Internet Engineering Task Force   MMUSIC WG
Internet Draft   J.Rosenberg,H.Schulzrinne
draft-ietf-mmusic-sip-100rel-01.txt   Bell Laboratories,Columbia U.
May 20, 1999
Expires: November 20, 1999

Reliability of Provisional Responses in SIP

Abstract

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

1 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 are most commonly used for responses to an INVITE request. They provide information on call progress, such as trying (100), alerting (180), and queueing (182). However, when run over UDP, SIP does not guarantee that these messages are delivered reliably, or in order.

However, a number of applications require reliability and in-order delivery of provisional responses to INVITE. 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 to INVITEs are delivered reliably, independent of the underlying transport mechanism. The extension applies only to the INVITE method. Reliability of provisional responses for other methods is not provided. The extension is simple, requiring two new header fields, and no new methods. It fits well within the generic framework of SIP reliability. It is partly backwards compatible, so that a Require header is not needed (it can be included if the UAC insists on the feature, of course), although a Proxy-Require header is needed.

2 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [2] and indicate requirement levels for compliant implementations.

3 Overview

The reliability mechanism is based on the standard windowed acknowledgement technique. When a server generates a provisional response which is to be delivered reliably, it places a sequence number (via the RSeq header field) in the provisional response. These sequence numbers always start at zero, since they are defined only within the context of a transaction. This elimiates the need for SYN handshakes as in TCP. The provisional response is then retransmitted with an exponential backoff.

The UAC maintains a variable, sn, which is the highest sequence number seen in a reliable response. When the client receives a provisional response that has been sent reliably, and this response has a sequence number one higher than sn, sn is incremented, and the request is retransmitted. Otherwise, if the response has a sequence number greater than one higher, sn is not incremented. Either way, the request is retransmitted, and the value of sn is placed in the RAck header in the request.

When the server sees a request retransmission with an RAck header with a value equalling the sequence number in the last reliably transmitted response, it stops retransmitting that response, and is free to send the next provisional response, with a higher sequence number.

The mechanism is similar to TCP, but with a constant window of one. The use of a fixed size window comes at the penalty of reduced response throughput. The througput of 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.

The server can still generate unreliable provisional responses by sending them without an RSeq header. A UAC which receives a provisional response without a RSeq does not retransmit the request. This allows for backwards compatibility; a UAS which doesn't know how to transmit reliable responses will never place an RSeq header in a response, and so the SIP transaction will proceed normally.

Similarly, the initial INVITE from the client contains an RAck header. This serves as an indicator to the server than the client supports the reliability mechanism. A UAS which doesn't see this header in a request knows it cannot provide reliable provisional responses.

4 Detailed Protocol Semantics

A transaction begins when the client sends a request. The client sends the INVITE request as per RFC2543 [1]. The RAck header MUST be placed in the request, with a value of zero, if the client understands and is willing to support this extension for the transaction.

When the initial INVITE is received by the server, it MAY send a 100 response (depending on whether it is a proxy or not). A 100 response is normally sent reliably according to the current SIP specification. This is because the client retransmits its request until a response (i.e., 100) is received, and the server retransmits the 100 response upon request retransmission. As a result, no additional means is needed to reliably send a 100 response over a single hop. Furthermore, the SIP specification mandates that the 100 response is not forwarded through a proxy. For these reasons, 100 responses MUST NOT contain an RSeq header.

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 MUST be initialized to 0.

The server MAY send a reliable response if the initial INVITE request from the client contained a RAck header with a value of 0. If the request contained a Require header, and the server is a UAS, the UAS SHOULD send all non-100 provisional responses reliably. If the request contained a Proxy-Require header, and the server is a proxy, the server SHOULD send all locally generated non-100 provisional responses reliably. It also SHOULD reliably send upstream any responses received reliably from a downstream server. The server MUST NOT send a reliable response if the initial INVITE request did not contain an RAck header with a value of zero. When the server decides to send a provisional response reliably, it MUST increment rn, and MUST place this incremented value in the RSeq header in the response. The provisional response SHOULD be retransmitted at intervals with an exponential backoff, starting at T1 (default of 500ms), and doubling after each retransmission.

