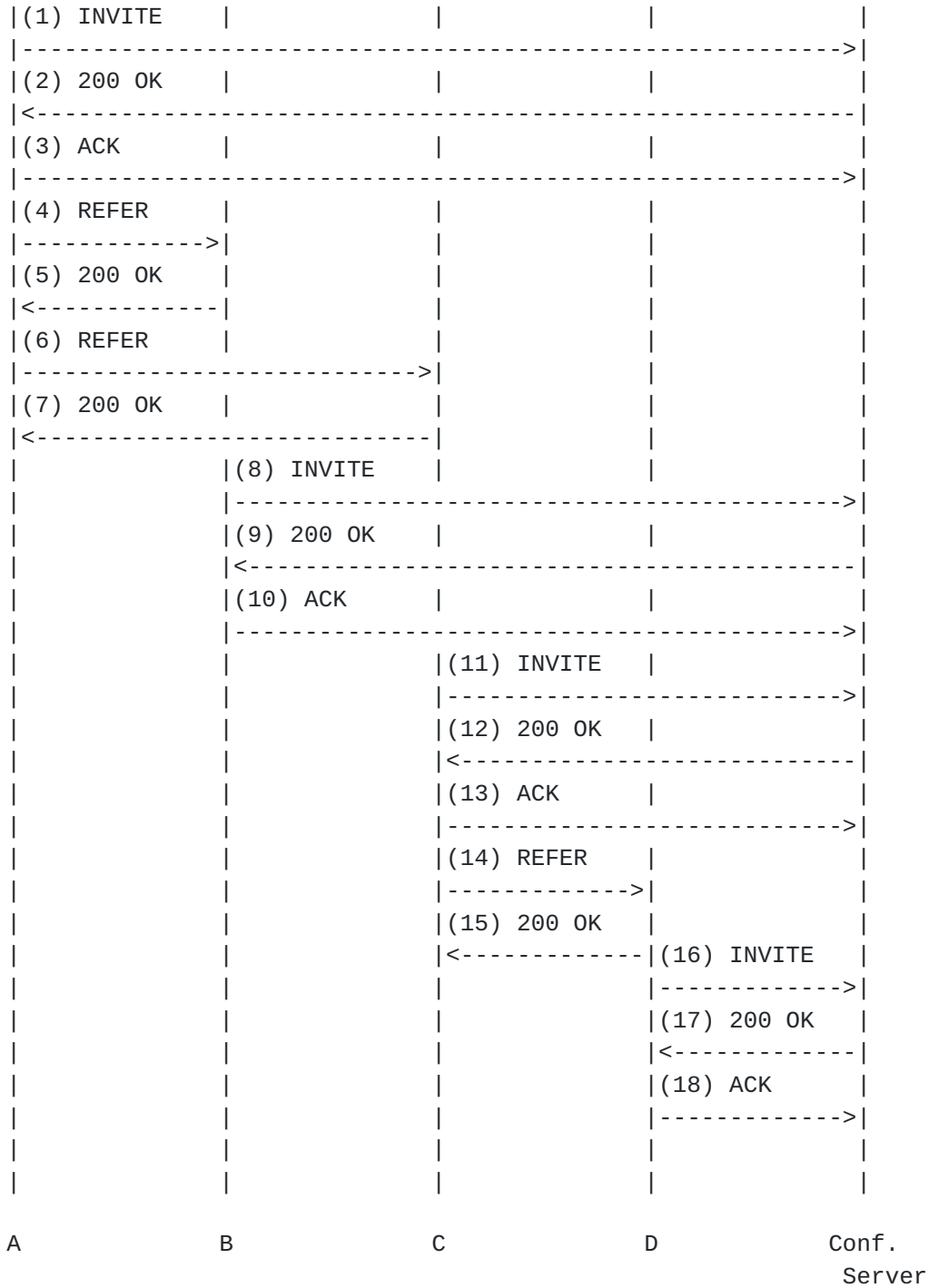

Figure 5: Adhoc transition from end-system mixed: part I

conference by sending an INVITE to it (1-3). A then REFERs the two end systems it is connected to (B and C), to the server (4-5 and 6-7 respectively). This causes B to INVITE itself to the conference server (8-10), and C to do the same (11-13). Since C had gotten a REFER from B, it "passes it on" to D by sending a REFER to it (14-15). This causes D to join the conference server by sending it an INVITE (16-18).

