Considerations for the Session Initiation Protocol's non-INVITE Transaction
draft-sparks-sip-noninvite-01

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

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 April 16, 2004.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This draft explores several issues with the Session Initiation Protocol's non-INVITE transaction. It focuses on the use of provisional responses and on problems related to transaction timeouts. It proposes two alternative improvements to the existing situation.
Table of Contents

1. Introduction ........................................... 3
2. Problems under the current specifications ........... 3
   2.1 NITs must complete immediately or risk losing a race .... 3
   2.2 Provisional responses can delay recovery from lost final responses ........................................... 4
   2.3 Delayed responses will temporarily blacklist an element .. 6
   2.4 408 for non-INVITE is not useful ...................... 7
   2.5 Non-INVITE timeouts doom forking proxies ............ 8
   2.6 Mismatched timer values make winning the race harder .... 8
3. Alternative A: Improving the situation with a fixed NIT duration ........................................... 9
   3.1 Improving the situation when responses are only delayed .. 9
   3.1.1 Proposal 1: Make the best use of provisional responses ... 9
   3.1.2 Proposal 2: Remove the useless late-response storm .... 10
   3.1.3 Proposal 3: Improve a UAS's knowledge of how much time it has to respond .................................... 10
   3.2 Improving the situation when an element is not going to respond ........................................... 12
   3.2.1 Proposal 4: Strengthen specification of caching success and failures in RFC 3263 ....................... 12
   3.3 When an application needs more time ..................... 13
   3.3.1 Strawman Proposal 5: Specify try again later behavior ... 13
4. Alternative B: Allowing NITs to pend ...................... 14
   4.1 Proposal 6: Allow the non-INVITE transaction to pend indefinitely ........................................... 14
5. Acknowledgments ........................................... 15
   References ............................................ 15
   Author's Address ....................................... 16
   Intellectual Property and Copyright Statements ........ 17
1. Introduction

This draft explores several issues with the non-INVITE transaction. It proposes two alternative paths towards improving the existing situation. Alternative A works within the existing fixed transaction length. Alternative B allows transactions to pend. We can choose one of these alternatives, or choose to pursue alternative A in the short term, and B with a longer term focus.

Alternative A contains several proposals. These proposals stand on their own and may be accepted or rejected independently. Some of Alternative A's proposals are reused in Alternative B, where they also stand independent of each other.

2. Problems under the current specifications

There are a number of unpleasant edge conditions created by the SIP non-INVITE transaction model's fixed duration. The negative aspects of some of these are exacerbated by the effect provisional responses have on the non-INVITE transaction state machines as currently defined.

2.1 NITs must complete immediately or risk losing a race

The non-INVITE transaction is designed to have a fixed and finite duration (dependent on T1). A consequence of this design is that participants must strive to complete the transaction as quickly as possible. Consider the race condition shown in Figure 1.
Figure 1: NI Race Condition

The UAS in this figure believes it has responded to the request in time, and that the request succeeded. The UAC, on the other hand, believes the request has timed-out, hence failed. No longer having a matching client transaction, the UAC core will ignore what it believes to be a spurious response. As far as the UAC is concerned, it received no response at all to its request. The ultimate result is the UAS and UAC have conflicting views of the outcome of the transaction.

Therefore, a UAS cannot wait until the last possible moment to send a final response within a NIT. It must, instead, send its response so that it will arrive at the UAC before that UAC times out. Unfortunately, the UAS has no way to accurately measure the propagation time of the request or predict the propagation time of the response. The uncertainty it faces is compounded by each proxy that participates in the transaction. Thus, the UAS's only choice is to send its final response as soon as it possibly can and hope for the best.

This result constrains the set of problems that can be solved with a single NIT. Any delay introduced during processing of a request increases the probability of losing the race. If the timing characteristics of that processing are not predictable and controllable, a single NIT is an inappropriate model for handling the request. One viable alternative is to accept the request with a 202 and send the ultimate results in a new request in the reciprocal direction.

In specialized networks, a UAS might have some reliable knowledge of inter-hop latency and could use that knowledge to determine if it has time to delay its final response in order to perform some processing such as a database lookup while mitigating its risk of losing the race in Figure 1. Establishing this knowledge across arbitrary networks (perhaps using resource reservation techniques and deterministic transports) is not currently feasible.

