Group Messaging in SIP

Abstract

This document introduces various problems and issues relating to the usage of the Session Initiation Protocol (SIP) for group messaging. Several potential extensions are analyzed and proposed. Both one-shot group messaging (paging) and session based group messaging are covered in the document. Group chat between human beings is one application for group messaging, others exists also such as SIP group messaging between robots or game applets.

1 Introduction

The Session Initiation Protocol (SIP) [1], does not currently clearly define how (instant) messaging works in the multi-party case, such as SIP multi-party chat or group gaming messaging. A client may naturally send the same SIP request to several parties with different To and Call-ID headers but this is highly inefficient and does not scale to large group sizes.

Another possibility is to use reliable multicast below SIP but currently there is minimal multicast deployment.

A more effective way to provide group messaging in SIP is to have a messaging server (standard SIP UA) hosting a SIP URL identifying a group (like sip:group15@server.com). However,
there are several open issues such as handling of call-id’s, responses, retransmissions and delivery notifications in large groups.

Currently SIP supports several messaging schemes. One-shot messaging means that SIP MESSAGE is sent without any concept of thread similarly to SIP OPTIONS method. Thus, the call-id is different each time a new MESSAGE is sent.

In session-based messaging SIP INVITE is typically used to create the instant messaging session. Actual media flow is carried usually in SIP MESSAGE methods. The idea is similar to establishing a RTP session, e.g. receiver advertises in SDP what Call-ID should be used when sending instant messages to it. The Call-ID’s are different in each direction.

However, in server hosted group messaging the Call-ID may be defined by the server.

The mechanism described here is targeted especially for MESSAGE request, but it applies to similar SIP requests as well.

### 2 Applications of SIP Group Messaging

SIP group messaging means that one SIP method (MESSAGE or similar request) is sent to a group. The group address is usually hosted by a server.

Example applications utilizing SIP group messaging include:

- SIP group chat: A SIP session is established with chat server using SIP INVITE. The group size is usually very small, since large group sizes are impractical in chat. Thus, scalability is not a big problem but thread and message identifications are useful. Thread means the related SIP requests inside a session. Typical example of a thread in a SIP group chat is that there are couple of topics being discussed each topic being a separate thread inside a common session.

- Important instant message to be sent to the group address like sip:customers@stock-broker.com. The group size may be huge, even millions. Scalability is an essential requirement. Message identification is also important in order to be able to refer to specific message afterwards. Another possible requirement is the fairness for receivers, in that they should receive the message at about the same time. It is also important to know the status of message delivery.

- Multi-party on-line game: Game participants exchange game-specific data using SIP MESSAGE or similar SIP request. Support for ordering information across SIP requests sent by the group server is useful. This allows clients to know the exact order of the requests sent by the server.

### 3 Requirements

This chapter presents some requirements for SIP-based group communications. Some of them are mainly
- Scalability: First of all, the original sender of the message should not have to handle all responses but instead the server should take care of them, making scalability an issue only for the group server. If the group size is large (e.g. thousands or millions) and all responses arrive at the same time then the feedback implosion is imminent. This is mainly implementation issue.

- Fairness: In some applications, clients should receive the message as same time as possible (not so that first client in the group receives it 15 minutes earlier than the last one). This is important feature e.g. in stock market announcements and in some games. However, this is typically a server implementation issue, although this might be hinted in the SIP level also.

- Ordering: It should be possible to define ordering so that receiving client knows exactly what was the original order of the incoming messages if they arrive out-of-order e.g., because of retransmissions. Date header (if rewritten by the server) provides rough 1-second granularity, which is probably not enough for all situations. Possible some kind of global sequence number (increasing by one) needs to be set by the server.

- Message identification: In some applications it is very useful to be able to refer to specific messages.

- Thread identification: In some applications, like chat, it is useful to have the concept of threads and sub-threads. Thread is an separate topic in a group session typically having its own Subject header. Threads can be nested so that sub-thread is a thread inside an existing thread. If the chat group name is long-lasting and/or the group is large and/or there are a lot of messages in the group then the thread-id would be very useful to distinguish the different threads.

