A Solution to the Heterogeneous Error Response Forking Problem (HERFP) in the Session Initiation Protocol (SIP)
draft-mahy-sipping-herfp-fix-00.txt

Abstract

The SIP protocol defines a role for proxy servers which can forward requests to multiple contacts associated with a specific resource or person. While each of these contacts is expected to send a response of some kind, responses for each branch are not necessarily sent back to the original requester. The proxy server forwards only the "best" final response back to the original request. This behavior causes a situation known as the Heterogeneous Error Response Forking Problem (HERFP) in which the original requester has no opportunity to see or
This document describes a backwards compatible solution to the HERFP problem for INVITE transactions.

Table of Contents

1. Conventions ........................................ 3
2. Background ....................................... 3
3. Overview of Solution ............................. 5
4. Proxy Behavior ................................... 6
   4.1 Handling repairable errors .................. 6
   4.2 Receiving subsequent requests with the single-branch property .............. 8
5. User Agent Client Behavior ....................... 9
6. User Agent Server Behavior ....................... 10
7. Security Considerations .......................... 10
8. IANA Considerations ................................ 11
   8.1 The "herf" option-tag ....................... 11
   8.2 The "130 Repairable Error" response-code . 11
9. Acknowledgments ................................... 11
10. References ....................................... 11
    10.1 Normative References ...................... 11
    10.2 Informational References ................... 12
    Author’s Address ................................ 12
A. Historical Context ............................... 12
   A.1 HERFP Problem Description .................. 12
   A.2 The 155 Response ............................. 15
    Intellectual Property and Copyright Statements .......... 20
1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [3].

2. Background

The Session Initiation Protocol (SIP) [1] defines several logical roles, including proxy servers (which forward requests toward their destination), and User Agents (which originate and respond to requests). In addition to transparently forwarding requests, SIP proxy servers can also "fork" requests to multiple User Agent Servers (UAS) if the proxy is authoritative for the domain portion of the Request URI. When forking, proxies forward the same request to multiple contacts which typically have registered as instances of a particular user or service. A proxy can forward requests simultaneously (parallel forking), in series (serial forking), or in combination. As a request is forwarded to a set of contacts, each UAS that receives the request is expected to send a response.

When a proxy forks, it first builds a "target set", a list of User Agent Servers to whom requests will be forwarded. Once forwarding a request, the proxy collects responses from each UAS in a "response context". For INVITE requests, proxies immediately forward all provisional responses and 200-class (success) final responses back to the UAC. For other final responses (regardless of the method of the request), only a single "best" response is sent back to the UAC. The proxy has to delay sending the final response until all branches have completed. This is especially problematic for INVITE transactions, since they can theoretically pend for several minutes, after which most humans have given up attempting communication. In addition, many common SIP error responses are automatically repairable and are used extensively to allow User Agents to negotiate capabilities. These repairable errors are often completely lost if another User Agent finds the request acceptable or returns a "better" error response.

For non-INVITE requests (for example, a SUBSCRIBE request) provisional responses are practically non-existent and only one final response is sent, even if multiple branches returned a 200 response. The SIP events framework (RFC 3265 [6]) effectively deals with HERFP by using NOTIFY requests to convey the success or failure of a SUBSCRIBE request. The single response to a SUBSCRIBE might even arrive after the corresponding NOTIFY request making it effectively redundant. Consequently, this document only addresses HERFP for INVITE transactions. Sending requests other than INVITE and SUBSCRIBE in a manner which causes them to fork is contraindicated.
To illustrate a simple case of HERFP, the UAC below sends a request which includes a body format which is understood by UAS2, but not by UAS1. For example, the UAC might have used a multipart/mixed with a session description and an optional image or sound. UAC1 does not support multipart/mixed, so it returns a 415 response. The UAC can trivially repair this 415 response by resending the request with just the session description. Unfortunately, the proxy has to wait until all branches generate a final response before forwarding the best response. Since the request was acceptable to UAS2, the proxy waits for that branch to finish before it can repair the error. In many cases, the proxy will wait for a long enough amount of time that the human operating the UAC gives up and abandons the call.

```
UAC  Proxy  UAS1  UAS2
 INVITE -->
  INVITE -->
  INVITE -->
  415 -->
  ACK -->
  180 -->
  180 -->
  CANCEL -->
  200 OK -->
  CANCEL -->
  200 OK -->
  487 -->
  415 -->
  ACK -->
```
3. Overview of Solution

HERFP was first described in late 2001. It has remained one of the most challenging problems remaining for the SIP protocol. To effectively address the problem, it is useful to examine the overall goals for a solution to HERFP.