Once the REFER triggered INVITEs complete, notifications start to get sent. Since B completed first, it will be the first to send a NOTIFY to A (19) followed by C (21). At this point, A can terminate its legs to B and C (23-24 and 25-26 respectively). Since D completed its REFER triggered INVITE next, it generates a NOTIFY to C (27). This causes C to terminate its leg with D (29). The call has now transitioned to a centralized server.

> OPEN ISSUE: There is no way for A to know that the entire conference has transitioned. Also, as above, its not clear that a REFER from the conference server wouldn't be better.

Once the conference has been formed, further operation is identical to the dial-in conferencing model of Section 4. The only difference in the conferences is that the conference identifier is dynamic in this case, and static in Section 4. This makes users asynchronously joining nearly impossible.

5.1 Inviting Users to Join

Once the ad-hoc conference has been created on the server, inviting users proceeds as defined in Section 4.1.

5.2 Users Joining

Once the ad-hoc conference has been created on the server, joining proceeds as defined in Section 4.2.

5.3 Scalability

The scalability of this conference model is identical to that of dial-in conference servers, as described in Section 4.3.

5.4 Location of Service Logic

The logic for handling the transition process must be located in at least one UA in the conference. All UAs that are mixers in a end system mixed conference must know to propagate the REFER requests they receive during the transition.