4 Overview of Basic Operation

A SIP group is typically created using an INVITE request [4] so that in the Request-URI line (and in To header) there is a new group name containing a Request-URI with a previously-unused group name. One problem is that the group name may already exists in the server. The OPTIONS request can be used to check whether the group already exists and what is the description for the group, if it exists. INVITE and REFER requests are used to invite and join users into the group in SIP. However, the details for group creation, as well as adding, deleting and inviting users are beyond the scope of this document.

In group messaging, the messaging server (SIP UA) hosts a group name. Clients send group messages to the messaging server using the group name in the Request-URI line and in the To header. From name is the SIP URL of the sender or anonymous name if the sender wishes to remain anonymous.

The Call-ID depends on the situation. If the message belongs to an established SIP session, then it has been agreed in the signaling phase. If the message is one-shot message then the
Call-ID is generated (by the sender) specifically for this message.

Before sending messages to the group address, client may want to subscribe to the status message delivery. This is done by subscribing to specific group-message-status event in the server. This subscription applies to all messages sent to the specific group name from the subscriber. This approach does not require any changes to SIP specifications and scales relatively well to large group sizes.

Server receives the message and responds with 200 ok. When the server has received the message addressed to a group, it generates new message to each group member based on its group membership knowledge. The messaging server takes care of the retransmissions and keeps the state.

The responses from clients in the receiving side will eventually arrive to the messaging server, which does not pass those back to the sender since it would cause feedback implosion in the sender’s network interface.

Instead, server has the knowledge of message delivery and may notify client about the delivery status by using optional event notification scheme. Rather than notifying the message sender for each message delivered, the group messaging server can deliver summaries only if the delivery status has changed significantly. This was triggered earlier using SUBSCRIBE.

5 Issues and Requirements to be Considered

5.1 Message Identification

Currently SIP does not offer support for proper message identification. In server hosted group communication there are always two messages (call-legs) involved, one between the sender and the server, and another between the server and the receiving client. They have usually the same To and From headers, but different Request-URI, Call-ID and CSeq headers. Date header is also rewritten. The Subject header usually remains the same.

Thus, there is no unique way to refer to a specific message.

It is possible to refer to a specific message between a server and a specific client via the Call-ID, To, From and CSeq headers (and Date) but other clients do not know these.

Thus, better identification mechanism might be needed in SIP level. The message content may include identification information but it is content-specific feature.

One possibility is to create a new (unique) message-id header which is created similar to email message-id [8] or SIP Call-ID headers:

Message identification is expected to be useful in several environments for example to refer to a specific instant message afterwards ("In instant message XXX you said"..), or to reply to a specific instant message (allows sub-threading, see section 5.2)

5.2 Thread Identification
Currently it has not been agreed how to do threading in SIP messaging. SIP session may be established but inside that session there may be a lot of messages and different conversations going on. Also, in many cases the receivers are not human beings but "robots" or automated applications, which may utilize threads for their own purposes.

Today it is possible to identify the thread to some extent by using the Subject header in messages. For example, established SIP session may have a couple of different threads going-on with different Subject headers. First message in the thread has the "Subject: mytopic" header and next messages in that thread have "Subject: Re: mytopic" like of arrangement. This works quite well in chat applications. User Interface is able to present them accordingly, but there are still problems e.g. if the "Re:" addition is not clearly defined. Also, the length of the Subject header can be very long, so it is quite difficult to use it as a reliable thread identification. In chat applications it is easier since humans can see the Subject line and treat them accordingly.

Broader kind of threading may be assumed if the Call-ID remain the same (so that server uses all the time same Call-ID). This is even more limited way of doing threading.

These approaches do not allow sub-threading, like in network news in which there may be several main threads and below them more specific sub-threads.

One possibility to provide thread identification is to use Message-ID in a References [5] header (like in network news). There is a original message and if somebody responds to that specific message, then References header includes the message-id of the original message. This allows nice sub-threading and it is possible, for example to present them in chat window accordingly.

The main purpose of the References header is to allow messages to be grouped into conversations by the user interface program. Note that conversation between human beings is just one potential usage of the header. This allows conversations within a group to be kept together, and potentially users might shut off entire conversations in the user interface without leaving the group. User interfaces may not make use of this header, but all automatically generated followups should generate the References line for the benefit of systems that do use it.

Thus, Message-id and References headers together can provide threading and sub-threading.

Every message which is sent without a References header starts a new thread. If there are replies (followups) to that message, they have a References header.

