Email System Event Notification Model
draft-dusseault-email-notif-model-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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 November 2, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

Email servers have event information which is of interest to a wide variety of clients, devices and users, and this motivates an effort to tie Email servers into the existing Internet notifications architecture as described by the Common Profile for Presence (CPP). This document describes where Email servers fit into CPP and what pieces are missing. This is not purely a requirements document because it describes a specific architecture, but it makes some requirements on future documents that fill in pieces of the
architecture.

Table of Contents

1. Introduction ........................................... 3
   1.1. Requirements Language .......................... 4
2. Architecture and Terminology .......................... 4
   2.1. Basic Architecture ............................ 4
   2.2. Terminology ..................................... 7
3. Addressing, Location, Capabilities .................... 8
   3.1. Server Addresses and Capabilities .............. 8
   3.2. Resource addressing ............................ 9
   3.3. Hop-by-hop authentication ........................ 9
   3.4. Mail Store Information Access Control ........... 10
   3.5. Confidentiality ................................ 11
   3.6. Reliability ..................................... 11
   3.7. Spam ........................................... 12
4. SIP Functionality ...................................... 12
   4.1. Authenticating SIP clients to PNA ............... 13
5. XMPP Functionality .................................... 13
   5.1. XMPP Addressing ................................ 13
   5.2. Authenticating to PNA ............................ 14
6. SIEVE and Unsolicited Notifications .................... 15
7. Acknowledgements ..................................... 16
8. IANA Considerations .................................. 16
9. Security Considerations ................................ 16
10. References ........................................ 17
    10.1. Normative References .......................... 17
    10.2. Informative References ....................... 17
Author’s Address ........................................ 18
Intellectual Property and Copyright Statements .......... 19
1. Introduction

An Email server can be an IMAP server RFC 3501 [RFC3501] or a POP server RFC 1939 [RFC1939]. It could also be a Webmail server, offering email access through a presentation layer delivered over HTTP RFC 2616 [RFC2616] -- or might support other online interfaces. Although the semantic and storage models differ, these servers are sufficiently similar to be considered together for the purposes of describing interesting events originating from email message stores and how to get access to such event streams. Typically, these servers are capable of storing up to gigabytes of messages per user, handling hundreds of incoming messages per user per day, and allow clients to track which messages have been handled (typically ‘seen’ or ‘read’) and which ones are new.

Work to describe a event model for mail stores has already progressed in the LEMONADE Working Group [I-D.ietf-lemonade-msgevent]. The Introduction of that draft also describes some of the demand for interoperability for producing, consuming and interpreting these events. Meanwhile, the SIEVE Working Group is working on an extension to the SIEVE email filtering language RFC 3028 [RFC3028] that allows a notification to be sent in the case of a SIEVE rule match [I-D.ietf-sieve-notify]. One way to deliver events to existing email clients is to use IMAP for in-band notifications; the Lemonade WG is working on a NOTIFY capability to provide more such functionality. This document instead focuses on out-of-band notifications.

Outside of the email standards groups, two Internet event notification delivery systems have been standardized and deployed. These systems are based on SIP [RFC3261] and XMPP [RFC3920], and each was initially designed and deployed primarily as a PRESENCE SERVICE (section 2.1 of RFC 2778 [RFC2778]). Interoperability between these two systems is defined and described by the Common Profile for Presence (CPP) model RFC 3859 [RFC3859]. Both systems are architected to scale efficiently and to handle cross-domain authentication issues. Salient differences between these two systems will be described in sections [REF-TODO].

Only a few other pieces are required to encourage interoperability between vendors’ products:

- The abstract message event model (a chartered work item of the LEMONADE Working Group -- [I-D.ietf-lemonade-msgevent]).

- A SIP Event Package mapping the event model for use in SIP Event Notifications RFC 3265 [RFC3265].
A mapping of the event model for use in XMPP PubSub [xmpp-pubsub]

A standard way for email servers to deliver large streams of events into the notification system

An explanation of how clients can subscribe to certain more limited streams of events via the notification system, including how to know what server addresses and resource names to subscribe to and how to discover the capabilities of email servers and aggregators.

