A New Forking Mechanism for Session Initiation Protocol (SIP)
draft-worley-sipping-forking-03

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 4, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
Abstract

The rules for SIP proxies are organized so that when a UAC sends an out-of-dialog request, even if the request is forked to a number of UASs, (usually) only one UAS will accept the request, and only the final response from that UAS will be returned to the UAC. This forking mechanism is optimal for an INVITE intended to connect one human user with another human uses, but is poor for requests that have a "one to many" nature, especially PUBLISH and SUBSCRIBE requests, but also including some INVITEs. This document proposes an alternative forking mechanism that better supports "one to many" requests, and that mechanism be the standardized meaning of the (existing but weakly specified) "Request-Disposition: no-cancel, parallel" header.

Table of Contents

1. Introduction ................................................. 3
2. The current forking mechanism ............................... 4
3. Problems with the current forking mechanism ............... 5
   3.1. Determinism and diagnosis ............................ 5
   3.2. PUBLISH ............................................. 5
   3.3. SUBSCRIBE .......................................... 5
   3.4. Intercom ............................................. 6
   3.5. Conferencing ......................................... 6
   3.6. Messaging ............................................. 7
4. A new proxy forking mechanism ............................... 8
5. Using the new forking mechanism for routing diagnosis ..... 9
6. Relationship to URI-list services ............................ 10
7. Upward compatibility relative to RFC 3841 ................. 11
8. Security considerations .................................... 12
9. Acknowledgments ............................................. 13
10. Revision history ............................................. 14
   10.1. Changes from draft-worley-sipping-forking-00 to
draft-worley-siping-forking-01 ............................... 14
   10.2. Changes from draft-worley-sipping-forking-01 to
draft-worley-siping-forking-02 ............................... 14
   10.3. Changes from draft-worley-sipping-forking-02 to
draft-worley-siping-forking-03 ............................... 14
11. Normative references ........................................ 16
   Author’s Address ........................................... 17
1. Introduction

When a UAC sends an out-of-dialog request, it may pass through many proxies, which may fork the request and deliver copies of it to several UASs. The rules for SIP proxies are organized so that (usually) only one UAS will accept the request, and only the final response from that UAS will be returned to the UAC. This process produces results which are optimal for INVITEs, which are intended to connect one human user with another human user. But this process is not optimal in other circumstances, especially SUBSCRIBE and PUBLISH requests, where an automaton UAC wishes to send information to or receive information from a set of automaton UASs.

In particular, these requests would perform better if they were forked using mechanism that differs in these ways:

1. the request is forked immediately (parallelly) to all targets, regardless of the "q" values of the contact URIs,
2. the request is forked to all applicable UASs, even after one UAS has accepted the request, and
3. all final responses from all UASs are returned to the UAC, rather than being consolidated into one final response.

We propose that the above forking process be considered the standardized definition of the header "Request-Disposition: no-cancel, parallel" [ref-prefs], so that a UAC can request that a request be forked using this alternative mechanism.
2. The current forking mechanism

The rules for SIP proxies in RFC 3261 [ref-sip] are organized so that for any out-of-dialog request, (usually) only one UAS accepts the request and only its final response is received by the UAC. (Within-dialog requests are intrinsically routed to a single destination, the remote target.)

In order to accomplish this, various requirements are placed on proxies: If a proxy receives a 2xx response from one UAS to which it has forked the request, it must send no further copies of the request to other UASs. If the request was an INVITE, it must immediately CANCEL all other forked copies of the request. A proxy should (normally) consolidate all final responses it receives from UASs into one "best response" to be forwarded to the UAC.

The principle that the proxy ensures that only one UAS accepts the request is enforced only approximately -- Due to parallel forking, the request may be sent to two UASs simultaneously, and they may both accept it. If the request method is not INVITE, the proxy will choose one of the responses to send to the UAC and discard the other. If the request method is INVITE, the proxy is required to forward both 2xx responses to the UAC, so that it can be aware that two dialogs were established at the UASs. In that case, the UAC usually sends BYEs to terminate any dialogs beyond the one created by the first 2xx response it receives.

This behavior for proxies was specified in RFC 3261 because it is optimal for most INVITEs, which are intended to create a communication path between one human user and another human user. There are problems with this forking mechanism when establishing other kinds of communication sessions.
3. Problems with the current forking mechanism

The currently specified behavior for proxies is less useful in many situations other than an ordinary INVITE.