- Convey the semantics of repairable error responses directly to the sender of a (dialog-forming) INVITE request.
- Provide an opportunity for a UAC to retry an INVITE to one branch without canceling other pending branches.
- Do not require modification of the SIP transaction state machine.
- Work through existing RFC 3261 compliant proxy servers.
- Allow the forking proxy to still add or cancel branches.
- Work consistently with unmodified User Agent Servers.

A previous attempt [7] to solve HERFP required each UAS to generate a new provisional response encapsulating the actual final response. However the entire HERFP problem stems from the fact that different UAS implementations will behave differently and frequently implement different sets of extensions. The last goal reflects that a satisfactory solution should work with unmodified User Agent Servers.

Instead of requiring new UAS behavior, this solution enlists the services of the proxy to generate a provisional response of its own (a 130 Repairable Error response) for each branch. Each 130 response encapsulates the repairable final response from one branch. The proxy acts temporarily as a UAS to send these provisional responses. The proxy generates and provides a new URI that the UAC will contact after repairing the error. This URI is similar in spirit to a Globally Unique UA URI (GRUU) [5], except that the URI refers to a specific branch of a specific target set only. Each new URI refers only to one specific failed branch, but is still associated with the list of candidate recipients of the original transaction (the target set).

A UAC which supports this extension reacts to a 130 response by sending a new INVITE request (with the same Call-ID) to the URI in the Contact header of the 130 response. This new request is generated in the same context as the original INVITE request, which is unaffected by the new request. The proxy can still try new branches in the candidate set or cancel old ones. Using this technique, the original requester can immediately fix repairable error responses.

Now consider the same example described above but employing the solution described in this document. The UAC sends a request with a multipart/mixed body. The Proxy forwards this request to UAS1 and UAS2. UAS1 sends the proxy a 415 response. The proxy generates a
URI with the appropriate properties, and generates a 130 Repairable Error response with the 415 response embedded as a message/sip body. The UAC sends a new INVITE to the URI that the proxy generated with only a session description in the body. The proxy forwards the INVITE to UAS1, but manages the forking logic as if the new request was in the original target set. When UAS1 sends a 200 OK, the proxy cancels the branch with UAS2.

<table>
<thead>
<tr>
<th>UAC</th>
<th>Proxy</th>
<th>UAS1</th>
<th>UAS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>--INVITE--&gt;</td>
<td>--INVITE--&gt;</td>
<td>--INVITE--&gt;</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-----180------</td>
</tr>
<tr>
<td>&lt;-------180--</td>
<td>&lt;-----415----</td>
<td>&lt;-----ACK----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;-------130--</td>
<td>&lt;-----180----</td>
<td>&lt;-----180----&gt;</td>
<td></td>
</tr>
<tr>
<td>--INVITE--&gt;</td>
<td>--INVITE--&gt;</td>
<td>&lt;---200----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;-------180--</td>
<td>&lt;-----200----</td>
<td>&lt;---200----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;-200 OK-</td>
<td>----ACK-----&gt;</td>
<td>----CANCEL----&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;--------200 (CANCEL)-</td>
<td>&lt;--------487--&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;---------------487---</td>
<td>----ACK-----&gt;</td>
<td></td>
</tr>
</tbody>
</table>

4. Proxy Behavior

4.1 Handling repairable errors

A proxy which supports this extension performs the following steps when receiving a repairable error:
  o Determine if the UAC supports this extension
  o Determine if the proxy is awaiting pending responses to complete the response context.
  o Generate a URI which identifies this specific branch
  o Encapsulate the original response in a message/sip body

Mahy                     Expires January 9, 2006                [Page 6]
o Generate a 130 Repairable Error provisional response
o Add an Identity header or its equivalent for responses
o Send the 130 response appropriately

To determine if the UAC supports this extension, the proxy needs to check for the presence of the "herf" option-tag in the original INVITE request (typically in a Supported header). If the UAC does not advertise support for this option, response processing continues normally. The proxy also checks the response context of the request. If there are no more branches pending in the response context of this transaction, processing continues normally. The rest of this section assumes that the UAC supports this extension, and that there are pending branches remaining in the response context.

When a proxy receives a 400-class or 500-class response other than a 503, 487, or 408, the proxy SHOULD generate a 130 Repairable Error response as a User Agent Server. If the proxy receives a 300-class response, the proxy can decide based on local policy whether to recurse, or generate a 130 Repairable Error response.

To generate a 130 response, the proxy first creates a message/sip body containing the original (3xx, 4xx, or 5xx) response. The Content-Disposition header for this for this body MUST be "signal" [or we could define a new disposition called "error"]. The proxy does not add the response to the response context for the purpose of returning the best response. The proxy generates a unique To tag for the response. The response context continues to pend until the proxy has positive knowledge that the 130 response was successfully received by the UAC (either the corresponding 130 response is acknowledged or the single-branch URI is contacted).

