Internet Engineering Task Force SIP WG
Internet Draft
Jonathan Rosenberg
dynamicsoft
draft-rosenberg-sip-reconstitute-00.txt
July 13, 2001
Expires: February 2002

Reconsituting Call State in SIP User Agents

STATUS OF THIS MEMO

Abstract

In a SIP network, call and session state resides in the user agents at the edge of the network. These user agents can be elements such as gateways, conferencing servers, and media servers whose availability is important for service delivery. In order to achieve fault tolerance for these user agents, this state must somehow be replicated to backup servers. Traditionally, replication is done through direct memory copies between a primary and its backup. However, the soft-state nature of SIP re-INVITEs means that an alternate mechanism for call state replication is possible. This document proposes mechanisms for reconstituting call state in a UA through triggered re-INVITEs from a peer.

1 Introduction

In SIP [1] networks, call state resides in the user agents which sit at the "edge" of the network. SIP proxies do not maintain call state. As a result of this, a failure of a SIP proxy mid-call has no effect on in-progress calls. The result is high availability of SIP-based proxy networks.

SIP service providers do not just deploy proxies, however. They often need to deploy user agents as well. These include gateways, softswitches, conferencing servers, dialog servers, and application servers, amongst others. Many times, these user agents are also termination points for media. Unfortunately, failures of these types of user agents are more complex. Because the user agent contains call and session state, a backup cannot be used without some kind of transfer of that state. Traditionally, this state transfer is done through dedicated, direct connections between primaries and backups. As state changes in the primary, it is immediately transferred to the backup. This approach is expensive to implement, and provides only a single backup.

Fortunately, the soft-state nature of SIP INVITE requests provides an alternative in some cases. SIP re-INVITE requests are "soft-state" because they contain complete information about the call and the session, rather than conveying deltas from the previous re-INVITE. This means that a UA can potentially reconstitute call and session state upon receipt of a re-INVITE. This concept is discussed briefly in the SIP bis draft [2] to facilitate "crash and restart" of user agents. The mechanism is far more useful for supporting transfer of state to backups, however. In either case, the specification does not provide sufficient guidance on how to provide this capability. This draft addresses that deficiency.

Our approach is to allow for a UA to rapidly detect failure of its peer UA, and then send a re-INVITE to reconstitute state in a backup of its peer. DNS SRV procedures [3] can cause this re-INVITE to be routed to the backup, instead of the failed primary. The re-INVITE reconstitutes call state in the backup, allowing service to continue without interruption for the users.

Our approach is backwards compatible. No new headers, methods, bodies, or response codes are defined. The approach requires additional localized processing, beyond rfc2543bis, amongst the servers that wish to back each other up. Unfortunately, the approach does depend on additional processing from user agents on the other side of the call from the servers that wish to reconstitute this state. We discuss these requirements in Section 4.

2 Applicability

The configuration under consideration is straightforward and presented in Figure 1.

```
............................
.                          .  .................................
.                          .  .                               .
.                          .  .                               .
.                          .  .                               .
.                          .  .                               .
.                          .  .                   +--------+  .
.                          .  .                   |        |  .
.                          .  .                   | UA2    |  .
. +--------+    +--------+ .  .  +--------+       |        |  .
. |        |    |        | .  .  |        |       |        |  .
. | UA1    |    |   P1   | .  .  |   P2   |       +--------+  .
. |        |    |        | .  .  |        |                   .
. |        |    |        | .  .  |        |       +--------+  .
. +--------+    +--------+ .  .  +--------+       |        |  .
.                          .  .                   | UA3    |  .
.                          .  .                   |        |  .
.                          .  .                   |        |  .
.                          .  .                   +--------+  .
.                          .  .                               .
.        Domain foo        .  .         Domain bar            .
............................  .................................



                       .
```

Figure 1: Scenario for reconstituting call state

UA1 is in domain foo, and makes a call which terminates at UA2 in domain bar. This call traverses two proxies. At some point, UA2 fails. We would like to have UA3, a backup, take its place and recover the call. We would like this recovery to be nearly instantaneous, without disruption to the user of UA1.