5.3 Ordering

Group messaging should enable ordering. This means that clients should know in what order (or time) the messages were processed and sent by the server.

SIP messages carry a Date header, 1-second granularity, which is
set by the original sender. It may take some time until server receives it, process it and sends it forward. During the 1-second period there may be several messages received, processed and sent. Also, sender clocks may not be well synchronized.

Therefore server should add ordering information to the outgoing message.

One possibility is (like in NNTP) to have extra Date header (e.g. Server-Date) in which the timestamp is provided by the server. The original timestamp (put by the sender) is kept as is. The new Date header may have finer granularity than one second, like e.g. in [6]. Easier solution is that only one Date header is used and it will be set by the server. This gives ordering information given that 1-second granularity is enough.

Another option is to have server-defined sequence number, like Group-Seq for each group which defines clearly the order in which messages were sent from the server. This has some advantages over new Date header. First, there may always be several messages sent almost exactly at the same time, and second, new Date header would have almost the same time than the original Date.

Therefore new Group-Seq like of header (increasing by one) would be the most logical solution. It would start from random number and increase by one for each new message in a group. This would also allow receiving clients to notice if some messages were lost (e.g. all retransmissions were lost) but the main advantage would be in automated applications, which now have the order of messages. For example, in some games there is a situation in which "the last action overrides earlier actions". Therefore the order of the messages is important. This would allow similar view of the game for all game members. Otherwise there is a danger that group members have different view of the reality.

5.4 Notification of Message Delivery

Current SIP group messaging schemes assume that only the server responds with 200 ok to the sending client. This means that sender does not know about the status of the message delivery. The server has the information so it is possible to request it from the server by using standard SIP event notification scheme.

This means that client willing to receive status information about its group message delivery must send the SUBSCRIBE method to the server before sending message(s) and to request to be notified about the status.

This requires new Event name (package).

Usage of event name in SUBSCRIBE and NOTIFY:
Event: group-message-status

The content format of NOTIFY is decided separately for each application since there exist different requirements.

For some applications it is enough to have brief summary of message delivery, e.g. free-form ascii text in response payload, like: "17 messages were delivered, 1 (sip:tom@xyz.com) failed".
It may also be simple structured format, like:
messages-tried: 18
delivered: 17
failed: 1

Some applications may require more specific summary reports such as a XML document specifying full status (time of delivery, name of recipient etc) for each recipient.

One potential format to be utilized in this context is the SMTP Delivery Status Notification format (DSN) [7]. It has a lot SMTP specific headers but at least some headers (e.g. Action: delayed) are useful in SIP messaging context also. Special MIME-type message/delivery-status has been reserved for DSN.

Messaging server may store messages and send them afterwards, e.g. when the recipient logs on. The sender may be notified about these kind of events also. This is a server implementation issue. Rate-control should be done in server to avoid sending NOTIFYs too often.

If the group size is large, server probably should not send NOTIFYs every time there is a change (i.e. message is delivered or there was a failure). Instead, server should wait for a while before sending summary notification and after that only occasional notifications are sent. Notifications can be rate-limited or triggered when a threshold of changes has been reached.

5.5 Scalability

The current scheme scales well from sender point of view since the sender does not receive responses from the recipients. However, the server is now the bottleneck. In group chat applications the group size is usually small, but in other applications such as stock market announcements the group size may be extremely large. Therefore there should be scalability support for large groups.

If the group size is large and all recipients send responses approximately at the same time then the server (and network links near the server) may have difficulties with the simultaneous responses. To overcome this, there are several alternatives of which most are implementation specific.

For example, following techniques may be used to improve scalability:

- Several servers are connected using Corba. Each server takes care of subset of the recipients and after that it reports the results (message delivery status) back to the master server.

- In one server case, server may have a transmission bucket which allow 100 recipients at a time. When one transaction completes then new recipient is added to bucket. This way there are always 100 message transactions going-on as long as there are unsent recipients left.

These kind of optimizations are needed only in large groups.

It should be noted that in some situations, such as in games or in announcements there might be a fairness requirement, that
all recipients should receive the message at the same time.

5.6 Fairness of Delivery

This means that every receiver (client) should receive the message as same time as technically possible. In other words, server side should send the messages as same time as possible. Some group server implementations might send the message in order so that first receiver may receive the message minutes (even tens of minutes) earlier than others. In some cases this is not acceptable. Therefore SIP should provide tools to make fair, simultaneous transfer possible.