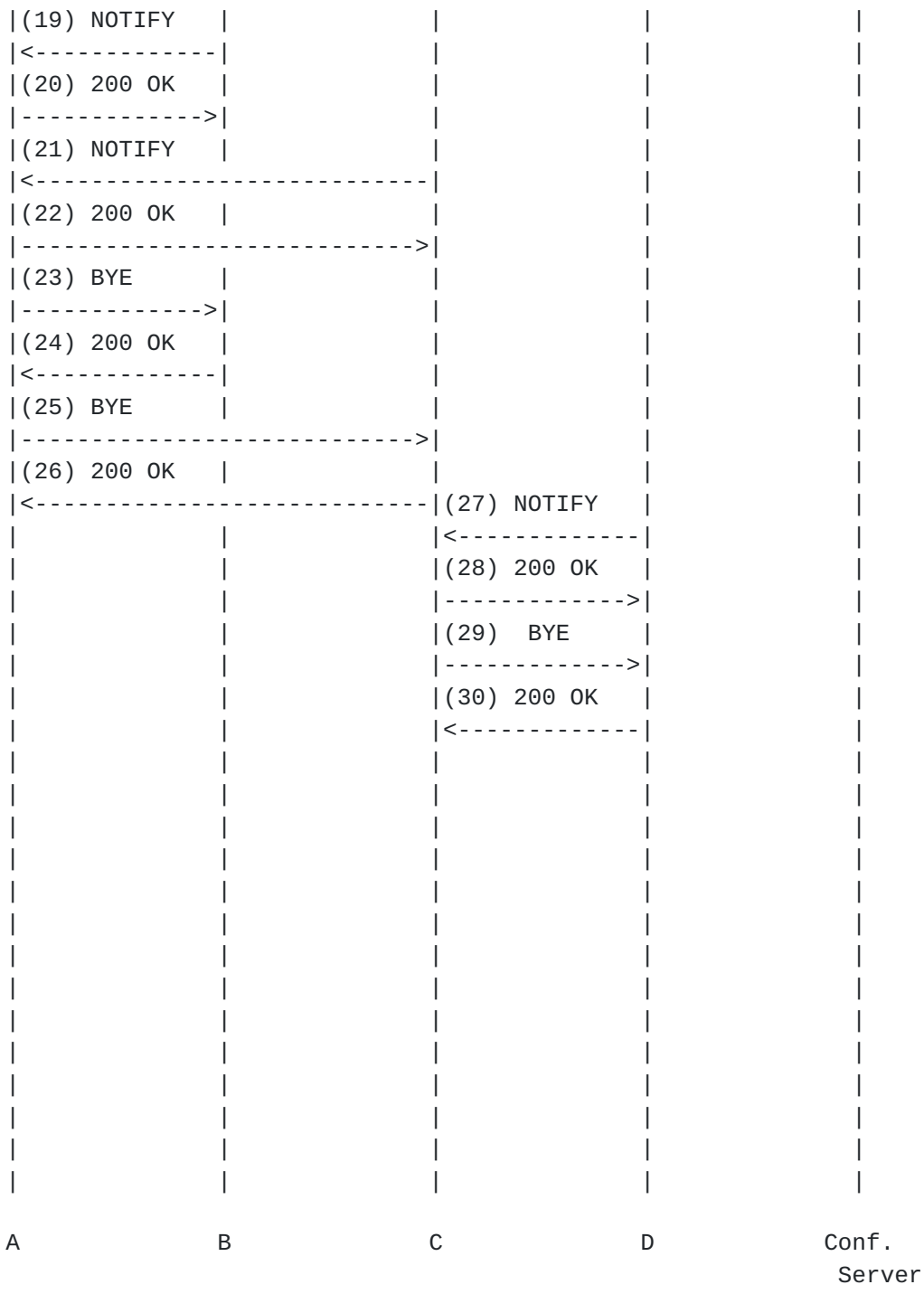

Figure 6: Adhoc transition from end-system mixed: part II

5.5 Discovering Participant Identities

Once the ad-hoc conference is established, conference identities are

6 Dial-Out Conferences

Dial-out conferences are a simple variation on dial-in conferences. Instead of the users joining the conference by sending an INVITE to the server, the server chooses the users who are to be members of the conference, and then sends them the INVITE. Typically dial out conferences are pre-arranged, with specific start times and an initial group membership list. However, there are other means for the dial-out server to determine the list of participants, including user presence [13]. The model in no way limits the means by which the server determines the set of users.

Once the users accept or reject the call from the dial out server, the behavior of this system is identical to the dial-in server case of Section 4. Thus, a dial-out conference server will generally need to support dial-in access for the same conference, if it wishes to allow joining after the conference begins.

Note that, from the participants perspective, they will learn the conference identity (the URL) from the From field in the INVITE messages received from the server.

OPEN ISSUE: Or is the Contact more appropriate?

6.1 Inviting Users to Join

Once the conference is established, inviting users to join is identical to the scenario described in Section 4.1. Note that the URL to be used in the REFER is obtained from the From field of the INVITE received from the dial-out server.

6.2 Users Joining

Once the conference is established, joining is identical to the scenario described in Section 4.2. Note that the URL to be used in the INVITE of new participants is obtained from the From field of the INVITE received from the dial-out server by the initial participants.

6.3 Scalability

The scalability of this conference model is identical to that of dial-in conference servers, as described in Section 4.3.

6.4 Location of Service Logic

The SIP UA of the conference participants does not require any special processing. The RTP implementation in those clients, however,

should support RTCP and be prepared to receive contributing sources.

All of the new logic for providing this service resides in the conferencing server. No SIP extensions are needed, simply logic that resides above the SIP stack to manage the conferencing service.

6.5 Discovering Participant Identities

Once the conference is established, conference identities are determined through RTCP, as in the dial-in case.

7 Centralized Signaling, Distributed Media

In this conferencing model, there is a centralized controller, as in the dial-in and dial-out cases. However, the centralized server handles signaling only. The media is still sent directly between participants, using either multicast or multi-unicast. Multi-unicast is when a user sends multiple packets (one for each recipient, addressed to that recipient). This is referred to as a "Decentralized Multipoint Conference" in H.323. Interestingly, this conference model is possible with baseline SIP.

It works through third party call control [14]. The conference server uses re-INVITEs to each participant when a new one joins. The re-INVITEs add a media stream that gets sent to the new participant (and similarly in the reverse direction).

Let us assume for the moment that a conference already exists with three participants. In this state, each participant is sending media directly to each other. This is because the SDP that the conference server has given to each participant contains three media lines, each of type audio, with connection addresses and ports corresponding to each of the three users.