Next, the proxy generates a "single-branch" URI which corresponds to this branch of this target set. The hostport production of the single-branch URI MUST be identical to the hostport production from the Request URI of the original request. If the Request URI of the original request was a SIPS URI, the single-branch URI MUST be a SIPS URI as well unless the error response was a 416 Unsupported URI Scheme, in which case the proxy SHOULD generate a single-branch URI using the SIP scheme. Otherwise the construction of a single-branch URI is local policy of the proxy and is not subject to standardization.

The proxy SHOULD embed a To header in the single-branch URI that corresponds to the Identity of the branch. Typically, this identity is the same identity which was in the original request. The scheme of the embedded To URI MUST match the scheme of the single-branch URI. The hostport of the embedded To URI MUST be domain for which the proxy can provide the Identity service.
The proxy now generates a 130 Repairable Error provisional response and adds a Contact header field containing the single-branch URI (including the embedded To header), and the message/sip body containing the original response.

The proxy SHOULD add an Identity header or its equivalent used for response identity. This insures the integrity and authenticity of the 130 response and protects from tampering the linkage between the URI provided in the Contact header of the 130 response and the original request.

At this point, the proxy is ready to send the provisional response. If the original INVITE included the 100rel option-tag, the proxy sends the 130 response reliably according to the rules in RFC 3262 [2]. Whether the 130 was sent reliably or unreliably, the proxy MUST retransmit the 130 response every 60 seconds until the proxy has positive knowledge that the 130 response was successfully received by the UAC (either the corresponding 130 response is acknowledged or the single-branch URI is contacted).

Note that provisional responses in SIP can be sent reliably or unreliably. This mechanism can be used in either case. The proxy MUST support the ability to send provisional reliable responses (RFC 3262). Whether the proxy sends 130 Repairable Error responses reliably or unreliably is up to the UAC. If the UAC indicates that it supports reliable provisional responses, the proxy server sends them reliably. Otherwise the proxy sends them unreliably. In most networks the unreliable provisional will arrive and provide the desired behavior. This represents a significant improvement over current behavior. If the unreliable provisional do not arrive, we have not solved HERFP, but the situation is no worse than with existing implementations.

If the proxy responds reliably it MUST include an answer (if the INVITE contained an offer) or an offer (otherwise) in the 130 response. The proxy can satisfy this requirement by generating a minimal offer or answer. A minimally appropriate answer declines all media lines in the offer. A minimally appropriate offer includes no media lines. When a 130 is sent reliably, the message/sip body containing the error and the session description are placed into a multipart/mixed body in the 130 response. UACs which support this extension and provisional reliability MUST support the multipart/mixed MIME type.

4.2 Receiving subsequent requests with the single-branch property

As soon as the proxy is contacted at a single-branch URI for the first time, the proxy tries to find the appropriate branch. If the proxy cannot find the appropriate branch it MUST return a 481
response. If the proxy finds the branch, it marks the original response context for that branch as if the branch returned a 487 response. If the request is a PRACK, the proxy returns a 200 OK response to the PRACK with an appropriate RACK header. If the request is a CANCEL, the proxy returns a 200 OK response to the CANCEL and notes that this branch has been cancelled. If the request is an INVITE, the proxy generates a response context for the new request consisting of one target and forwards the INVITE to the UAS for that target.

The proxy forwards provisional response for the new response context normally. When a final response to the new request is received it is forwarded immediately since the new response context consists of only one branch. If the final response to the new INVITE request is a 200-class or 600-class response, the proxy MUST CANCEL all other pending branches which were created from or related to the original INVITE request. In other words, the proxy must find all pending branches of both the "parent" transaction and all pending "sibling" transactions. In addition, the proxy MUST invalidate all the single-branch URIs associated with the original request.

Note that for a particular branch, the proxy might receive a new INVITE request which repairs one error, but for which there are other unresolved, but repairable error responses. While this situation is currently rare, proxy server MUST NOT invalidate single-branch URIs until Timer C expires for that branch, the branch is cancelled by the UAC, or a 200-class or 600-class response has been received on a parent or sibling transaction.

5. User Agent Client Behavior

A User Agent Client which supports this extension SHOULD advertise for this extension by including the "herf" option-tag in a Supported header field value in dialog-forming INVITE requests. The UAC needs the ability to send multiple invitations in the same user interface context, for example as if the UAC tried multiple contacts from a 300-class response simultaneously.