2.2 Provisional responses can delay recovery from lost final responses

The non-INVITE client transaction state machine provides reliability for NITs over unreliable transports (UDP) through retransmission of the request message. Timer E is set to T1 when a request is initially transmitted. As long as the machine remains in the Trying state, each time Timer E fires, it will be reset to twice its previous value (capping at T2) and the request is retransmitted.

If the non-INVITE client transaction state machine sees a provisional
response, it transitions to the Proceeding state, where
retransmission continues, but the algorithm for resetting Timer E
is simply to use T2 instead of doubling at each firing. (Note that Timer
E is not altered during the transition to Proceeding).

Making the transition to the Proceeding state before Timer E is reset
to T2 can cause recovery from a lost final response to take extra
time. Figure 2 shows recovery from a lost final response with and
without a provisional message during this window. Recovery occurs
within 2*T1 in the case without the provisional. With the
provisional, recovery is delayed until T2, which by default is 8*T1.
In practical terms, a provisional response to a NIT in currently
deployed networks can delay transaction completion by up to 3.5
seconds.

Figure 2: Provisionals can harm recovery
No additional delay is introduced if the first provisional response is received after Timer E has reached its maximum reset interval of T2.

2.3 Delayed responses will temporarily blacklist an element

A SIP element's use of SRV is specified in RFC 3263 [2]. That specification discusses how SIP assures high availability by having upstream elements detect failure of downstream elements. It proceeds to define several types of failure detection and instructions for failover. Two of the behaviors it describes are important to this document:

- Within a transaction, transport failure is detected either through an explicit report from the transport layer or through timeout. Note specifically that timeout will indicate transport failure regardless of the transport in use. When transport failure is detected, the request is retried at the next element from the sorted results of the SRV query.

- Between transactions, locations reporting temporary failure (through 503/Retry-After for example) are not used until their requested black-out period expires.

The specification notes the benefit of caching locations that are successfully contacted, but does not discuss how such a cache is maintained. It is unclear whether an element should stop using (temporarily blacklist) a location returned in the SRV query that results in a transport error. If it does, when should such a location be removed from the blacklist?

Without such a blacklist (or equivalent mechanism), the intended availability mechanism fails miserably. Consider traffic between two domains. Proxy pA in domain A needs to forward a sequence of non-INVITE requests to domain B. Through DNS SRV, pA discovers pB1 and pB2, and the ordering rules of [2] and [3] indicate it should use pB1 first. The first request to pB1 times out. Since pA is a proxy and a NIT has a fixed duration, pA has no opportunity to retry the request at pB2. If pA does not remember pB1's failure, the second request (and all subsequent non-INVITE requests until pB1 recovers) are doomed to the same failure. Caching would allow the subsequent requests to be tried at pB2.

Since miserable failure is not acceptable in deployed networks, we should anticipate that elements will, in fact, cache timeout failures between transactions. Then the race in Figure 1 becomes important. If an element fails to respond "soon enough", it has effectively not responded at all, and will be blacklisted at its peer for some period
of time.

(Note that even with caching, the first request timeout results in a timeout failure all the way back to the original submitter. The failover mechanisms in [2] work well to increase the resiliency of a given INVITE transaction, but do nothing for a given non-INVITE transaction.)

2.4 408 for non-INVITE is not useful

Consider the race condition in Figure 1 when the final response is 408 instead of 200. Under the current specification, the race is guaranteed to be lost. Most existing endpoints will emit a 408 for a non-INVITE request 64*T1 after receiving the request if they haven't emitted an earlier final response. Such a 408 is guaranteed to arrive at the next upstream element too late to be useful. In fact, in the presence of proxies, these messages are even harmful. When the 408 arrives, each proxy will have already terminated its associated client transaction due to timeout. So, each proxy must forward the 408 upstream statelessly. This, in turn, is guaranteed to arrive too late. As Figure 3 shows, this can ultimately result in bombarding the original requester with spurious 408s. (Note that the proxy's client transaction state machine never enters the Completed state, so Timer K does not enter into play).

```
<table>
<thead>
<tr>
<th>UAC</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>=====</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>^</td>
<td>`-&gt;==</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>`-&gt;==</td>
<td></td>
<td>`-&gt;==</td>
<td></td>
</tr>
</tbody>
</table>
|     | `->==|       | `
|     | `
|     | v
|     | (timeout) --- === |
|     | | | | |
|     | | | | |
|     | | | | |
|     | | | | |
|     | | | | |
|     | | | | |
|     | | | | |
|     | | | | |
```