3.1. Determinism and diagnosis

The current forking mechanism is admirably suited for offloading the work of dealing with the multiple destination UASs to the proxy that is nearest the point of forking. But ipso facto, it hides any failures or unexpected behaviors from the UAC, making it difficult for a human or automaton located at the UAC to diagnose request routing problems without having administrative access to every proxy along the route. (And it is impossible to determine the identity of a proxy that is involved without already having administrative access to the "upstream" proxies.) In addition, because of the "race" between two UASs that are willing to accept a request, the outcome of the request can be non-deterministic, so a series of diagnostic requests may produce inconsistent responses.

3.2. PUBLISH

The PUBLISH [ref-publish] and SUBSCRIBE [ref-subscribe] requests seem admirably suited for an automaton UAC to distribute information to, or retrieve information from, a set of automaton UASs. But the current forking mechanism makes it impossible for the UAC to reliably send one request to all members of a set of UASs that are the forking targets of one URI.

This is evident for PUBLISH requests, which carry the information to be disseminated. If the PUBLISH is intended to send information to a particular UA, the request-URI could specify that UA. But there is currently no way to reliably disseminate information to all of the UAs that are the destinations for a particular AOR; all that is guaranteed is that at least one UAS will receive the information. For most PUBLISH requests, it would be preferable for the request to be distributed to all forking targets.

3.3. SUBSCRIBE

Any method which attempts to gather information from a group of UAs (such as SUBSCRIBE) has similar problems as PUBLISH, but the use of SUBSCRIBE to establish subscriptions at the UASs, which then send NOTIFY requests, makes the situation even messier: Because the method is not INVITE, all 2xx responses are consolidated into one response by the proxies. This often delays the 2xx response from reaching the UAC until after the NOTIFYs sent by the UASs reach the UAC. In practice, the UAC can only determine the set of subscriptions that it
has created by watching for the NOTIFYs that they send.

In the prototypical example of the pick-up by a user on a phone of a call ringing on another phone [ref-pickup], the phone executing the pick-up sends a SUBSCRIBE to the AOR for the extension to be picked up. The proxy forks this SUBSCRIBE to all the phones that have an appearance of the extension, and each UAS phone sends a NOTIFY detailing the early dialogs on that phone. This information enables the executing phone to find the dialog which it must pick up, regardless of the UA on which the dialog is currently ringing.

If all incoming calls to that AOR are parallel-forked, all of these subscriptions will return the same set of dialogs, so how the SUBSCRIBE is forked is not important. (Actually, the different UAs will return homologous sets of early dialogs, with the same Call-Id’s and from-tags, but different to-tags.) But if the different UAs for the AOR are serially forked, the dialog will be ringing on only one UA at a time, and it is important that the SUBSCRIBE that is searching for that dialog be forked simultaneously to all of the UAs for the AOR. The current forking mechanism does not make this possible.

3.4. Intercom

Another situation in which it would be preferable for all potential targets to receive the request is in an "intercom" feature of an office phone system, where an audio stream is distributed to all the phones in the system, to be output through a speaker on each phone. To implement an intercom request as a single INVITE from the UAC, the single INVITE must be forked to all of targets for request-URI, and the UAC must be able to establish dialogs with all of them.

3.5. Conferencing

A more subtle case where connecting to all targets would be useful is the routing of an INVITE generated by a conference system which forks to multiple targets. [ref-conference] Sometimes such an INVITE is intended to contact a single human, but at other times, its request-URI routes to many humans, and the intention is to include all of them in the conference. Including multiple respondents in the conference is straightforward to achieve if the forking process allows dialogs to be established with all the UASs, by simply including all of the dialogs in the conference.

Another example shows there are additional complications in this situation. Consider sending an initial INVITE to the alias sip:sales@example.com, if you want to have a conference with all your sales personnel. This alias in turn resolves to the AORs
sip:alice@example.com, sip:bob@example.com, and sip:colin@example.com. It is natural to fork the INVITE in parallel to all these targets.

Now, if sip:alice@example.com resolves to two contacts (desk phone and cell phone, maybe) with differing q-values, what should happen? Should the INVITE be forked in the normal (probably serial) manner to these contacts? (That probably is the best way to reach Alice.) Or should both contacts receive the INVITE via parallel forking?