1.1. Requirements Language

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 [RFC2119].

2. Architecture and Terminology

2.1. Basic Architecture

All the components of the desired email event infrastructure already exist (though not all the interconnections are standardized). This section describes the components. This architecture ignores the existence of SIEVE and SIEVE-generated events until section 6, as these are necessarily unsolicited notification rather than pub-sub notifications.
The Email server or message store, which has plenty of other work to do besides worry about notification delivery, can be configured to send a stream of all events to a local "publisher notifications aggregator" (PNA). This aggregator may support XMPP, SIP or both, and certainly supports publish and subscribe functionality. Some local clients may be able to send subscribe requests directly to the publishing notifications aggregator, but in many cases clients will authenticate to a "consumer notifications aggregator" (CNA) and the CNA will forward the subscription request to the PNA and act as a proxy for notifications.

The email server and PNA are in the same administrative domain. This makes much of the architecture simpler, and is likely to be the most common case anyway. Even if not all use cases can be handled with this limitation, the feasibility of the architecture thus far can be proven in practice before adding the complexity to allow different administrative domains for email server and PNA.

This architecture requires a subscription request before any notifications are sent to the client. The biggest benefit here is that the subscription request is an implicit authorization, along each step of the publish-subscribe chain, for notifications to be sent later. The subscription request also provides potential for clients to customize what information is received.
The Message Event schema [I-D.ietf-lemonade-msgevent] describes events relating to creating and editing messages, receiving messages, deleting, expiring or expunging messages, and creating, deleting and renaming mailboxes.

Consider the following assumptions around the processing and bandwidth demands of events, compared to the existing demands of running a modern mail server:

- For ordinary user mailboxes, we can ignore the overhead of events relating to editing and sending messages originating from the user, and consider only incoming messages (a couple of orders of magnitude more messages typically).

- Quota-related, login/logout and mailbox creation events are sparse compared to incoming message events.

- Notifications never carry a significant proportion of an email body.

- Notifications require no extra processing of the email body beyond what email servers already do. (This doesn’t prevent event information that results from processing the server does already -- e.g. a server might perform spam filtering and only generate events on new non-spam messages). Thus, we consider the length of incoming emails irrelevant to a back-of-the-envelope processing requirements calculation.

- Reading and listing activities generate no event notifications (e.g. clients downloading emails, requesting mailbox listings, searches), except for the case of changing the "read" flag which typically happens once.

- Each new email received by a message store for a given user generates only a handful of events (e.g. new message, message read, message expired, message expunged). That’s because the typical lifecycle of an email is that it arrives, is handled, and archived or deleted once -- most emails do not go through long cycles of flag changes or other state changes.

Thus, if N is the average number of emails that arrive for a given user, the number of events per user is likely to be \( x \times N \), where \( x \) is under 10, or in other words \( O(N) \) (order N). The number of notification messages sent may ultimately be greater because there may be several clients subscribed to a given event, but the estimate of \( O(N) \) still holds for the number of events that the email server must send to the PNA for potential fan-out even when notification services become very popular.
Note that the PNA is responsible for maintaining subscriptions. The mail server sends each event once to the PNA whether there are many subscriptions for that event, just one, or zero. Each event is unaddressed, but naturally must contain enough information for the PNA to match it to the list of subscriptions. Matching events to subscriptions should be trivial if event types are named consistently, and if the resource that generated the event can be matched against either targeted or hierarchical subscriptions (e.g. through path prefix matching). The most difficult requirement here for the PNA will be to determine whether the subscription request ought succeed in the first place or be rejected due to privilege failure.

It should be possible to accommodate multiple notification transports with this architecture, namely both SIP and XMPP. Clients should be able to choose which to implement. Notification servers might have to support both. The email server should implement only one transport which might be neither SIP nor XMPP.

The protocol used by the mail store to send events to the PNA could be a proprietary protocol configured by the administrator of both servers. It could be a new standard TBD. The PNA could act as an IMAP client using the NOTIFY capability if the mail store supports IMAP.

