Basic Level of Interoperability for Session Initiation Protocol (SIP) Services (BLISS) Problem Statement
draft-ietf-bliss-problem-statement-02

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 August 27, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

The Session Initiation Protocol (SIP) has been designed as a general purpose protocol for establishing and managing multimedia sessions. It provides many core functions and extensions in support of features such as transferring of calls, parking calls, and so on. However, interoperability of more advanced features between different vendors has been poor. This document describes the reason behind these interoperability problems, and presents a framework for addressing
them.

Table of Contents

1. Introduction .................................................. 3
2. The Confusion of Tongues .................................. 4
3. Concrete Examples ........................................... 5
   3.1. Call Forward No Answer ................................. 5
      3.1.1. Approach 1: UA Redirects .......................... 6
      3.1.2. Approach II: Proxy Forwards ....................... 7
      3.1.3. Approach III: UA Proxies .......................... 8
      3.1.4. Approach IV: Call Processing Language ......... 9
      3.1.5. Failure Cases ...................................... 10
         3.1.5.1. No One Implements ........................... 11
         3.1.5.2. Both Implement ................................ 11
         3.1.5.3. Missing Half .................................. 11
4. Solution Considerations ..................................... 11
5. BLISS Solution Framework .................................. 13
   5.1. Phase I - Identify a Feature Group ................. 13
   5.2. Phase II - Gather Data ................................ 15
   5.3. BLISS Phase III - Problem Definition ............... 15
   5.4. BLISS Phase 4 - Minimum Interop Definition ....... 16
6. Structure of the BLISS Final Deliverable .................. 17
7. Security Considerations ...................................... 18
8. IANA Considerations .......................................... 18
9. Acknowledgements ............................................ 18
10. Informative References ...................................... 18
    Author’s Address ........................................... 19
    Intellectual Property and Copyright Statements .......... 20
1. Introduction

The Session Initiation Protocol (SIP) ([RFC3261] has been designed as a general purpose protocol for establishing and managing multimedia sessions. In this role, it provides many core functions and extensions to support "session management features". In this context, session management features (or just features in this specification) are operations, typically invoked by the user, that provide some form value-added functionality within the context of a multimedia session. Examples of features include putting a call on hold (possibly with music), transferring calls, creating ad-hoc conferences, having calls automatically forwarded, and so on.

The SIP specification itself includes primitives to support some of these features. For example, [RFC 3264] defines SDP signaling parameters for placing a call on hold. Numerous SIP extensions have been developed which focus on functionality needed for session management features. The REFER specification, [RFC 3515], defines a primitive operation for a user agent to ask another user agent to send a SIP request, typically to initiate a session. REFER is used to support many features, such as transfer, park, and hold. The Replaces specification, [RFC 3891], allows one dialog to replace another. This header field is useful for consultation transfer features. The dialog event package, [RFC 4235], allows one UA to learn about the dialog states on another UA. This package is useful for features such as shared line.

However, despite this veritable plethora of specifications that can support session management features, in practice, interoperability has been quite poor for these kinds of functions. When user agents from one vendor are connected to servers and user agents from other vendors, very few of these types of features actually work. In most cases, call hold and basic transfer are broadly interoperable, but more advanced features such as park and resume, music-on-hold, and shared line appearances, do not work.

In some cases, these interoperability failures are the fault of poor implementations. In other cases, they are purposeful failures, meant to ensure that third party equipment is not utilized in a vendor’s solution. However, in many cases the problem is with the specifications. There are two primary specification problems that can cause interoperability failure:

- A feature requires functionality that is not defined in any specification. Therefore, the feature cannot be implemented in an interoperable way.
A feature can be implemented in many different ways, each one using different specifications or different call flows, and assuming different functionality in each participating component of the system. However, each component in a particular deployment each chose a different way, and therefore the overall system lacks interoperability.