Recovery with SIP call state reconstitution is not always possible. Generally, this will be because there are additional resources, besides call state, which cannot be replicated, reconstituted, or discarded. For example, UA2 may hold physical resources, such as a PSTN circuit, which cannot be replicated to UA3. As a result, recovering to a backup PSTN gateway may not be possible. However, we have observed that for a large class of SIP user agents, reconstitution is feasible and practical:

Instant Conference Servers: The instant conference servers described in the application component architecture [4] are user agents. They mix together all streams received for the same request URI. Since conferencing state is created dynamically when a call arrives, these servers are ideal candidates for availability through state reconstitution. Consider a conference with three participants connected to a single server. The server fails. Each of the three participants detects this, and sends a re-INVITE that arrives at a backup. As all three re-INVITEs have the same request-URI, this recreates a new conference at the backup server, with all three participants mixed together.

Dialog Servers: Dialog servers interpret VoiceXML scripts, and interact with a caller to collect some kind of information. These servers are discussed in the application component architecture [4]. From a SIP perspective, they are user agents. The state of a dialog server includes the call state, the identity of the current VoiceXML script, and state associated with the VoiceXML interpreter. This latter state is a collection of variables and a pointer to the current point in the execution of the VoiceXML script. When the re-INVITE arrives at a backup, the call state is easily reconstituted. We have proposed that the VoiceXML script to interpret be specified as a URL in the request URI of an INVITE. Therefore, when the re-INVITE arrives, the VoiceXML script being executed can be determined. Unfortunately, unless the VoiceXML interpreter context is stored elsewhere, this state is lost. However, voice processing can restart from the top of the script. From the user's perspective, the failure manifests itself as a glitch. For example, they might hear, "Please enter your credit car..... Please enter your credit card now.". We believe this will frequently be acceptable.

Its also important to note that a complex voice browsing application is usually series of VoiceXML scripts. Data collected from scripts prior to failure will have already been posted to the web server, where web Application

Servers (like Weblogic and Websphere) have mechanisms to reliably store and manage it. As a result, so long as the identity of the current VoiceXML script can be recovered, the voice browsing application can continue without loss of critical data.

Text-to-speech Converters: These servers provide continuous text-to-speech converstion between an audio stream and a text RTP stream [4]. Besides call state, the only other state is the language of the stream being converted. Fortunately, this information is provided within the request-URI, and can be recovered at a backup server. As a result, complete recovery is possible for text-to-speech converters.

Single-User UA: There are a class of single-user end devices where a failure can be recovered by rebooting or rapidly restarting the application. For example, one can imagine a wireless PDA that has instant-on capabilities. If, in the middle of a call, the VoIP application crashes, the OS can detect this and immediately restart the application. Or, the user might accidentally hit the power button, or unplug it, or change the batteries. In these cases, the VoIP application can crash and recover within a few seconds. Here, there is no backup, but rather the original UA itself loses its state and needs to recover it. Since the call state is the primary piece of state, recovery through reconstitution is possible.

In all of the cases above, a key requirement is that the failure of UA2 is detected rapidly by UA1. We discuss how this can be accomplished in the next section.

3 Failure Detection

Rapid failure detection by the peer, UA1, is a key requirement for state reconstitution. The SIP session timer [5] provides the ability to detect failures. However, the frequency between refreshes would need to be very small (on the order of hundreds of milliseconds) in order to usefully recover call state. The session timer intervals cannot scale down this low without adversely affecting proxy capacities.

Instead, we propose that failure detection occur end-to-end using the media stream. Ideally, the RTP or RTCP packets sent by UA1 should begin generating ICMP errors (either port unreachable or host unreachable) upon the failure of UA2. Software failures will generally result in port unreachable errors. Hardware failures can

result in host unreachable failures if the host was also running a routing process, advertising reachability to itself using a host route.