The call flow from here is shown in Figure 7. In the figure, the word after the INV or SIP response code refers to the connection adress(es) in the SDP in the message. +X means the addition of a stream with X as the recipient address.

A new participant joins the conference. It does so by sending an INVITE (1)to the server, with the conference ID in the request URI. The SDP in the INVITE contains a single media stream, with an IP address and port where it would like to receive media (D). The 200 response from the conference server (2) contains a single media line with an IP address of 0.0.0.0 and a random port, indicating hold.

The next step is for the server to obtain two more addresses where

the new participant will be receiving media (it already has one from the original INVITE). To do this, it sends a re-INVITE to the new participant (4). This re-INVITE contains two additional media streams (for three total), all three of which are on hold. The 200 response to the re-INVITE (5) contains two additional IP addresses and ports where the user is willing to receive media.

Now the server needs to inform the other parties that they should begin sending media to the new user. It first sends a re-INVITE to user C (7). This re-INVITE adds an additional media stream to the two already that C has been sending. This new media stream uses one of the three connection addresses and ports returned by D in message (5). Call this address/port D1. The other two are D2 and D3. The 200 OK response from user C (8) contains the address and port where C is willing to receive a new, third media stream. Call this port C3. The server holds on to this port, as it will use it later on, sending it to D, so that D sends media there. At this point, however, C can begin sending media to D.

This re-INVITE process happens for B and for A as well. In the re-INVITE to B (10), the server adds an additional media line (above the two already in use by C) using address/port D2. The response (11) contains a new address/port to send media to B. Call this port B3. In the re-INVITE to A (13), the server adds an additional media line using address/port D3. The response (14) contains a new address/port to send media to A. Call this port A3.

Finally, the server sends a re-INVITE (15) to the new party. This re-INVITE takes all three streams off hold, and updates their connection addresses and ports with C3, B3, and A3, respectively. The 200 OK response (16) returns the same ports and addresses returned in message (5) (as noted in [14], these addresses/ports MUST NOT change). Now, D can send media to A,B and C.

The result of these manipulations is, indeed, a full mesh of unicast RTP streams between all participants. Unlike the case of end system mixing, the stream sent by any participant to all of the others is identical. Each particpant needs to mix, but it mixes the media it receives, and plays that out the speakers. This is normal behavior for multiple streams of the same type. Note that the SIP relationship is still point-to-point. There are four calls at the end of Figure 7, one from each participant to the server, each with a different Call-ID.

Note that hybrids are easily possible. Certain users can instead be mixed (sending audio to the conference server), while others are set to send audio to each other.