(Timeout) --- ===
This response bombardment is not limited to the 408 response, though it only exists when participating client transaction state machines are timing out. Figure 4 generalizes Figure 1 to include multiple hops. Note that even though the UAS responds "in time" to P3, the response is too late for P2, P1 and the UAC.

```
UAC        P1         P2         P3         UAS
|          |          |          |          |
---  ===---.     |          |          |          |
^    |     `-->===---.     |          |          |
|    |          |     `-->===---.     |          |
|    |          |          |     `-->===---.     |
64*T1  |          |          |          |     `-->===---.
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
|    |          |          |          |          |
```

Figure 4: Additional timeout related error

### 2.5 Non-INVITE timeouts doom forking proxies

A single branch with a delayed or missing final response will dominate the processing at proxy that receives no 2xx responses to a forked non-INVITE request. Since this proxy is required to allow all of its client transactions to terminate before choosing a "best response". This forces the proxy's server transaction to lose the race in Figure 1. Any response it ultimately forwards (a 401 for example) will arrive at the upstream elements too late to be used. Thus, if no element among the branches would return a 2xx response, failure of a single element (or its transport) dooms the proxy to failure.

### 2.6 Mismatched timer values make winning the race harder

There are many failure scenarios due to misconfiguration or
misbehavior that the SIP specification does not discuss. One is placing two elements with different configured values for T1 and T2 on the same network. Review of Figure 1 illustrates that the race failure is only made more likely in this misconfigured state (it may appear that shortening T1 at the element behaving as a UAS improves this particular situation, but remember that these elements may trade roles on the next request). Since the protocol provides no mechanism for discovering/negotiating a peer's timer values, exceptional care must be taken when deploying systems with non-defaults to ensure they will _never_ directly communicate with elements with default values.

3. Alternative A: Improving the situation with a fixed NIT duration

3.1 Improving the situation when responses are only delayed

There are two goals to achieve when we constrain the problem to those cases where all elements are ultimately responsive and networks ultimately deliver messages:

- Reduce the probability of losing the race, preferably to the point that it is negligible
- Reduce or eliminate useless messaging

3.1.1 Proposal 1: Make the best use of provisional responses

- Disallow non-100 provisionals to non-INVITE requests
- Disallow 100 Trying to non-INVITE requests before Timer E reaches T2 (for UDP hops)
- Allow 100 Trying after Timer E reaches T2 (for UDP hops)
- Allow 100 Trying for hops over reliable transports

Since Non-INVITE transactions must complete rapidly (Section 2.1), any information beyond "I'm here" (which can be provided by a 100 Trying) can be just as usefully delayed to the final response. Sending non-100 provisionals wastes bandwidth.

As shown in Section 2.2, sending any provisional response inside a NIT before Timer E reaches T2 damages recovery from failure of an unreliable transport.

Without a provisional, a late final response is the same as no response at all and will likely result in blacklisting the late responding element (Section 2.3). If an element is delaying its final
response at all, sending a 100 Trying after Timer E reaches T2 prevents this blacklisting without damaging recovery from unreliable transport failure.

Blacklisting on a late response occurs even over reliable transports. Thus, if an element processing a request received over a reliable transport is delaying its final response at all, sending a 100 Trying well in advance of the timeout will prevent blacklisting. Sending a 100 Trying immediately will not harm the transaction as it would over UDP, but a policy of always sending such a message results in unnecessary traffic. A policy of sending a 100 Trying after the period of time in which Timer E reaches T2 had this been a UDP hop is one reasonable compromise.

3.1.2 Proposal 2: Remove the useless late-response storm

- Disallow 408 to non-INVITE requests
- Absorb late non-INVITE responses at proxies

A 408 to non-INVITE will always arrive too late to be useful (Section 2.4). The client already has full knowledge of the timeout. The only information this message would convey is whether or not the server believed the transaction timed out. However, with the current design of the NIT, a client can't do anything with this knowledge. Thus the 408 simply wasting network resources and contributes to the response bombardment illustrated in Figure 3.