TODO: Include router configuration details.

If UA1 is sending media, the result is that failures can be detected within a single RTT. If UA2 is not sending media, it will require roughly 2.5 seconds on average (half the default RTCP interval). This may be too long. Fortunately, recent work in AVT allows for the RTCP bandwidth fraction to increase, resulting in a decrease in this interval [6]. Achieving detection times on the order of 100 ms is easily achievable.

4 Triggered re-INVITE

Once UA1 has detected the failure of UA2, it sends a re-INVITE in order to reconstitute state at the peer. This re-INVITE MUST contain SDP, and SHOULD contain the same SDP that UA1 last provided to UA2. Using the SRV and/or proxy routing mechanisms described in Section 5, this INVITE will arrive at an alternate server, UA3. If UA3 is incapable of reconstituting state, the re-INVITE will result in a 481 (UA3 can determine that this INVITE is for an old call by the presence of the tag in the To field). A 481 response will cause UA1 to send a BYE, terminating the call. Otherwise, UA3 reconstitutes call and session state and then returns a 200 OK. If, for some reason, the media stream continues to generate errors, UA1 SHOULD try a total of three re-INVITEs, and then give up by sending a BYE.

Even if UA1 cannot reconstitute state itself, it must perform this re-INVITE upon ICMP errors, in order to support state reconstitution in its peers. This is the only new standardized behavior that the mechanism described in this document requires. It is our recommendation that this behavior be incorporated into the bis specification directly.

5 Routing the re-INVITE

To be useful, the re-INVITE request to reconstitute state must arrive at a backup, UA3, and not at failed UA2. Fortunately, this is easily accomplished using SRV records. The original INVITE from UA1 to UA2 passes through some number of proxies (potentially zero), and arrives at UA2. When UA2 inserts a Contact header into the 2xx response, this Contact header does not contain an IP address. Rather, its a domain name that has an SRV record. This record has, as its highest priority entry, the IP address of that specific host UA2. It has lower priority entries for backups (UA3 in this case).

Assuming there were no record-routing proxies, when UA1 tries the re-INVITE, it attempts to send the request to the SIP URL in the Contact header of the 2xx response to the initial INVITE. UA1 will apply the SRV procedures of [3] to this URL. The highest priority entry is tried (this is the failed server, UA2). This generates an ICMP error, so UA2 tries the next-highest priority entry, which is one of the backups. This request succeeds.

In the case of a record-routing proxy, say P2, P2 will simply apply the same SRV procedures that UA1 applied in the paragraph above.

6 UA Requirements

In order for a UA to recover its call state and session state, it must perform additional processing beyond what is specified in RFC2543.

Call state is contained in several places:

Remote CSeq: The highest CSeq seen from the peer.

Local CSeq: The highest CSeq sent by the UA.

Call Leg ID: The ID for this call leg, which is the combination of the To, From, and Call-ID in an INVITE.

Route set: The set of Route headers used to forward requests to the peer.

Session state is contained in several places:

Streams, codec, and parameters: The set of streams with the peer (audio, video, text), and for each, the set of codecs and any associated codec parameters.

Local port/IP address: The IP address and port where the media is being received.

Remote port/IP address: The IP address and port where the media is being sent to.

Incoming SN/TS: The most recent RTP timestamp and sequence number for incoming media.

Outgoing SN/TS: The most recent outgoing RTP timestamp and sequence number for media.

Remote SSRC/CNAME: The SSRC and CNAME for the incoming stream.

Local SSRC/CNAME: The SSRC and CNAME used by the host to send media.

In order to properly recover, all of these parameters need to be reset or reconstituted.