This architecture, together with the assumptions just stated, should ensure that the new work required for email servers to send event streams can be minimal. Essentially, the email server implementation needs to send all supported events (or limit to those types chosen by the server administrators) straight to the PNA. This delegates filtering, fanout and authorization to the PNA, where such functions already exist and can be used or adapted for email notification use cases.

This architecture gives clients choice and the ability to use existing libraries.

Finally, this architecture puts the demands for reliability and scale in the spots where they can be addressed most easily and with existing code -- in the existing CPP server infrastructure.

2.2. Terminology

Email Server - A message store capable of receiving and storing messages for users, along with the ability to distinguish between new messages and handled messages.

Mailbox - A container for Internet messages and/or child mailboxes.
A mailbox may or may not permit delivery of new messages via a mail delivery agent. When the word "mailbox" is used in this document it might mean the top-level folder corresponding to a user’s mail account, or any other within the hierarchy -- unless qualified, it could mean any kind or level of mailbox.

Notifications Aggregator - Any CPP "presence service", plus support for generalized publish and subscribe functionality.

Subscription - In the "PubSub" model, a client does not receive event notifications without an explicit active subscription.

Subscription Request - The message (possibly relayed) in which a client requests a subscription to an event stream.

3. Addressing, Location, Capabilities

This section provides an overview of how clients consuming email event streams can locate the sources of those event streams. As this is a model document this provides only a framework; implementation requirements will be stated in one or more separate documents.

3.1. Server Addresses and Capabilities

We assume that an email event consumer knows the address of the email server that may or may not generate the events it’s interested in. The email event consumer may be an IMAP or POP3 client already configured with the email server address, or it could be a separate device or piece of software. If it’s a separate agent, we assume that the user knows the email server address and can configure that into the client. It’s probably not reasonable to expect users to know the address of the PNA, however, so we need to provide some ways of getting from the email server address to the address a client can subscribe to.

IMAP has a capabilities feature. If the IMAP server is configured to send events to a PNA, this should be advertised with a new capability. It’s even possible to include a server address in a capability advertisement, so the IMAP server could advertise a PNA address at the same time as the capability itself. E.g. the CAPABILITY response could include "NOTIF=sip.example.com".

POP3 feature advertisement will be done via the POP3 extension mechanism in [RFC2449].

If the client does not find the PNA with one of the above approaches, or is not an IMAP or POP3 client, the next step to try should be
querying SRV records. The site hosting an email server with event streams can set up a well-known SRV record type. This could reveal the existence of a SIP server, an XMPP server or ideally both. Sites offering email event streams should support both SIP and XMPP in order to interoperate with a variety of clients.

Further information on addressing can be found in the sections on SIP and XMPP.

3.2. Resource addressing

Once the client has identified which server to make its subscribe request to, the client must specify which mailbox it is interested in. The most common case expected is that the client is interested in new message events for a top-level mailbox (if that’s where most mail arrives) or for the special-case INBOX mailbox, followed by interest in message read counts for any mailbox. Thus, the client has to indicate to the PNA what mailbox the user is interested in -- the PNA can’t assume that it can map the notification clients authentication identifier to an obvious mailbox name. For example, the XMPP user authenticated as ‘xmpp://ldusseault@example.org’ may wish to subscribe to the entire IMAP mailbox for ‘lisadu@example.com’ or to the INBOX mailbox alone or to some other mailbox.

Subscription requests will have to indicate whether the subscription covers only a single mailbox, or also the mailboxes contained in the subscribed mailbox. Alternately, it would simplify the architecture if it were possible to provide just one option for hierarchical handling, but it’s not yet clear which option, if any, meets most common use cases.

In some cases, users will have to provide mailbox names when subscribing. For example, to configure a message waiting indicator status bar applet for a laptop, the user will have to provide not only the mail server address but also the user login for the email server, password and mailbox name unless that is assumed to be the same as the user login.

A complete resource addressing solution is dependent on the addressing model of the notification transport, so this is considered further in the sections on SIP and XMPP.