This latter problem is the primary focus of this document. Section 2 describes the problem in architectural and more abstract terms. Section 3 then gives several concrete examples that demonstrate the problem. Section 4 then proposes a general framework for resolving the interoperability problem. Finally, Section 6 defines a template that can be utilized by specifications for addressing this interoperability problem.

2. The Confusion of Tongues

SIP is typically deployed in environments a large number of user agents and some number of servers, such as proxy servers, registrars, feature servers, and so on. Put together, these form a distributed system used to realize a multimedia communications network.

Architecturally, a SIP-based multimedia network can be thought of as a distributed state machine. Each node in the network implements a state machine, and messages sent by the protocol serve the purpose of synchronizing the state machines across nodes. If one considers these session management features (hold, transfer, park, etc.), each of them is ultimately trying to achieve a state change in the state machines of two or more nodes in the network. Call hold, for example, attempts to change the state of media transfer between a pair of user agents. More complex features, such as transfer, are an attempt to synchronize dialog and call states across three or more user agents. In all cases, SIP messaging is used between these agents to change the state machinery of the protocol.

If we consider a particular feature, the protocol machinery for accomplishing the feature requires logic on each node involved in the feature. Let us say that feature X can be implemented using two different techniques – X.1 and X.2. Each technique is composed of a series of message exchanges and associated state machine processing in each affected node. If all affected nodes implement the same logic – say the logic for X.1 – the feature works. Similarly, if all implement the logic for X.2, the feature works. However, if some of the nodes implement the logic for X.1, and others have implemented the logic for X.2, the outcome is unpredictable and the feature may not interoperate.
We call this problem "the confusion of tongues". It arises whenever there is more than one way to implement a particular feature amongst a set of nodes. While each approach is, by itself, conformant to the specifications, there are interoperability failures because of a heterogeneous selection of methodologies within a particular deployment.

This problem is ameliorated when the logic required for a particular feature exists almost entirely within a single node. Any feature involving multiple parties ultimately requires some form of logic in other nodes. However, when the logic required for a feature requires that the other nodes only support for the basic SIP specs - [RFC3261] and [RFC3263] - we call this a single ended feature. Single-ended features tend to be more interoperable because they rely on just the lingua franca - basic SIP - from everyone else. An example of a single-ended feature is mute, which can be done locally within a node without any signaling at all. Another feature is basic hold (without music), which requires only that the other side support [RFC3263].

Unfortunately, many features are fundamentally not single ended. A feature that is not single ended is called a multi-ended feature. Examples include transfer (which relies on at least support for REFER) and music-on-hold.

3. Concrete Examples

Several concrete examples can be demonstrated which demonstrate the confusion of tongues.

3.1. Call Forward No Answer

Call Forward No Answer (CFNA), is a very basic feature. In this feature, user X calls user Y. If user Y is not answering, the call is forwarded to another user, user Z. Typically this forwarding takes place after a certain amount of time.

Even for a simple feature like this, there are several ways of implementing it. Consider the reference architecture in Figure 1.
In this simple network, there are four "nodes" that are cooperating to implement this feature. There are three user agents, UA X, UA Y and UA Z. All three user agents are associated with a single proxy. When UA X makes a call to UA Y, the INVITE is sent to the proxy which delivers it to UA Y.

3.1.1. Approach 1: UA Redirects

In this approach, the call forwarding functionality is implemented in the user agents. The user agents have a field on the user interface that a user can enable to cause calls to be forwarded on no-answer. The user can also set up the forward-to URI through the user interface.

The basic call flow for this approach is shown in Figure 2.
When the call from UA X arrives at the proxy, it is forwarded to UA Y. User Y is not there, so UA Y rings for a time. After the call forward timeout has elapsed, UA Y generates a 302 response. This response contains a Contact header field containing the forward-to URI (sip:Z@example.com). This is received by the proxy, which recurses on the 3xx, causing the call to be forwarded to Z.

3.1.2. Approach II: Proxy Forwards

In this approach, the call forwarding functionality is implemented in the proxy. The proxy has a web interface that allows the user to set up the call forwarding feature and specify the forward-to URI.

