Internet Engineering Task Force MMUSIC WG
Internet Draft
J.Rosenberg,H.Schulzrinne
Bell Laboratories,Columbia U.
draft-ietf-mmusic-sip-100rel-00.txt
November 18, 1998
Expires: May 1999

Reliability of Provisional Responses in SIP

STATUS OF THIS MEMO

1 Abstract

This document specifies an extension to the Session Initiation Protocol (SIP) providing reliable provisional response messages.

2 Introduction

The Session Initiation Protocol (SIP) [1] is a request-response protocol for initiating, maintaining, and terminating multimedia sessions. Each SIP request is followed by one or more provisional responses, followed by a one or more definitive responses. These provisional responses, also called informational responses, have status codes within the 100-199 range. They provide information on call progress, such as trying (100), alerting (180), and queuing (181). However, when run over UDP, SIP does not guarantee that these messages are delivered reliably, or in order. A server simply transmits a provisional response. If the client retransmits the request, the server retransmits the most recent response, provisional or otherwise.

However, a number of applications require reliability and in-order delivery of provisional responses. These include gateway applications, wireless phones, ACD servers, and call queueing systems. Generally, these applications make use of the provisional responses to drive state machinery. This is especially true for the 180 Ringing provisional response, which maps to the Q.931 ALERTING message.

This document provides a simple extension to SIP for ensuring that provisional responses are delivered reliably, independent of the underlying transport mechanism. The extension is simple, requiring two new header fields, and no new methods. It fits well within the generic framework of SIP reliability.

3 Overview

The reliability mechanism is based on the standard windowed acknowledgement technique. When a server generates a provisional response, it places a sequence number (via the RSeq header field) in the provisional response. The sequence number always starts at zero. The sequence number space need only be unique within each Call-ID, To, and CSeq tuple. Because of this, there is no need for randomized sequence number selection or SYN handshakes as in TCP.

The server maintains a window of size 1, which is effectively the value of the highest unacknowledged provisional response that has been transmitted, call this rn. The client maintains a single variable, sn, which represents the highest in order provisional response received so far. Both sn and rn are initialized to -1.

When the server wishes to send a provisional response, it increments rn, places its value in the RSeq header field, and sends the response. The provisional response is retransmitted at intervals with an exponential backoff, starting at T1 (default of 500ms), and doubling after each retransmission. When the client receives the response, it checks the sequence number. If it is one higher than the current value of sn, sn is incremented, otherwise sn is unchanged. It then resends the original request (independently of whether the value of sn has changed), and includes the sequence number sn in the request in the header field RAck.

When the request is received at the server, if the sequence number in the message is equal to the current value of rn, the provisional response is no longer retransmitted. The server is free to increment rn and transmit another provisional response. If the value of the sequence number in the request is one less than the current value of rn, the response is retransmitted, and the server may not generate an additional provisional response.

The mechanism is essentially TCP without congestion control, and with a window of one. The result is a fairly simple mechanism. However, the penalty is that the throughput of provisional responses is fairly low (1 per RTT without loss, lower with loss). However, as the provisional responses are used to signal changes in phone call states, which generally occur on timescales on the order of hundreds of milliseconds to seconds, such a limited throughput appears acceptable. The mechanism can be extended to support larger window sizes, if necessary.

4 Header Fields

Two new header fields are defined, RSeq and RAck. The BNF for these are:

```
RSeq    =    "RSeq" ":" 1*DIGIT
RAck    =    "RAck" ":" 1*DIGIT
```

RSeq is a response header field. It is mandatory when used with this extension. RAck is a request header field. It is mandatory when used with this extension.

The use of reliable provisional responses is signaled by the UAC to the UAS through the Requires header field. This document specifies the named extension org.ietf.sip.reliable-100 requests which require reliable 100's must include this name in the Requires header field and in the Proxy-Require header field, as proxies need to participate.