3.3. Hop-by-hop authentication

The link between the message store and the PNA does not require authentication of either the message store or the PNA. Generally a notifications server will not accept large streams of events from random sources anyway, instead an email provider (ISP or employer or
other) will link together small numbers of known email servers to known notifications servers. Assuming this configuring relationship simplifies the event push solution and eliminates some questions of possible denial of service attacks. Note that even though its not required to authenticate the email server to the PNA or vice versa, a solution which does so may be easy and have useful side effects. For example, the email server could act as a client to the PNA and use existing SIP or XMPP authentication and certificate checking.

The link between the PNA and CNA is authenticated both ways using existing SIP and XMPP mechanisms, and this authentication determines whether the PNA trusts that the CNA is the correct place to send notifications, and whether the CNA allows the PNA to send a stream of events.

The link between the CNA and client is authenticated both ways using existing SIP and XMPP mechanisms. This determines whether the client accepts incoming messages from the CNA and trusts that the CNA has done its own authentication of PNAs further upstream. It also determines whether the CNA is willing to vouch for the client’s chosen authentication identity, in the case where authentication from the client to other servers further upstream is not directly possible.

3.4. Mail Store Information Access Control

In order to get access to event streams from private mail stores, the client consuming the event stream must be able to authenticate to the PNA, either as the mailbox owner or as a notifications node authorized to receive these events. The architecture described here does not necessarily involve querying the email server to authorize a subscription request because the email server will always send all published events to the PNA, thus the PNA must be capable of authorizing subscriptions. (An exception to this statement is if non-standardized mechanisms are used for communicating between PNA and the email server, but for purposes of this framework which doesn’t have such a mechanism, that’s considered to be equivalent to the PNA authorizing the subscription.)

A PNA might be able to query an IMAP server using the IMAP ACL standard [RFC4314] to see what users have read permissions on various mailboxes, and use that to decide to authorize subscriptions. Alternately, the PNA might be configured by the administrator to know (without consulting an email server) which subscriptions to allow. Since the PNA has to be in the same domain as the email server, the choice of how the PNA authorizes subscriptions once it knows the subscriber identity can be left to implementors and administrators.
See the section on SIP for a method of authenticating clients to the PNA.

3.5. Confidentiality

It is likely that end-to-end confidentiality is NOT a requirement for email notifications to be useful. As a comparison, end-to-end confidentiality tends not to be used in instant message systems even where the full contents of messages are involved. In this case, where notifications are not required to have any message content, end-to-end confidentiality is therefore even less important. Hop-by-hop confidentiality will be required, however.

3.6. Reliability

Certainly clients need some reliable information out of event streams, however this is not the same thing as requiring a fully reliably event notification transport. The most common reliability requirement is for a "message waiting" indicator to be accurate in a reasonably timely manner. To state the opposite case, it would be a bad thing if a user saw that there were no messages when in fact a message had been waiting for some time, or if a user saw that there were messages waiting when in fact all the new messages had already been handled. To a lesser extent, the number of new or unread messages must also be reasonably accurate although of course it’s less important to know if there are 299 or 300 unread messages than if there are 0 or 1.

This architecture offers a case where it’s easier to get reasonably reliable knowledge out of an unreliable transport than it is to build a reliable transport. In other words, we should be able to design a way to know whether there are messages waiting even if some events are dropped. Some design options:

- Include total counts in events -- rather than simply deliver "new message" events, deliver "new messages = 299" events.
- Provide heartbeat subscriptions from PNA that repeat message counts -- a client can know if it is not receiving heartbeats that its information may be out-of-date
- Allow out-of-band verification (via IMAP and POP3 for starters)

There are some kinds of notifications that do not require reliability at all. These are hints to the client that reduce the latency of actions that are going to be taken at some later point anyway.

Rather than attempt to discriminate between events that require and
do not require reliability, we assume that fully reliable event
delivery may be addressed later, and in the meantime event delivery
at the level of reliability provided by existing XMPP and SIP systems
will be adequate for many interesting uses.

It should however be noted that many SIP and XMPP systems do indeed
provide high reliability at least across the servers, some offering
"five nines" uptime and no single point of failure. Client libraries
and existing CPP clients may or may not take full advantage of high
reliability depending on need.

