Network Working Group
Internet-Draft
Expires: September 2, 2006

R. Sparks
Estacado Systems
March 1, 2006

Future work addressing issues with the Session Initiation Protocol's non-INVITE Transaction
draft-sparks-sip-nit-future-02

Status of this Memo

This Internet-Draft will expire on September 2, 2006.

Copyright Notice

Abstract

This draft explores several possible remedies for a set of issues that have been identified with the Session Initiation Protocol's non-INVITE transaction. These proposed solutions are in rough form and have not yet accumulated working group consensus. They range from minor extensions to protocol behavior to fundamentally changing the non-INVITE transaction model.

Table of Contents

1. Introduction

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. These problems are documented in [RFC4321]. In summary, as defined [RFC3261]:

> A non-INVITE transaction must complete immediately or risk losing a race
>
> Losing the race will cause the requester to stop sending traffic to the responder (the responder will be temporarily blacklisted)
>
> Provisional responses can delay recovery from lost final responses
>
> The 408 response is useless for the non-INVITE transaction
>
> As non-INVITE transactions through N proxies time-out, there can be an O(N^2) storm of the useless 408 responses

Solutions to many of these problems have been developed and normative updates to the SIP specification are provided in [RFC4320]. This draft explores a set of solutions to address the remaining problems. The solutions are broken into two alternatives. Alternative A focuses on incremental repair to the existing non-INVITE transaction model. Alternative B proposes changing the model.

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

2.1. Improving the situation when responses are only delayed

2.1.1. Improve a UAS's knowledge of how much time it has to respond

Consider the race lost in [RFC4321]. 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 T1, 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 T1, 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 1 (which for compactness assumes T1=500ms 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 - 2*T1 = 29.5. P2 advertises a timeleft of 29.5 - 3 - 2*T1 = 25.5. P3 advertises 25.5 - 0.5 - 2*T1 = 24

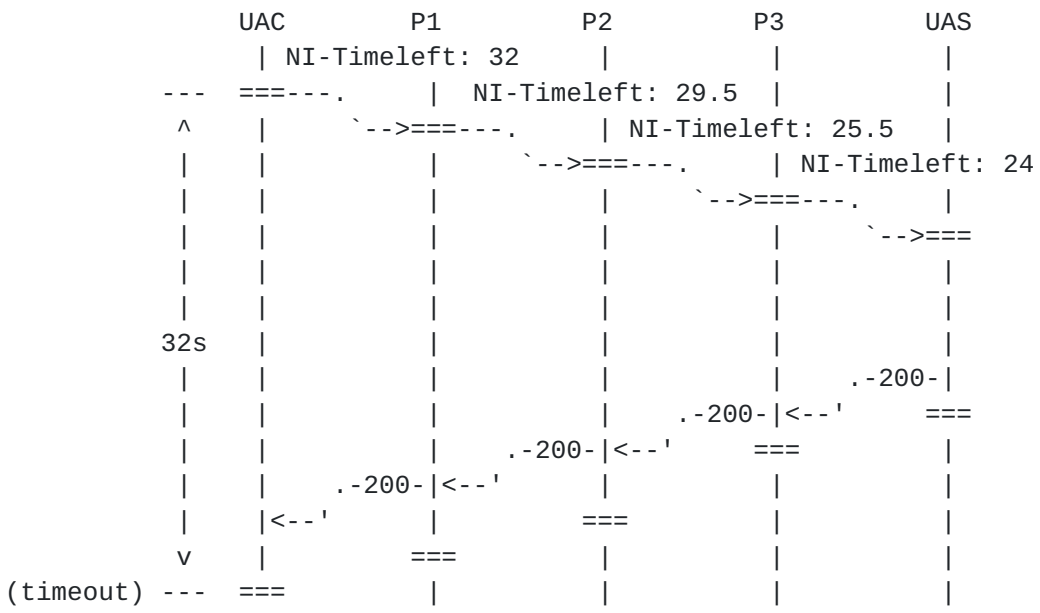

Figure 1: Explicitly indicating timeleft

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_Via-1)
   where
      IPD = estimate of internal processing delay of a
            request and a response (strawman: 1s)
      n_Via = number of Via header field values in the request
```

2.1.2. 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 ([RFC3265]) 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".

2.1.2.1. 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:

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

- 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:

- [RFC3261] 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).

- Emitting and handling 300-class responses for requests inside a dialog is not well-specified in [RFC3261]. It is unlikely that existing implementations would exhibit interoperable behavior if they encountered them.

- 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.

2.2. Improving the situation when an element is not going to respond

2.2.1. Avoiding proxy doom

The mechnanism described in Section 2.1.1 gives a proxy the information it needs to respond in time to avoid the proxy doom problem described in [RFC4321].

2.2.2. Well-specifying blacklisting behavior

As [RFC4321] discusses, behavior once an element is identified as non-responsive is currently underspecified. [RFC3263] speaks only non-normatively about caching the addresses of servers that have successfully been communicated with for an unspecified period of time. If elements do not remember failed attempts to communicate with non-responsive elements between transactions, future non-INVITE transactions will also fail. In those cases where the server-resolution algorithms specified in [RFC3263] yield the same first element to try each time, no non-INVITE transaction towards the URI being resolved will ever succeed, even if there are other servers available for that URI further down the RFC 3263 algorithm.

Early drafts of [RFC4320] proposed solving this problem by specifying success and failure caching. Knowing when to remove an address from these caches was an open issue. Subsequent discussion suggests that we should explore defining an active mechanism for determining when a non-responsive element becomes responsive again.

3. Alternative B: Allowing NITs to pend

The root causes of the problems described in [RFC4320] is the fixed-length non-INVITE transaction and the extra mechanics for providing reliability over unreliable transports.

3.1. Allow the non-INVITE transaction to pend indefinitely

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

- the race condition goes away
- the 408 response would become meaningful once again
- the late response blacklisting problem disappears
- the 408 bombardment problem disappears
- 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 [RFC4321] 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 non-INVITE transaction.

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.

4. Acknowledgments

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

5. References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[RFC4320] Sparks, R., "Actions Addressing Identified Issues with the Session Initiation Protocol's (SIP) Non-INVITE Transaction", RFC 4320, January 2006.

[RFC4321] Sparks, R., "Problems Identified Associated with the Session Initiation Protocol's (SIP) Non-INVITE Transaction", RFC 4321, January 2006.

Author's Address

Robert J. Sparks
Estacado Systems
17210 Campbell Road Suite 250
Dallas, TX 75252

Email: RjS@nostrum.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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