Further complications can be introduced by assuming that sip:sales@example.com initially is forked to sip:sales-tokyo@example.com, sip:sales-ciudad-mexico@example.com, and sip:sales-new-york@example.com, each of which then forks to individual salespeople at those offices, and then assuming that each salesperson’s AOR is forwarded through several proxies before reaching the UAs.

It might be possible to mark the INVITE such that the forking of sip:sales@example.com is done in parallel according to the new scheme, and that the forking of each resulting AOR is done in the traditional manner, but it seems to me that such variants are endless, and it would be impossible to specify them all in a systematic way. And in any case, the UAC doesn’t reliably know the downstream forking structure.

3.6. Messaging

Some "messaging" applications would be better served by forking to all available UASs than by the current forking mechanism. The MESSAGE [ref-message] request can be used to send SMS-style short messages, and in many cases, it would be preferable to fork them to all UAs for the AOR, as a UA might accept the request even if its user is not present (and store the message for display to the user later). So we wish to distribute the MESSAGE to all UAs, not being able to predict which one can first deliver the message to the user.

Similarly, a UAC that supports the MSRP messaging protocol [ref-msrp] as a media session type may, like many instant messaging clients, be willing to support simultaneous messaging sessions, and so would prefer to issue INVITEs with a meaning much like an INVITE to a voice conference. Section 3.5
4. A new proxy forking mechanism

Because the current forking mechanism is ideal for single-session INVITEs, it should not be discontinued. But there is a need for a new forking mechanism that can be applied to requests which the UAS wants to establish dialogs with all possible targets. The alternative forking mechanism differs from the current forking mechanism in three ways:

1. A proxy simultaneously sends copies of the request to all targets in the contact set, regardless of whether any targets have already returned final responses, and regardless of any specified "q" values in any contact set.

2. A proxy does not send CANCELs to other legs of a forked INVITE when one leg returns a 2xx response.

3. A proxy returns all final responses to the requester immediately when they are received (much as a stateless proxy would), rather than holding them and consolidating them into one response.

One possibility for activating this mechanism would be to have the proxy choose the forking mechanism based on the method of the request. But this is undesirable for several reasons: (1) It requires every proxy in the signaling path to know the proper treatment of every method that a UAC might send. (2) The example of an INVITE generated by a conference system (Section 3.5) shows that the best processing of requests depends on more factors than just the method. (3) Making proxy behavior dependent on the request method violates separation of concerns, warning that it would tend to interfere with later SIP extensions.

Thus, we need to define a mechanism which a UAC can use mark requests for its desired desired forking mechanism. For upward compatibility, requests which are not marked must be processed as specified in RFC 3261. Requests which are marked for the alternative forking mechanism, but which are forked by a proxy which does not understand the alternative mechanism, will still be processed as in RFC 3261, but that causes no loss of functionality relative to the current situation. The "Request-Disposition: no-cancel parallel" header defined in RFC 3841 [ref-prefs] appears to be the natural way to activate this mechanism.
5. Using the new forking mechanism for routing diagnosis

TCP/IP networking has evolved a rich set of tools for diagnosing network problems, such as ping, traceroute, nslookup, dig, and telnet. Some of these tools depend on features of the TCP/IP protocol suite that were added to specifically support diagnostics, whereas others exploit the protocol suite in ways not imagined when it was designed. Currently, few diagnostic tools exist for SIP networking other than the OPTIONS request for testing end-to-end connectivity. In particular, it is hard for a UA to trace the routing of a SIP request; tracing usually requires administrative access to the proxies involved.

A simple "traceroute" can be made by sending a series of OPTIONS requests with Max-Forwards values starting at one and increasing by one. Each OPTIONS request will be penetrate one step further along the proxy chain before being returned as either a 2xx (OK) or 483 (Too Many Hops) response. (See section 11.2 of [ref-sip].) But each step of the process can return only one response (OPTIONS being a non-INVITE request), thus hiding most information about any forking that may be occurring.

Using the new forking mechanism, the responses to this series of OPTIONS requests would show evidence of the forking that is involved the the request’s path, as the responses from each request would span the breadth of the forking at the specified depth in the routing tree.