If a proxy were able to identify a response as a useless late non-INVITE response, it could absorb the message and not abuse upstream elements with it. A simple change to the non-INVITE client state machine will allow a proxy to identify these responses. Modify the machine to continue to live after Timer F fires to absorb the useless responses. This is similar to what is already provided by Timer K for absorbing retransmitted responses, but the absorption behavior must exist even for reliable transports. (Perhaps it would be sufficient to move the Timer F transition to the Completed state and always set Timer K regardless of transport). This approach suppresses late final responses, such as the 200 in Figure 4, at the element where it first becomes useless.

3.1.3 Proposal 3: Improve a UAS's knowledge of how much time it has to respond

Consider the race lost in Figure 4. The UAS could win this race if it responded soon enough for its 200 to reach the UAC before the UAC timed out. Unfortunately, there is no way, given the current specifications, for the UAC to know how much time it really has left.
It might make a rough guess at the propagation time due to network transmission by counting Via header field values and assuming each hop took at most \( T_1 \), but it has no idea at all what the propagation delay through each of the proxies was.

The UAS's situation could be dramatically improved if the next upstream element explicitly indicated how much time was left. Each element would assume a network delay for any message of \( T_1 \), and estimate the sum of its own internal propagation delay for both the request and the final response, resulting in the messaging shown in Figure 5 (which for compactness assumes \( T_1=500 \text{ms} \) at each hop). Assume the internal delay introduced by P1, P2, and P3 is 1.5s, 3s, and 0.5s respectively. P1 advertises a timeleft of \( 32 - 1.5 - 2T_1 = 29.5 \). P2 advertises a timeleft of \( 29.5 - 3 - 2T_1 = 25.5 \). P3 advertises \( 25.5 - 0.5 - 2T_1 = 24 \).

![Figure 5: Explicitly indicating timeleft](image)

Note that each element determines how much time was and will be lost to network propagation delay over the first upstream hop in incorporates that into its calculation. The UAS will need to do this as well, so in our example above, it knows that it only has 23 seconds to respond.

The estimate of timeleft can be improved if an element has better knowledge of the real network propagation delay. The element can measure its internal propagation delay for the request, but will have
to estimate the propagation delay for the response.

To improve behavior in the presence of existing elements that will not supply a timeleft indication, an element that receives a non-INVITE request without the indication could behave as if it had received value of

\[ 64*T1 - (2*T1 + IPD)*(n\textunderscore Via-1) \]

where

- \( IPD \) = estimate of internal processing delay of a request and a response (strawman: 1s)
- \( n\textunderscore Via \) = number of Via header field values in the request

3.2 Improving the situation when an element is not going to respond

When we expand the scope of the problem to also deal with element or network failure, we have more goals to achieve:

- Identifying when an element is non-responsive
- Minimizing or eliminating falsely identifying responsive elements as non-responsive
- Avoiding non-responsive elements with future requests

Accepting Proposal 1 will dramatically improve an element's ability to distinguish between failure and delayed response from the next downstream element. With this proposal, some response, either provisional or final, is almost certainly going to be received before the transaction times out. So, an element can more safely assume that no response at all indicates the peer is not available and follow the existing requirements in [1] and [2] for that case.

Accepting Proposal 3 provides a similar, but not as strong, improvement in differentiating delayed responses from failure. Proposals 1 and 3 taken together provide the best improvement. Proposal 3 also addresses the proxy doom problem ([Section 2.5](#)).

As [Section 2.3](#) discusses, behavior once an element is identified as non-responsive is currently underspecified. [2] speaks only non-normatively about caching the addresses of servers that have successfully been communicated with for an unspecified period of time.

3.2.1 Proposal 4: Strengthen specification of caching success and failures in [RFC 3263](#)
o Make the caching recommendation normative for servers successfully reached (SHOULD)

o Add failures due to non-responsiveness to that cache (also SHOULD)

o Recommend a expiration for cache members (strawman: 5 minutes)

This cache could also be used to remember servers that have issued a 503 (with or without a Retry-After).

3.3 When an application needs more time

Application designers are faced with significant challenges when the semantics of processing a request require more time (human intervention for example) than the non-INVITE transaction allows. SIP Events ([4]) deals with this by spreading the semantics of processing a new subscription request across two or more non-INVITE requests - a SUBSCRIBE and subsequent NOTIFYs. For example, if a server receives a request for a subscription that cannot be granted or refused until a human provides input, the SUBSCRIBE request will be accepted with a 202 Accepted. A subsequent NOTIFY will convey whether or not the subscription has been allowed or denied.