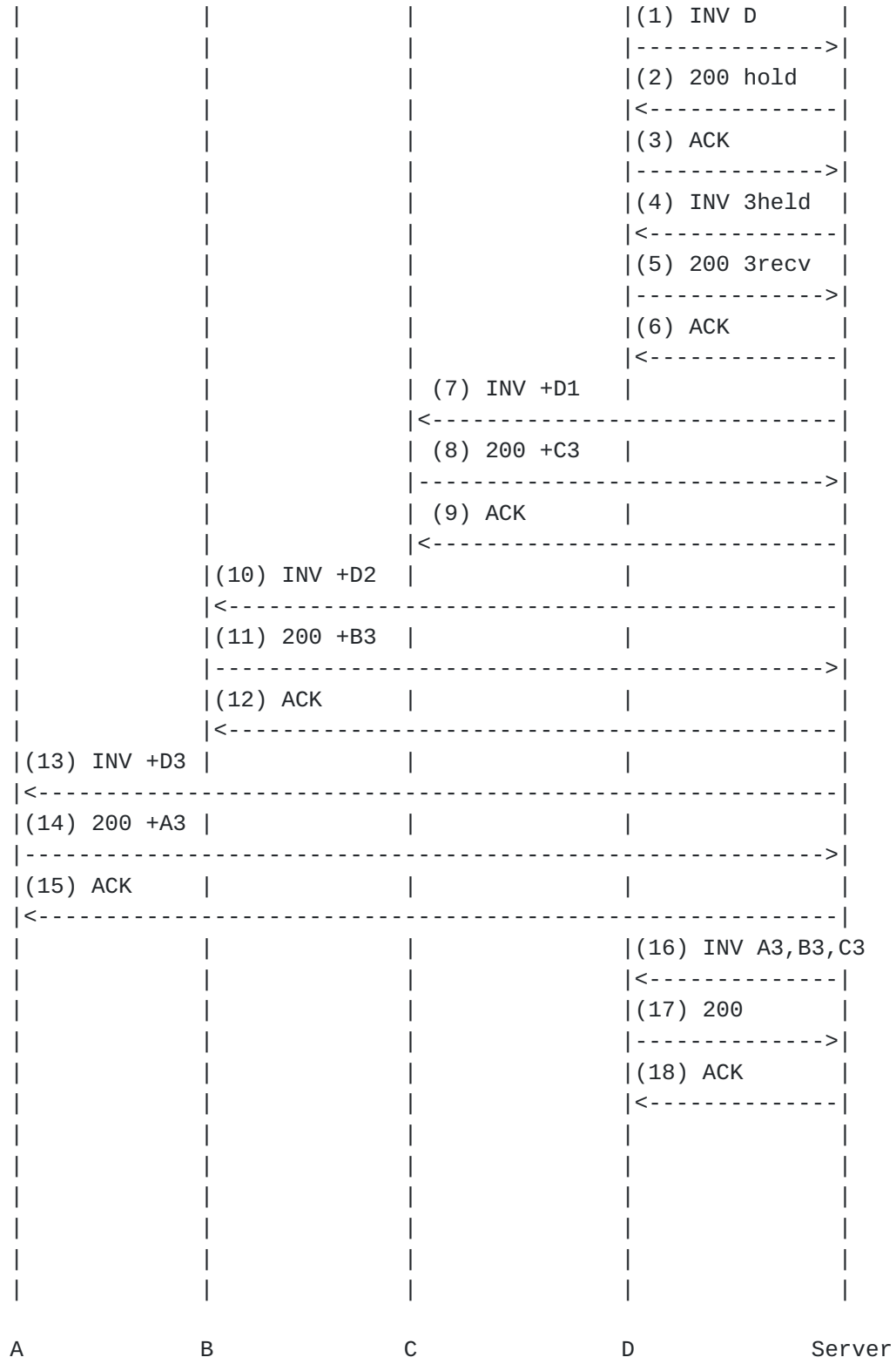
(1) INV D
(2) 200 hold
(3) ACK
(4) INV 3held
(5) 200 3recv
(6) ACK
(7) INV +D1
(8) 200 +C3
(9) ACK
(10) INV +D2
(11) 200 +B3
(12) ACK
(13) INV +D3
(14) 200 +A3
(15) ACK
(16) INV A3,B3,C3
(17) 200
(18) ACK
A
B
C
D
Server

Figure 7: Centralized Signaling, Decentralized Media

7.1 Inviting Users to Join

Inviting users to join works identically to the dial-in conference bridge scenario 4.

7.2 Users Joining

A user joins in the same way described in section 4.

7.3 Scalability

The scalability of this conferencing model depends on many factors. From a media perspective, the conference server never even touches a single media stream. However, for N participants, each participant needs to be able to receive, decode, and mix N-1 media streams. For users accessing the server through dial-in modems, this will severely limit the sizes of these conferences. However, the processing burden is much less than that of the end system mixing model. This is because each end user needs to decode N-1 streams, but only encode 1. Decoding is much, much cheaper than encoding, so supporting many decodes is not necessarily a problem. This is especially the case when silence suppression is in use. In that case, streams are only sent by talking users. This means any given user only needs to decode (and receive) as many streams at a time as there are users talking. THis can vastly improve scalability of the conference.

There is a signaling burden on the server, however. If there are N users in the conference, addition of a new user (the N+1th) requires N+3 INVITE transactions, each of which has three messages. Similarly, departure of a user requires N BYE transactions, each of which has 2 messages. For large N, and highly dynamic conferences, this can represent a potential burden. However, we believe this bottleneck is much farther out than the processing and bandwidth bottlenecks at the end users.

For these reasons, we believe this conference model is ideal in corporate enterprises, where bandwidth is more plentiful and PCs are generally faster.