When a User Agent which supports this extension receives a 130 Repairable Error response to an INVITE request, it performs the following steps.
- Verify the validity of the Identity headers (if present)
- Send a PRACK request if reliability was requested
- Determine if the error is repairable
- Either generate a new INVITE to repair the error, or generate a CANCEL request to acknowledge receipt of the 130 response.

The UAC SHOULD first verify that the 130 response was sent by a host which is authoritative for the domain of the original request and
that the 130 response was not tampered with en route. The UAC checks
that the Identity hash verifies and that the signer of the Identity
header corresponds to the hostport production from the Request URI of
the original request.

If the 130 response was sent reliably, the UAC MUST send a PRACK
request to the URI in the Contact header field of the 130 response.

Next the UAC determines if it can and is willing to repair the error
by examining the message/sip body (which may be a MIME part inside a
multipart/mixed body). UACs which support this extension and
provisional reliability MUST support the multipart/mixed MIME type.
The UAC MAY decide based on local policy not to repair the error or
it may be unable to do so. In that case, the UAC MUST send a CANCEL
request to the URI in the Contact header field of the 130 response.
Note that this CANCEL only cancels a single branch.

If the UAC is willing and able to repair the error, it generates a
new INVITE request using the same Call-ID, but a different from-tag.
It then sends this new INVITE to the URI in the Contact header field
of the 130 response. If an embedded To header is present in the
Contact URI, the UAC MUST override the To header of the new INVITE to
use the value provided in the Contact header.

6. User Agent Server Behavior

This document requires no new behavior by User Agent Servers. It was
designed to work only if the User Agent Client and the Proxy support
this extension. There is an opportunity to improve the current
situation when only the UAC and one UAS cooperate. Such behavior is
potentially complimentary, but out of scope of this document.

7. Security Considerations

An attacker that maliciously injects 130 responses could
theoretically direct a large number of new requests towards a
specific proxy. To prevent this attack, the UAC SHOULD verify that a
130 response has a valid Identity header (or its response equivalent)
signed using a key from a certificate whose subjectAltName is
equivalent to the hostport production from the Request URI, and that
the certificate is rooted in a trusted certificate chain. The
security considerations of a 130 response in this context are
identical to injecting a malicious 300-class response.

A UAS that maliciously injects a 130 could theoretically downgrade
the security of a dialog from SIPS to SIP. The UAC SHOULD include
configurable policy to automatically repair or ignore 416 responses
or to prompt the user.
A UAS that maliciously injects a 130 could selectively disable capabilities or extensions. The security considerations of such an attack are similar to injecting the corresponding 400-class response.

8. IANA Considerations

The following entries should be added to the registries for SIP option-tags and response-codes, respectively.

8.1 The "herf" option-tag

Name of option: herf
Description: Support for safe forking in the face of heterogeneous error responses
SIP headers defined: none
Normative description: This document

8.2 The "130 Repairable Error" response-code

Response Code Number: 130
Default Reason Phrase: Repairable Error

9. Acknowledgments

This idea was the result of 1) participating in discussions with Jonathan Rosenberg, Paul Kyzivat, Jon Peterson, and Cullen Jennings on the properties of URIs in conjunction with the GRUU extension; 2) thoughts I had while implementing best response matching in the repro open-source SIP proxy, 3) numerous discussions about response Identity with Jon Peterson and Cullen Jennings, 4) and a discussion with Mark Eastman about a solution to the Early Attended Transfer problem.

10. References

10.1 Normative References


Appendix A. Historical Context

The Heterogeneous Error Response Forking Problem (HERFP) was described in various SIP working group mailing list threads in late 2001 and then described more formally in a long expired Internet Draft ([draft-rosenberg-sip-unify-00.txt] [8]) in January of 2002. The problem description from the draft is copied here.

A.1 HERFP Problem Description

HERFP, as it is called, is, in our opinion, the most complex remaining problem with the SIP specification.

It relates to the rules for response processing at a forking proxy.
A proxy never forwards more than one error response back to the [User Agent Client (UAC)]. This is needed to prevent response implosion, but more importantly, to support services at proxies. A forking proxy only returns an error response upstream if all forked requests generate an error response. However, a 200 OK [to an INVITE] is always forwarded upstream immediately.