The basic call flow for this approach is shown in Figure 3.
When the call from UA X arrives at the proxy, the proxy sends the INVITE to UA Y. UA Y rings for a time. The call timeout timer runs on the proxy. After the timeout has elapsed, the proxy generates a CANCEL, causing the call to stop ringing at UA X. It then consults its internal configuration, notes that call forwarding on no-answer is configured for user Y. It obtains the forward-to URI, and sends an INVITE to it. User Z answers and the call proceeds.

3.1.3. Approach III: UA Proxies

In this last approach, the user agent implements the call forwarding, but does so by acting as a proxy, forwarding the call to Z on its own. As in Approach I, the UA would have an interface on its UI for enabling call forwarding and entering the forward-to URI.

The basic call flow for this approach is shown in Figure 4.
UA X sends an INVITE to its proxy targeted for Y. The proxy sends this INVITE to UA Y. The user does not answer. So, after a timeout, the UA acts like a proxy and sends the INVITE back to P, this time with a Request-URI identifying Z. The proxy forwards this to Z, and the call completes.

3.1.4. Approach IV: Call Processing Language

In this approach, the proxy implements the call forwarding logic. However, instead of the logic being configured through a web page, it has been uploaded to the proxy server through a Call Processing Language (CPL) [RFC3880] script that the UA included in its registration request.

The basic call flow for this approach is shown in Figure 5.
This flow is nearly identical to the one in Figure 3, however, the logic in the proxy is guided by the CPL script.

3.1.5. Failure Cases

We have now described four different call forwarding implementations. All four are compliant to RFC 3261. All four assume some form of "feature logic" in some of the components in order to realize this feature. For Approach I, this logic is entirely in the UA, and consists of the activation of the feature, configuration of the forward-to URI, execution of the timer, and then causing of a redirect to the forward-to URI. This implementation of the feature is single ended. For approach II, the logic is entirely in the proxy, and consists of the activation of the feature through the web, configuration of the forward-to URI through the web, execution of the timer, and then causing of CANCEL and sequential fork to the forward-to URI. This implementation approach is also single-ended.

Figure 5: CFNA Approach IV
In approach III, all of the logic exists on the UA, and consists of
the activation of the feature, configuration of the forward-to URI,
execution of the timer, and then causing of a proxy to the forward-to
URI. This approach is also single-ended. In approach IV, all of the
feature logic is in the proxy, but it is implemented by CPL, and the
UA has a CPL implementation that establishes the forwarding number
configuration. Consequently, this approach is multi-ended.

If one considers several different combinations of implementation,
several error cases arise.

3.1.5.1. No One Implements

In this case, the UA assumes approach II (that is, it assumes the
proxy handles call forwarding), while the proxy assumes approaches I
or III (that is, the UA handles call forwarding). In this case, the
call will arrive at the proxy, which forwards it to UA Y, where it
rings indefinitely. The feature does not get provided at all.

3.1.5.2. Both Implement

In this case, the UA assumes approach I (that is, it assumes that it
handles call forwarding), and the proxy assumes approach II (that it,
it assumes that it handles call forwarding). In this case, assuming
that the forwarding number ends up being provisioned in both places,
the actual behavior of the system is a race condition. If the timer
fires first at the proxy, the call is forwarded to the number
configured on the proxy. If the timer fires first on the UA, the
call is forwarded to the number configured on the UA. If these
forwarding numbers are different, this results in highly confusing
behavior.

3.1.5.3. Missing Half

In this case, the UA implements CPL, but the proxy does not. Or, the
proxy implements CPL, but the UA does not. In either case, the logic
for the forwarding feature cannot be configured, and the feature does
not work.

4. Solution Considerations

There are many ways this interoperability problem can be solved. The
most obvious solution is to actually enumerate every specific feature
that we wish to support with SIP (Call Forward No Answer, Call
Forward Busy, Hold, Music-on-hold, and so on). Then, for each
feature, identify a specific call flow that realizes it, and describe
the exact functionality required in each component of the system. In
the case of call forward no answer, for example, we would choose one of the four approaches, define the information that needs to be configured (timeout, activation state, call forwarding URI), and describe the timer and how it operates. This approach would actually lead to excellent interoperability, but would come at high cost. The set of interoperable features would be limited to only those which we explicitly specify, and there would be little room for innovation.