However, a 483 response is not guaranteed to contain much information about the SIP agent which generated the response. Nor is it required to contain much information about the request as it appeared at that agent. In particular, knowing the request-URI from the request at that point would be invaluable, as would knowing the Via headers, which allow the routing tree to be reconstructed automatically. But if SIP agents implement [ref-hop] and generate 483 responses that return the headers of the failing request, the collection of 483 responses would provide detailed information about how the request had been routed.
6. Relationship to URI-list services

The new forking mechanism can be compared to the "URI-list services" mechanism [ref-uri-services]. In both cases, a server downstream from the UAC sends copies of the request to a set of destinations. In URI-list services, this server is a specialized URI-list server; in this proposal, the server is a SIP proxy operating in a special mode. The biggest difference is that in URI-list services, the list of targets is determined directly by the UAC by providing a URI-list (or a reference to one), which allows the UAC to create the list of targets on an ad-hoc basis; whereas in this proposal, the list of targets must be the set of contacts for a SIP URI that is interpreted by a proxy, which allows the list to be maintained by the proxy independently of any UAC that might use it. (This independence is what creates the diagnosis problem Section 3.1.)
7. Upward compatibility relative to RFC 3841

After writing the 00 version of this Internet-Draft, the author discovered that RFC 3841 [ref-prefs] has defined the Request-Disposition header so that the header "Request-Disposition: no-cancel, parallel" header has nearly the semantics that this draft proposes: Features 1 and 2 of Section 4 are specifically included in [ref-prefs], whereas feature 3 (final responses are forwarded immediately) seems to be included only by implication from feature 2 (proxy does not cancel forks upon receiving a 2xx from another fork). But that RFC is rather sketchy about the exact significance of the Request-Disposition header, so if we standardize the meaning of that header to be the forking mechanism defined here, we must worry about upward compatibility with any previous implementations of "Request-Disposition: no-cancel, parallel" -- which may more closely resemble RFC 3261 forking.

The most visible difference between this forking mechanism and possible previous implementations of RFC 3841 is that a UAC may receive multiple responses to a single request. If it is an INVITE request, RFC 3261 allows that multiple responses might be received, though only if they are all 2xx responses. But multiple responses from a non-INVITE request would also possible if the proxy that forks the request is stateless, or if a response was processed by a different proxy in a replicated set of proxies than processed the request. These situations are probably not quite within the RFC 3261 specification, but seem like they could easily arise in practice, and UACs should already be able to handle them.

In general, the significant problem would be in the UAC processing responses and reporting them to the application that invokes the SIP stack. But in information publishing or gathering operations, response processing is not particularly important. When publishing information, the application will not be monitoring the responses, as it does not manage its connection to any particular destination. Any reported success or failure from any particular destination will not change the behavior of the publisher, as there is little it can do to adjust to the failure of one UAS. In information gathering, the responses to a SUBSCRIBE request are not important either, as the significant information comes from the NOTIFY requests that are sent by the created subscriptions, and each of those is processed separately by the SIP stack already.

So it seems that we can safely assume that any previously implemented UAC that uses "Request-Disposition: no-cancel, parallel" will work acceptably if a proxy uses this new forking mechanism in response to that header.
8. Security considerations

This mechanism presents no security considerations that are known to the author.
9. Acknowledgments

The author would like to thank Scott Lawrence and Michael Procter for their comments and advice.
10. Revision history

10.1. Changes from draft-worley-sipping-forking-00 to draft-worley-siping-forking-01

Added "Session Initiation Protocol (SIP)" to the title.

Updated contact information.

Adopted "Request-Disposition: parallel" as the indicator for the new forking mechanism, and added discussion of upward compatibility relative to previous implementations of RFC 3841.

Expanded on the example of a conference call to sip:sales@example.com to show how complex the situation can become.

Added comparison with URI-list services.

Added Security considerations, Acknowledgments, and Revision history sections. Re-titled References section as Normative references.


Added statement that this I-D intends to create a standardized definition for the "Request-Disposition: parallel" header.

10.2. Changes from draft-worley-sipping-forking-01 to draft-worley-siping-forking-02

Updated contact information.

Changed "Request-Disposition: parallel" to "Request-Disposition: no-cancel, parallel", which includes more of the desired behavior by reference to [ref-prefs].

Updated references to Internet-Drafts.

10.3. Changes from draft-worley-sipping-forking-02 to draft-worley-siping-forking-03
Updated contact information.

Updated references.
11. Normative references


Author’s Address

Dale R. Worley
Nortel Networks Corp.
600 Technology Park Dr.
Billerica, MA  01821
US

Phone: +1 978 288 5505
Email: dworley@nortel.com
URI: http://www.nortel.com