The problem is that if a request forks, and one UAS generates an error because the INVITE is not acceptable for some reason (no credentials, bad , bad body type, unsupported extension, etc.), that response is held at the forking proxy until the other forks respond. Of course, another branch may find the request acceptable, and therefore never generate an error response. The effect is to cancel out the benefits of forking.

```
uac     p1     uas1    uas2
(1) INVITE  |        |
       --------> |
(2) INVITE  |
             --------> |
(3) INVITE  |
             ------------------> |
(4) 401  |
             <-------- |
(5) ACK  |
             --------> |
(6) 180  |
             <---------- |
(7) 180  |
             <------------------ |
(8) CANCEL  |
             <------------------ |
(9) 200 OK  |
             <------------------ |
(10) CANCEL  |
             <------------------ |
(11) 200 OK  |
             <------------------ |
(12) 487  |
             <------------------ |
(13) ACK  |
             <------------------ |
(14) 401  |
             <------------------ |
(15) ACK  |
             <------------------ |
```
Figure 2: Basic HERFP Case

Figure 2 shows the simplest form of the problem. In this flow, the UAC sends an INVITE to proxy P1, which forks to UAS1 and UAS2. UAS1 might be a cell phone, and UAS2 a business phone. UAS1 rejects with a 401, and so never rings. However, UAS2 does not require credentials (or the request already had them), and therefore it rings. However, the user is not at their business phone, although they are available at the cell phone. After ringing for 20s, the caller gives up, and therefore sends CANCEL. This stops UAS2 from ringing, and results in the proxy forwarding the now-old 401 to the UAC. The UAC is not likely to retry, since the user just hung up. Thus, no call is

Another HERFP case is shown in Figure 3. This is a case of sequential forking for a call forwarding service. The UAC calls a user, and the proxy first forks the call to UAS1. The user is not there, so the phone rings for 5s, and is then cancelled by the proxy, which forks to UAS2. UAS2 challenges, resulting in a 401 being returned to the UAC. The UAC tries again, which causes re-invocation of the call forwarding service! UAC1 rings once more for another 5s, and then finally the call is connected to UAS2. Interestingly, if the first UAC doesn’t challenge but the others do, and there are N phones tried before completion, the first phone will ring N times! A user standing by UAS1 but electing to not answer will probably view it as a prank or malicious call.

The problem is that information needs to be propagated back to the UAC immediately, and the UAC needs to resubmit it, but the resubmission should not affect services somehow, e.g., should not re-invoke them as above.

```
+----------------+----------------+----------------+----------------+
<table>
<thead>
<tr>
<th>uac</th>
<th>pl</th>
<th>uas1</th>
<th>uas2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) INVITE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(2) INVITE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(3) 180</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(4) 180</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(5) CANCEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(6) 200 OK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(7) 487</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>(8) ACK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
A.2 The 155 Response

One aspect of our proposal is that a UAS can send a 155 response, instead of a final response, when supported by the UAC, to support services that are complicated by HERFP. The COMET can then be used to provide whatever information is requested by the error response. The COMET would operate just like the a re-INVITE would operate if the actual final response had been sent.
Figure 13: HERFP fix, scenario 1

To show the effectiveness of our proposal, we once again consider the two scenarios of Section 2.6. The call flow for the first case is now shown in Figure 13, except now this time with our proposed solution. For brevity, we ignore the PRACKs for the provisional responses. As before, the caller sends an INVITE, and the proxy forks it. This time, the proxy inserts a Require header in the
request that indicates services are being offered based on dialog state, and so the UAS should send provisionals instead of finals. UAS1 challenges for credentials, but this time, it sends a 155 response that contains the challenge in the WWW-Authenticate header (message 4). The proxy passes this upstream to the UAC. The UAC formulates the response, and places it in an Authorization header in the COMET (message 6). This goes directly to the UAS (the proxy did not record-route). Since the credentials are valid, the UAS proceeds with the session and rings (message 8), which is passed to the UAC. UAS2 does not challenge, and generates an immediate 180, which is passed to the UAC as well. In this example, as discussed in Section 2.6, the user is at UAS1, the call is answered there, resulting in a 200 OK (message 12). The proxy cancels the branch towards UAS2, and the call completes successfully this time!
Consider the second example of Section 2.6. The flow for this example, this time with our proposed solution, is shown in Figure 14. The initial flow proceeds as in Figure 3. UAS1 is rung, and there is no answer, resulting in a cancellation and an attempt to ring UAS2. UAS2 wishes to challenge. However, this time, it issues a 155 that otherwise looks like a 401, which contains a WWW-Authenticate header with the challenge. This response is passed to the proxy and forwarded to the UAC (once again, PRACK requests are not shown). The UAC generates credentials for the challenge, and sends a COMET with the response to the challenge. This is sent directly to UAS2, since the proxy did not record-route. The credentials are accepted,
causing the phone to ring. The user is there, so they pick up, generating a 200 OK, which is passed to the UAC, which sends an ACK to complete the call. Once again, a successful call setup!
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

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