3.7. Spam

Spam is mostly out of scope for consideration in notification
architecture. When spam is filtered into particular mailboxes,
mailbox subscriptions can allow the user to choose whether to receive
notifications of spam or not. Although the email server implementor
might be tempted to omit spam messages in message events to PNAs, it
should be noted this would be a very trivial performance improvement
for the email server -- the cost of sending these event notifications
is insignificant compared to the cost of spam detection.

There may be a use case for PNAs to interoperably detect when an
email is thought to be spam (e.g. a standardized indicator in the
event data ) besides relying on what mailbox it’s filed into.

4. SIP Functionality

SIP exchanges messages over sessions between endpoints that have SIP
URIs. It’s common for users to have local proxies where those URIs
are resolved to account information. Thus, the user’s local SIP
proxy acts as the consumer notifications aggregator. Similarly, an
email server can have a SIP address provided by its local SIP proxy
server. In addition, SIP allows for a variable number of additional
SIP proxies that can be ignored for the purpose of defining email
event services over SIP, but may be transparently used in practice.

The work to map email server events to SIP will include specifying a
SIP event package. A SIP event package defines an "Event" header
value which is the package name, and can also define a body. For
email event notifications, clients must be able to specify in one
SUBSCRIBE the email server, mailbox identifier and which email
event(s) are desired. Because the Event header and SIP address are
not intended to carry quite that much information, it might be
necessary to define a SUBSCRIBE body for email events.
4.1. Authenticating SIP clients to PNAs

A feature of SIP that is quite useful for this problem is its support for authenticating a single client to multiple servers in one session. For example, a client attempting to subscribe to a private event stream first authenticates to its proxy server (the CNA) to establish its authorization to be represented by a particular address, then authenticates via the CNA to the PNA to establish its authorization to access information from the email store. For further details, see section 22.3 of RFC 3261 [RFC3261].

Once the PNA has authenticated the user, it still needs to find out whether the user is authorized to get those kinds of events from the mail server. We may need to define how IMAP ACLs can govern access to event streams, and further, how the PNA can know these ACL settings.

5. XMPP Functionality

Although the core XMPP mechanisms are defined in IETF RFCs, important XMPP extensions are also defined through the XMPP Standards Foundation (xmpp.org), including pub-sub features.

5.1. XMPP Addressing

In the XMPP model, clients need to know a JID (Jabber Identifier) in order to address a subscription. A JID consists abstractly of a node name and a domain. JIDs most typically have a user identifier as the node name, so in the case of a user logging in as "juliet" and a domain "example.com", one would typically have a presence and instant messaging address of "juliet@example.com".

Once a JID is known, the client can discover pubsub features and browse various pubsub nodes using the Service Discovery features [xmpp-disco]. This is useful for email events, because it means Service Discovery can be used to discover the exact identifier to use to subscribe to any mailbox, if mailboxes are mapped to nodes.

In order for Service Discovery to allow discovery of mailbox addresses, the PNA must support XMPP and Service Discovery and be able to map mailboxes onto pubsub nodes. This may require either a way for the PNA to query what mailboxes exist on the IMAP server, or a way for it to build a list of mailboxes by seeing them used in notifications. Having the email server support the MailboxCreate and MailboxRename event types would allow the PNA to quickly learn about new mailboxes and allow browsing via XMPP Service Discovery. Once the client browses to discover what resources exist, it naturally
also has the correct addresses for those resources.

For example, the site "example.com" could advertise the "mail_events" node under the the "juliet@example.com" JID (and all other email user JIDs). The subscribe address for events relating to the entire mailbox hierarchy would be "juliet@example.com/mail_events". Underneath that, the address for events relating to the Inbox of juliet’s mailbox might be "juliet@example.com/mail_events/Inbox/". (Note that IMAP mailbox separators characters can vary -- one server might use ‘/’ and another ‘.’ -- but since the mailboxes are mapped to XMPP nodes, the separator for nodes in XMPP addresses is always ‘/’.)

The next step is to identify what types of events the subscriber is interested in. XMPP pubsub supports subscription options. There’s also a form-based mechanism for presenting the user directly with the subscription options, but for a well-defined domain like this one, it’s probably better to bypass the options form and directly specify one of a well-known list of event-types in a well-known option.