The first task for the backup is to detect that an incoming re-INVITE is for a call that is to be recovered, as opposed to a new call or misrouted call. This detection is done by examining the tag in the To field. A request without a tag in the To field is a new call. If the tag is present, but the server has no call state for the call leg, the INVITE may be for a call to be recovered, or it could be a call for a different UA (and thus might be misrouted).

To distinguish these two cases, we propose a specialized algorithm for computation of the tag in the To field. The tag is composed of a concatenation of two values, separated by a period. The first value is a globally unique ID, across space and time. The second value is a server group ID. This ID is the same for all instances of servers that can potentially act as backups for each other. Generally, the server group ID would be configured by the administrator.

By using a concatenation of these two values, we retain two properties. First, the tag is always globally unique across space and time. This is an important property for proper operation of forking and certain billing applications. Second, a server in a server group can determine whether or not an incoming call was meant for a server in the server group, or for some other server not in the group. If a new incoming call for server A in group 1 has a tag in the To field, and the server group ID in the tag indicates group 1, the call and session state are to be reconstituted.

Once its determined that the INVITE is for a call to be reconstituted, the call state is recovered in the following manner:

Remote CSeq: The remote CSeq is set to the CSeq of the incoming INVITE.

Local CSeq: In order for the peer to accept requests from UA3, the local CSeq must be larger than the local CSeq used by UA2. In order to accomplish this, the servers in a server group need to have synchronized clocks, within 100 ms granularity. Each server computes the initial CSeq for a call by taking the current time, expressed as a 32 bit value representing the number of 100 ms intervals since a configured recent epoch (for example, January 1, 2000). The uppermost 31 bits of this clock are taken, and then shifted right by 1. The result is the initial CSeq. This value is

guaranteed not to wrap for a very long time. It also has the desired property that when reconstituted at UA3, it is larger than the value used at UA2.

Call Leg ID: The Call Leg ID is copied from the incoming INVITE.

Route set: In order to recover the route set, the re-INVITE needs to convey the same Record-Route headers present in the initial INVITE, along with a Contact. Unfortunately, rfc2543 did not mandate that re-INVITEs were record-routed by proxies, nor does it mandate Contact in INVITE. However, rfc2543bis does mandate both Contact in INVITE, and record-routing of re-INVITEs. As a result, the route set can be partially reconstructed, depending on the fraction of proxies which are bis compliant.

Recovery of the session state is accomplished in the following manner:

Streams, codec, and parameters: The re-INVITE contains the complete set of streams, codecs, and codec parameters. Therefore, these are reset based on the incoming re-INVITE.

Local port/IP address: The local port and IP address are rechosen The new values are returned in the 200 OK, updating them with the peer.

Remote port/IP address: The remote IP address and port are conveyed in the re-INVITE.

Incoming SN/TS: The incoming SN and TS will be reset by the arrival of the first media packet from the peer.

Outgoing SN/TS: Unfortunately, the outgoing TS and SN cannot be recovered. However, these values are not relevant if UA3 uses a different SSRC and CNAME than UA2. In that case, the peer will see UA3 as a new RTP participant, and establish a new SN and TS context for it. Eventually, UA2 times out as an RTP participant. This does require that UA's properly implement RFC1889 [7] handling of RTP sessions with multiple participants.

Remote SSRC/CNAME: The SSRC and CNAME for the incoming stream are reset by the arrival of the first RTP and RTCP packets.

Local SSRC/CNAME: The local SSRC and CNAME are rechosen by UA3, and will not match those used by UA2. This is beneficial, in fact, as we discuss above.

7 Special Case: Separate Media and Signaling

Because of the separation of media from signaling, it is possible that different devices terminate the media and the signaling. As a result, the device terminating the media might fail, while the device handling the signaling (UA2) remains functional. In such a configuration, a re-INVITE from UA1 must cause UA2 to attempt to recover or refresh the media state. Without this requirement, state reconstitution will not function.