When a client receives a provisional response, it checks for the presence of the RSeq header. If it is not present, the response was an unreliable provisional response. The client MUST NOT retransmit the request. As per [1], the client also ceases exponentially backing off request retransmissions when any response (with or without the RSeq header) is received.

> If the server does not understand this extension, it will behave according to the base SIP specification, and retransmit responses upon request retransmissions. A client which retransmits requests upon response retransmissions would cause a feedback loop of constant request and response retransmissions. By checking for the RSeq header, the client can determine whether the server is supporting this extension for this response.

If, however, the provisional response contains an RSeq header, the value is compared against sn. If it is one higher than the current value of sn, sn is incremented, otherwise sn is unchanged. The client SHOULD then resend the original request (independently of whether the value of sn has changed), and MUST include the sequence number sn in the request in the header field RAck.

When a request is received at a server, it checks for the presence of the RAck header. If it is not present, the server retransmits the

last response that was sent. If the RAck header is present, and the value is lower than the value of rn, the last reliable response is retransmitted. If the RAck header was present in the request, and the value is equal to the current value of rn, the exponentially backing off response retransmissions cease. Additional copies of the request with the same or lower value of RAck that are received by the server SHOULD NOT cause the server to retransmit any response (as they would in the above case if RAck were lower), unless rn is zero. The server always retransmits the last response sent (provisional, reliable provisional, or otherwise) when a request is received with both RAck and rn equal to 0.

> This handles the case where a proxy server doesn't send a 100 response, but transmits a reliable response as the first response. To make sure the initial request is transmitted reliably, the server has to retransmit the first response upon request retransmissions.

Once a request has arrived with RAck equal to rn, the server is free to increment rn and transmit another provisional response. The server MUST NOT ever generate an additional reliable response until it has received a request with an RAck header with a value equal to rn.

When the server is ready to send a final response, it does so according to [1]. An ACK request causes retransmissions of the final response to cease. The server SHOULD NOT continue to retransmit any reliable provisional responses once a final response has been sent.

5 Header Syntax

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. RAck is a request header field.

If a client insists that all provisional responses (those generated by proxies and UAS's) be sent reliably, it MUST include both the Require and Proxy-Require headers in all requests. A UAC MAY alternately send requests only with the Proxy-Require header. This will cause all non-100 provisional responses generated by proxies to be sent reliably. Responses sent by UAS's may, or may not be sent

reliably, at the discretion of the UAS.

This document specifies the named extension org.ietf.sip.reliable-100.

6 Operation with Proxies

A SIP request may pass through any number of proxies, some of which may fork the request. The reliability mechanism defined here requires proxies to be aware of the extension. Consider what would happen if a proxy receives a request with a RSeq header, but no Proxy-Require header, and the proxy does not know the extension. As per normal SIP rules, the proxy would forward the request, with the RSeq header in tact, to the downstream proxy. If that proxy did understand the extension, it might try and send a reliable response to the first proxy. The first proxy would see the provisional response retransmissions, but never resend the request. This would cause an excess of network traffic, and block transmission of other provisional responses at the downstream proxy.

The situation would be even more catastrophic for a forking proxy. Consider the case where the first proxy forks the request to downstream proxies A and B. Both A and B understand the extension, and each try to send a reliable response. The first proxy forwards both responses upstream. But, since it does not understand the extension, it does not remove or change the value of the RSeq header in either response. Thus, the client receiving these requests will think they are retransmissions, rather than being two separate responses.

Implementation of this extension in a stateless proxy is not done according to the rules in section 4. A stateless proxy implementing this extension MUST forward all requests it receives downstream, and MUST forward all responses it receives upstream, including provisional responses. Actual reliability is achieved between the first pair of stateful proxies.