The feature itself can be implemented in several ways in server but there should probably be some hint in the SIP level also so that application may use some new header together with Require header in order to utilize the simultaneous delivery feature of the group server.

For example, the following case describes the situation in which important message is sent to the group server and simultaneous delivery is required:

MESSAGE sip:customers@server.com SIP/2.0
Require: simultaneous-delivery
Delivery: yes

For very large groups this may be impossible to guarantee so the usefulness of this feature is questionable.

6 Examples

The following example describes the proposed scheme in detail. The group consists Bob, Alice and Tom.

This example may or may not be session based messaging.

First, Bob subscribes (F1) to event group-message-status and gets reply to it (F2). Then (F3) Bob sends a MESSAGE to the server and gets back a 200 OK (F4) response. Then, the server takes the message, generates new messages (F5, F6) by rewriting the Request-URI and creating new Call-D’s.

Then, it gets back all response codes (F7, F8). It does not pass them back to sender but instead keeps the knowledge of the message delivery of each recipient. Since Bob has subscribed to the group-message-status event the server sends the NOTIFY (F9) describing the status of message delivery. The payload format is text.

Now Alice is replying (F10) to Bob’s message. Alice has not subscribed to status delivery. Then, server sends the message to Bob (F11). Rest of the messaging is not shown here since it is similar to the delivery of Bob’s first message.
The message syntax is as follows (some headers are not shown):

F1: Client subscribes to group-message-status
"SUBSCRIBE sip:group15@im.com SIP/2.0"
From: sip:bob@sender.com
To: sip:group15@im.com
Call-id: def@sender.com
Event: group-message-status
Accept: text/plain, message/delivery-status

F2:
200 OK

F3: Bob sends message to server hosted group
"MESSAGE sip:group15@im.com SIP/2.0"
From: sip:bob@sender.com
To: sip:group15@im.com
Call-id: xyz@sender.com
Message-id: xyz@sender.com

F4: Server to UAC
200 OK

F5: Server to Alice
"MESSAGE sip:alice@provider1.com SIP/2.0"
From: sip:bob@sender.com
To: sip:group15@im.com
Call-id: aaa@im.com
Group-Seq: MESSAGE 1
Message-id: xyz@sender.com

F6: Server to Tom
MESSAGE sip:tom@provider2.com SIP/2.0
From: sip:bob@sender.com
To: sip:group15@im.com
Call-id: bbb@im.com
Group-Seq: MESSAGE 1
Message-id: xyz@sender.com

F7: Alice to server
200 OK

F8: Tom to server
200 OK

F9: Notification to Bob
NOTIFY sip:bob@sender.com SIP/2.0
From: sip:server@im.com
To: sip:bob@sender.com
Call-ID: def@sender.com
Event: group-message-status
Content-Type: text/plain

All messages (2/2) were sent successfully!

F10: Alice sends a followup message to Bob’s message.
The message continue the thread started by Bob’s
original message.
MESSAGE sip:group15@im.com SIP/2.0
From: sip:alice@provider1.com
To: sip:group15@im.com
Call-ID: 123@provider1.com
References: xyz@sender.com
Message-ID: abc@provider1.com

F11 Server sends message to Bob.
MESSAGE sip:bob@sender.com SIP/2.0
From: sip:alice@provider1.com
To: sip:group15@im.com
Call-ID: ghi@im.com
References: xyz@sender.com
Message-ID: abc@provider1.com

7 Security

SIP provides numerous security mechanisms which can be utilized
in addition to those made available through the use of payload
formats such as message/cpim. See [1] and [3] for more details.

8 Summary

This draft has discussed several issues relating to the usage of
SIP in group messaging. It has been emphasized that chat between
humans is just one potential application and there are others as well. Therefore there are a lot of different kind of requirements of which many have been discussed in this document.

At least following extensions might be considered:
- new header: Message-ID
- new header: References
- new event package: optional scheme to find out the status of group message delivery.
- new header: Group-Seq (server defined group-specific sequence number)
- thread identification based on Message-id and References (and using Subject header in chat like of applications)

9 Author’s Addresses

Petri Koskelainen
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: petkos@cs.columbia.edu
electronic mail: petri.koskelainen@nokia.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

10 Bibliography