A common case for this separation is third party call control. [8]. Figure 2 shows the scenario. The controller sets up a call between user A and conference server B (messages 1-6). RTP flows between A and B. B fails. This causes RTP and RTCP packets from A to generate ICMP errors back to A. This causes A to send a re-INVITE (7). Even though this re-INVITE does not change the media at all (the same SDP as before, SDP A, is sent), the controller has to forward this re-INVITE to the other party in order to restablish call state. So, it tries to send the re-INVITE. Using SRV procedures, its first attempt to contact server B fails immediately with an ICMP error, so it tries server C. This re-INVITE succeeds, and reconstitutes the call and conference state in server C. Media then flows between A and C.

8 Reconstituting application state

The procedures above allow a UA to reconstitute the call and session state from an incoming re-INVITE. As we pointed out in Section 2, the application may have other state. In some cases, this state can be stored in the peer. This can be accomplished with re-INVITEs.

Consider an application running on UA2 which requires some application state. This state is small, and changes infrequently (for example, the URL of the currently executing voiceXML script). When the state changes, UA2 sends a re-INVITE to UA1. This re-INVITE contains a new Contact header. This Contact header has, encapsulated in the URI, a representation of the state. In this case, the HTTP URL for the VoiceXML script is URL encoded and placed in the user portion of the Contact URI.

This re-INVITE causes UA1 to update its route set, replacing the existing Contact with the new one. If UA2 fails, the re-INVITE from UA1 will arrive with this Contact URI in the request-URI. UA2 can use this to recover application state.

Effectively, UA2 is able to distribute its state to its peers. The state is transferred to the backup by the peer when recovery is about to take place.

```
          |(1) INV no SDP  |                 |                  |
          |<---------------|                 |                  |
          |(2) 200 OK SDP A|                 |                  |
          |--------------->|(3) INV SDP A    |                  |
          |                |---------------->|                  |
          |                |(4) 200 OK SDP B |                  |
          |                |<----------------|                  |
          |                |(5) ACK          |                  |
          |(6) ACK SDP B   |---------------->|                  |
          |<---------------|                 |                  |
          |       RTP      |                 |                  |
          |..................................|                  |
          |  ICMP error    |                 |                  |
          |<.................................Xfailure          |
          |                |                 X                  |
          |(7) INV SDP A   |                 X                  |
          |--------------->|(8) INV SDP A    X                  |
          |                |----------------------------------->|
          |                |(9) 200 OK SDP C X                  |
          |(10) 200 OK SDP C|<----------------------------------|
          |<---------------|(11) ACK         X                  |
          |(12) ACK        |----------------------------------->|
          |--------------->|                 X                  |
          |      RTP       |                 X                  |
          |.....................................................|
          |                |                 X                  |
          |                |                 X                  |

        User          Controller        Conference         Conference
          A                               Server             Server
                                            B                  C
```

Figure 2: Recovery in 3pcc cases

This mechanism is very similar to the State header specification [9], but does not require an extension to SIP.

This approach is not appropriate for storing any kind of application state. Because it imposes a burden on the peer and on the SIP network, and because it occupies space in the SIP message, it can only be used when the state is below a few hundred bytes, and when it needs to be updated only every few seconds at the most.

9 Conclusions

The proposed mechanism allows for highly available, fault tolerant SIP networks to be constructed. Rather than relying on expensive state replication techniques, we distribute call state, session state, and limited amounts of application state amongst peers at the edge of the network. Recovery of state then becomes an end-to-end operation at the application layer. This is nothing more than an expression of the end-to-end principle.

10 Authors Addresses

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

11 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] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session initiation protocol," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[3] H. Schulzrinne and J. Rosenberg, "SIP: Session initiation protocol -- locating SIP servers," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[4] 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.

[5] S. Donovan and J. Rosenberg, "SIP session timer," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[6] S. Casner, "SDP bandwidth modifiers for RTCP bandwidth," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[7] 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.

[8] 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.

[9] B. Marshall et al. , "SIP extensions for supporting distributed call state," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.