In the case that the user’s identifier is not an appropriate base address for email event subscriptions, it should instead be possible for the server to advertise subscriptions at "juliet+mail_events@example.com" or some other opaque node identifier. This is harder to discover, however, so the address based on the user’s account name and supporting Service Discovery is preferred.

5.2. Authenticating to PNA

In the XMPP model, an authorization identity is derived from authentication and asserted by the authenticating server. Thus, juliet@example.com is authenticated by the XMPP server at example.com, and that server asserts the user’s identity for use by other servers. If romeo@example.net has allowed instant messages from juliet@example.com, then the example.net server verifies that the example.com server has a certificate for the domain it asserts names in, and trusts all identifier assertions for that domain.

This works great for asserting JIDs but it doesn’t help us determine whether the JID xmpp://juliet@example.com is authorized to see events relating to the mailbox for juliet.capulet@example.org. For that to work, we need some way of granting email event access permissions to JIDs or a mechanism similar to SIP’s for asking for additional authentication to a remote realm (the email server’s realm). Further work here is TODO.
6. SIEVE and Unsolicited Notifications

Unsolicited notifications to end-user agents can be generated by an email server configured to do so, either by SIEVE or some non-standard mechanism, and be sent via SIP, XMPP, SMS or some other transport.

Unsolicited notifications are more like instant messages than like presence notifications, in that no subscription request travelled the reverse path first. Because there’s no subscription, there’s no guarantee that the recipient will be able to understand any given event schema. Thus, unsolicited notifications are more likely to be consumed by a human user and contain human-readable text. For example, a SIEVE script could be created to send a notification saying "Urgent Message arrived" to xmpp://juliet@example.com on receipt of an email from Romeo, and this would (if permitted delivery) likely appear in Juliet’s instant messaging client.

Strictly speaking unsolicited notifications are no longer part of the CPP model but instead follow the Common Profile for Instant Messaging [RFC3860] model. Some of these notifications would traverse the same path as CPP event notifications: if an unsolicited notification is sent via XMPP or SIP it would still travel from the email server to PNA to CNA to client).
Without subscribe and unsubscribe requests, there are a few drawbacks depending on use case:

- Delivery permission is harder for the receiving client to control
- It's not clear what will happen to messages received while the client is disconnected (e.g., should the CNA store the message? Should it reject the message, and if so, should the PNA stop sending such messages or continue to do so?).
- Interoperability for machine-readable agents is difficult depending on the event schema

The mechanism described for pubsub in section 2 -- where the email server hands events without addresses to the PNA and lets it sort out subscriptions and filters -- won’t work for this type of notification. Instead, the email server would have to send targeted events to the PNA for forwarding to specific addresses.

In order to get permission to send messages to an address over SIP and XMPP, the sender sometimes has to be added to a whitelist. It’s possible for some SIP or XMPP clients to be configured to reject messages from unknown senders. For example, the client could be configured to reject messages from any sender that didn’t already have a presence subscription approved. It’s beyond the scope of this framework how that can be resolved, but we point out that clients are certainly not required to set up permissions to link incoming message blocking to presence subscriptions.

7. Acknowledgements

TODO

8. IANA Considerations

There are no IANA considerations introduced in this document.

9. Security Considerations

This document makes no implementation requirements. However, it does make rash statements about security which need to be checked.

10. References
10.1. Normative References

[I-D.ietf-lemonade-msgevent]


10.2. Informative References

[I-D.daboo-imap-annotatemore]

[I-D.ietf-sieve-notify]


Author’s Address

Lisa Dusseault
CommerceNet
169 University Ave.
Palo Alto, CA 94301
US

Phone: 1-650-279-7365
Email: ldusseault@commerce.net
Full Copyright Statement

Copyright (C) The IETF Trust (2007).

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.

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, THE IETF TRUST 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.

Intellectual Property

The IETF takes no position regarding the validity or scope of any
Intellectual Property Rights or other rights that might be claimed to
certain 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

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.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF
Administrative Support Activity (IASA).