5 Operation with Proxies

A SIP request may pass through any number of proxies, some of which may fork the request. Furthermore, the SIP specification allows proxies to pass back provisional responses (except for the 100 response) upstream at the discretion of the administrator. If reliability of provisional responses were done end-to-end only, an intermediate proxy which discards provisional responses by default would interfere with the reliability. As such, all intermediate proxies must be aware of the use of the mechanism, and participate.

As a result, reliability of provisional responses is done hop-by-hop, similar to the way non-200-class final responses are handled in normal SIP operation. Stateless proxies can simply forward all provisional responses upstream, ignoring the reliability requirements. A stateful proxy must act as a virtual UAS-UAC in the

algorithm described in the previous section. Once a provisional response has been received reliably at a proxy, the proxy can reliably transmit it upstream towards the next stateful proxy, or may discard it.

Since a proxy may be receiving reliable provisional responses from several branches of a forked request, it will need to merge the provisional response streams together. There are no requirements about the ordering of provisional responses across branches. However, all provisional responses from a given branch must be transmitted reliably upstream in the same order they were received along a branch. For example, consider a forking proxy A which sends a request to UAS's B and C. B sends provisional response 0 towards A, and once it has been received, sends response 1. Similarly, B sends provisional response 2, and once received and acknowledged by A, sends provisional response 3. Proxy A may forward the provisional responses towards the UAS in any one of the following orders:

```
0,1,2,3
0,2,1,3
2,3,0,1
2,0,3,1
0,2,3,1
2,0,1,3
```

Since responses from several branches may be merged at a forking proxy, a proxy may need to renumber the provisional responses (always starting at zero, however) when forwarding them upstream. As this requires changing the RSeq value, the RSeq header field cannot be protected by either end-to-end encryption or authentication. Similarly, a stateful proxy will need to insert the RAck header field itself in all proxied requests.

6 Example

In this example, a UAC wishes to send an INVITE message and receive reliable 100-class responses. Such an INVITE might look like:

```
C->S: INVITE sip:watson@bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:alexander@bell-tel.com
      To: sip:watson@bell-tel.com
```

```
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 1 INVITE
      Subject: Come here Watson
      Require: org.ietf.sip.reliable-100
      Proxy-Require: org.ietf.sip.reliable-100
```

The server wishes to send a 180 Ringing provisional response reliably. The response will look like:

```
S->C: SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP saturn.bell-tel.com
      RSeq: 0
      From: sip:alexander@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 1 INVITE
```

This response is retransmitted with an exponential backoff. When the UAC receives the response, it retransmits the request, but adds the RAck header field:

```
C->S: INVITE sip:watson@bell-tel.com SIP/2.0
      RAck: 0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:alexander@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 1 INVITE
      Subject: Come here Watson
      Require: org.ietf.sip.reliable-100
      Proxy-Require: org.ietf.sip.reliable-100
```

7 Open Issues

There are a number of open issues:

1. It is possible to use a list of sequence numbers in the RAck header field instead of a single number. This would enable a SACK-like mechanism very easily. Is this worth the additional complication?

2. Should we support window sizes greater than one?

3. Currently, SIP requests with the same values of the To, From, Call-ID and CSeq fields are isomorphic. It is possible that certain implementations may discard non-isomorphic requests with identical values for these header fields. By adding the RAck header into a request retransmission, we break the isomorphism of retransmitted requests. Is this a problem?

8 Security Considerations

Since the RSeq value cannot be encrypted or authenticated end-to-end, nor can the RAck, man in the middle attacks are possible which can cause the provisional responses to be reordered at the UAC. This can be alleviated by the use of hop-by-hop encryption and authentication mechanisms, such as IPSEC [2,2].

9 Author's Addresses

Jonathan Rosenberg
Lucent Technologies, Bell Laboratories
101 Crawfords Corner Rd.
Holmdel, NJ 07733
Rm. 4C-526
email: jdrosen@bell-labs.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

10 Bibliography

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Internet Draft, Internet Engineering Task Force, Sept. 1998. Work in progress.

[2] R. Atkinson, "IP encapsulating security payload (ESP)," Request for Comments (Proposed Standard) 1827, Internet Engineering Task Force, Aug. 1995.