An alternate approach is to allow a server to tell a client "I can't do this right now, but try again in a little while".

3.3.1 Strawman Proposal 5: Specify try again later behavior

When a server discovers it needs more time than the current non-INVITE transaction will allow to finish the work needed to process the request, it could return a 302 response with:

o A contact pointing to itself with NO expiration time so that this value cannot be cached.

o A Retry-After header indicating when the client should try the request again

A client receiving this response SHOULD retry the request at the indicated time. A server MUST NOT apply the results of the request until the client successfully retries the request. (This limits the set of problems this tool can be used with to those whose side effects can be undone.) A client can effectively CANCEL a request by not coming back.

There are several issues that would need to be resolved if this approach is pursued:
o [1] forbids emitting a 302 with a contact equal to the Request-URI, so the "contact point to self" above would have to change each time (with respect to URI equality) such that the request still arrived at the same agent (requiring a GRUU).

o Emitting and handling 300-class responses for requests inside a dialog is not well-specified in [1]. It is unlikely that existing implementations would exhibit interoperable behavior if they encountered them.

o Proxies would need to know to not recurse on this kind of 302 response. This might require an explicitly signaled extension, or indicate that a 4xx or 5xx class response is more appropriate.

4. Alternative B: Allowing NITs to pend

The root causes of the problems this document attempts to address are the fixed-length NIT (which causes the race condition of Figure 1) and the extra mechanics for providing reliability over unreliable transports.

4.1 Proposal 6: Allow the non-INVITE transaction to pend indefinitely

We can change the definition of the non-INVITE transaction to allow it to pend indefinitely by removing Timer F. By doing so,

o the race condition goes away

o the 408 response would become meaningful once again

o the late response blacklisting problem disappears

o the 408 bombardment problem disappears

o the proxy doom problem is eliminated

Clients would use CANCEL to pending non-INVITEs to stimulate a final response when they are through waiting, similar to INVITE. Proxies will be spared the doom described in Section 2.5 since they can force branches to complete with CANCEL before sending a final response.

Responsibility for reliability over UDP would remain with the requester. This means that provisional responses will still not squelch request retransmission. A long pending non-INVITE request would be retransmitted once 4 seconds (for the default value of T2) once timer E reaches T2, but only over UDP. This might be mitigated by replacing T2 with another, larger, configurable value for use with
The primary disadvantage of this approach is that it raises the expense for handling non-INVITE transactions at proxies to the same level as INVITE transactions. Proxies will have to maintain state for NITs longer than they currently do. Proxies will need a way to end the transaction. We can give them this by duplicating INVITE behavior: create a timer analogous to Timer C. When it fires, send CANCELS down any outstanding branches and once they complete, send a 408 (assuming no branch returned a better final response) to the requester.

This change is backwards-safe, if not completely backwards compatible:

- Existing client, proposed server: The client's experience is unchanged. It will still abandon the transaction after Timer F fires. The failure scenarios are exactly those we currently have. The server will need to protect itself against never receiving a CANCEL (with an analog to Timer C).

- Proposed client, existing server: The behavior here is an improvement over the existing client-server behavior. The 408 emitted by an existing server would become meaningful to the proposed client. New methods that take advantage of the pending property will be rejected by the existing server with a 501. Existing servers might not be expecting CANCEL to non-INVITEs, but are not compliant to the existing specification if such a CANCEL induces incorrect behavior. We would need to add a constraint, similar to that already on the INVITE transaction, binding clients that receive no response within a short time to abandon the transaction instead of pending indefinitely to account for server failure.

If Alternative B is pursued, Proposals 1 (best use of provisionals) and 4 (3263 caching) from Alternative A should also be considered.

5. Acknowledgments

This document attempts to capture many conversations about non-INVITE issues. Significant contributors include Ben Campbell, Gonzalo Camarillo, Steve Donovan, Rohan Mahy, Dan Petrie, Adam Roach, Jonathan Rosenberg, and Dean Willis.

References


Author's Address

Robert J. Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX  75024

EMail: rsparks@dynamicsoft.com
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.