A stateful proxy implementing this extension MUST act as a virtual UAS-UAC in the algorithm described in the previous section. When any non-100 provisional response is received reliably at a proxy, the proxy MUST reliably transmit it upstream towards the next stateful proxy. When any non-100 provisional response is received unreliably at the proxy, the proxy MUST send the response unreliably upstream. Any provisional responses generated by the proxy itself (excepting 100) MUST be sent reliably upstream.

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 MUST 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 remove the RAck header from all requests it receives, and insert its own value into proxied requests.

7 Examples

7.1 Message Formatting

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
      RAck: 0
      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: 1
         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: 1
         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
```

7.2 Message Flows

This section illustrates a number of message flows using this extension. We abbreviate an INVITE request with a RAck header value of N as "INV N", and a provisional response with a RSeq header value of M as "1xx M". Packets which are lost are shown with an "X" in front of them.

7.2.1 UAC to UAS, with Require

In this case, the UAC sends a request directly to a UAS, and includes

the Require header, naming this extension. The extension is supported by the UAS. The UAS sends a 100 response first, and then a 180 reliably.

```
              UAC                        UAS

               -------INV 0-------------->
                     X<.......100.........
               -------INV 0--->X
               -------INV 0-------------->
(request       <..........100.............
retransmissions
cease)
                    X<...180 1............ (180 retransmits start, sn=1)

(rn inc to 1)  <.........180 1............
               -------INV 1---->

               <.........180 1............
               -------INV 1--------------> (180 retransmits cease)


                 X<....300............... (300 class retransmits start)
              <........300...............
              -----------ACK------------>
```

7.2.2 UAC to UAS, without Require, UAS doesn't understand

In this case, a UAC sends a request directly to the UAS, and doesn't include the Require header in the request. The UAS doesn't support the extension. The UAS sends a single 180 before sending a final response.

```
              UAC                        UAS

               -------INV 0-------------->
                     X<.......100.........
               -------INV 0--->X
               -------INV 0-------------->
(request       <..........100.............
retransmissions
cease)
```

```
                 <..........180 ...........

                   X<....300.............. (300 class retransmits start)
                <........300..............
                -----------ACK------------>
```

Note that after reception of the 180, the request is not retransmitted, since the response did not contain an RSeq header.

7.2.3 UAC to proxy to UAS

In this case, a UAC sends a request to a proxy, which forwards it to the final UAS. Both the Require and Proxy-Require headers are present in the request. The local proxy generates its own provisional response (188), and the UAS generates a 180:

```
       UAC                     PROXY                    UAS

        -----INV 0-------------> ----INV 0-->X
        -----INV 0-------------> ----INV 0------------->
                                       X<....100........
               X<....100........ <....100..............
        <........100...........

            X<......188 1.......
        <..........188 1.......
        ---------INV 1-->X
        <..........188 1.......
        --------INV 1---------->
                                        X<....180 1.....
                               <......180 1............
                               -------INV 1--->X
              X<....180 2..... <......180 1............
                                -------INV 1------------>
        <..........180 2.....
        -----INV 2------------>
```

Note that the proxy renumbers the two provisional responses before sending them upstream.

8 Open Issues

There are a number of open issues:

1. 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?

2. The mechanism currently requires proxies to understand it to work. It is possible to hack a solution without this constraint, by placing the RAck value as a parameter in the Via header, rather than its own header. The result would be those pairs of proxies which both understand provisional reliability would provide it, those that don't, would not. Is this useful?

9 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 [3,3].

10 Acknowledgements

The authors would like to thank Jonathan Lennox and Adam Roach for the comments on this document.

11 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

12 Bibliography

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments (Proposed Standard) 2543, Internet Engineering Task Force, Mar. 1999.

[2] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments (Best Current Practice) 2119, Internet Engineering Task Force, Mar. 1997.

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