7.4 Location of Service Logic

Nearly all of the logic for implementing this conferencing service lives in the server itself.

The only requirement from the end users is that they support multiple, parallel media streams of the same type, and that they be prepared to mix those streams together. They must also support the third party control primitives [14], which don't require anything

beyond baseline SIP, but are not likely supported unless explicit actions are taken to do so.

It is this combination - no need for media processing in the server, combined with no need for specialized SIP processing in the end systems, that makes this model attractive.

7.5 Discovering Participant Identities

Conference identities are discovered through RTCP. Each user will receive N-1 RTP streams, each of which has its own RTCP channel that carries the participant identification.

8 Summary of Models

Table 1 shows a summary of the differences between the various models.

Table 1: Summary of Models

Name	signaling	media	inviting	joining	discovering	scale
End-Mixing	tree	tree	normal invite	normal invite	RTCP	small
Multicast	pairs	m-cast	normal invite	multicast join	RTCP	large
Dial-Up	star	star	refer	normal invite	RTCP	medium
Ad-Hoc	star	star	refer	normal invite	RTCP	medium
Dial-Out	star	star	refer	normal invite	RTCP	medium
Decentral	star	fullmesh	refer + server messaging	normal invite and server msg.	RTCP	medium

9 Security Considerations

The use of a server that performs the mixing on behalf of other users, which is the case for all but one of the conference models described here, introduces security risks. That entity must be trusted by the others to properly mix the media - not omitting a stream, for example. As such, it is recommended that participants in a conference authenticate the identity of the server. In the dial-in, dial-out, and decentralized conferences, this will require authentication of responses by participants.

Mixing also eliminates the privacy possible with end-to-end media transport with mixing in the receivers. Such privacy is still possible in the large-scale multicast conferences, but requires shared keying material for the conference. Doing this for highly dynamic groups is still an open research problem.

10 Conclusion

In this draft, we have shown how to use baseline SIP (assuming endpoints that support the mixing and/or third party call control feature sets) to construct several multiparty conferencing models. These include end system mixing, large-scale multicast conferences, dial-in conference servers, dial-out conferences, ad-hoc centralized conferences, and centralized signaling, distributed media conferences.

11 Acknowledgements

We would like to thank Mary Barnes for her comments and input.

12 Changes since -00

- Added call flow examples.
- Added open issues within text.
- Added additional call flow for adding users to conference, by sending REFER to conference server with Refer-To of new participant.
- Discussed conference servers generating RTCP based on authenticated SIP identities.

13 Authors Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003

email: schulzrinne@cs.columbia.edu

14 Bibliography

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

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

[3] M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments 2327, Internet Engineering Task Force, Apr. 1998.

[4] M. Handley, C. Perkins, and E. Whelan, "Session announcement protocol," Request for Comments 2974, Internet Engineering Task Force, Oct. 2000.

[5] D. Thaler, M. Handley, and D. Estrin, "The internet multicast address allocation architecture," Request for Comments 2908, Internet Engineering Task Force, Sept. 2000.

[6] W. Fenner, "Internet group management protocol, version 2," Request for Comments 2236, Internet Engineering Task Force, Nov. 1997.

[7] D. Waitzman, C. Partridge, and S. E. Deering, "Distance vector multicast routing protocol," Request for Comments 1075, Internet Engineering Task Force, Nov. 1988.

[8] J. Rosenberg and H. Schulzrinne, "Timer reconsideration for enhanced RTP scalability," in Proceedings of the Conference on Computer Communications (IEEE Infocom) , (San Francisco, California), March/April 1998.

[9] J. Rosenberg, P. Mataga, and H. Schulzrinne, "An application server component architecture for SIP," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[10] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP authentication: Basic and digest access authentication," Request for Comments 2617, Internet Engineering Task Force, June 1999.

[11] R. Sparks, "SIP call control," Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[12] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," Request for Comments 2608, Internet Engineering Task Force, June 1999.

[13] J. Rosenberg et al. , "SIP extensions for presence," Internet Draft, Internet Engineering Task Force, Apr. 2001. Work in progress.

[14] J. Rosenberg, J. Peterson, H. Schulzrinne, and G. Camarillo, "Third party call control in SIP," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.