To avoid this pitfall and others like it, a proper solution to the interoperability has to be structured in such a way that it achieves the following goals:

Covers Everything: Ultimately, the goal of the solution is to make things work in reality. This means that the solution has to cover all aspects of the feature that can be a source of interoperability problems. This includes traditional signaling, media, and even provisioning and configuration issues. For example, the failure of Section 3.1.5.3 was caused by an inconsistent provisioning mechanism between the UA and the server. Consequently, interoperability requires this mechanism to be agreed upon to the degree required for interop. The objective of BLISS is that the resulting specifications ensure that you can take a UA from one vendor, plug it into the server of another, and it works - full stop.

Avoid Enumeration: One of the main goals of SIP is to provide a rich set of features. If it requires a specification to be developed for each and every feature, this goal of SIP is lost. Instead, SIP will be limited to a small number of features and it will be hard to add new ones. Therefore, any solution to the interoperability problem must avoid the need to enumerate each and every feature and document something about it.

Allow Variability in Definition: It should not be necessary to rigorously define the behavior of any particular feature. It is possible for variations to occur that do not affect interoperability. For example, a variation on CFNA is that a provisional response can be sent back to the originator informing them that the call was forwarded. This variation can be implemented without impacting interoperability at all; if the originator can render or utilize the provisional response, things work. If they can’t things still work on the originator simply doesn’t get that part of the feature. We should allow this kind of localized variability in what each feature does, to preserve innovation.
Assume Multimedia: Though many of the features discussed so far are very telephony centric, they all apply and can be used with any number of media types. In addition, it is important that the solution to the interoperability problem not assume a particular media type. Unless the feature is specifically about a media type (instant message logging for example), it must be possible for it to work with all media types.

Allow Variability in Implementation: Whenever possible, the solution to the interoperability problem should strive to allow variations in how the implementations work, while preserving interoperability. For example, in the case of call forwarding, the central source of interoperability failure is that is unclear whether the UAs or proxies have responsibility for the forwarding logic. If the decision was made that this logic is in the UA, then either Approach I or Approach III will work. Consequently, it is not necessary to specify which of those two approaches is to be implemented; just that the UA performs the implementation.

Support a Multiplicity of Environments: SIP is utilized in a broad set of environments. These include large service providers targeted to consumers, enterprises with business phones, and peer-to-peer systems where there is no central server at all. SIP is utilized in wireless networks with limited bandwidth and high packet loss, and in high-bandwidth wired environments. It is the goal of this process that interoperability be possible using the same set of specifications for all cases. The problem is not restricted to just enterprises, even though many advanced features typically get associated with enterprise.

5. BLISS Solution Framework

The framework for solving this interoperability dilemma is called BLISS - Basic Level of Interoperability for SIP Services. This solution is actually a process that a working group can follow to identify interoperability problems and then develop solutions.

5.1. Phase I - Identify a Feature Group

The first step is to identify a feature or set of features which have been known to be problematic in actual deployments. These features are collected into bundles called a feature group. A feature group is a collection of actual features that all have a similar flow, and for which it is believed the source of interoperability failures may be common. A feature group can also have just one feature. For example, Call Forward No Answer, Call Forward Busy, Call Forward Unconditional are all very similar, and clearly all have the same
interoperability problem described in Section 3.1. However, the root issue with these flows is that there needs to be a common understanding of where call treatment feature logic is executed, and how the desired treatment is signaled from the user to the place where it is implemented. Thus, other features that are similar, in that they make a decision on call handling based on user input or conditions, will likely also benefit from consideration.

Thus, a feature group is defined by a characteristic that identifies a large (and in fact, possibly infinite) number of actual "features" that all belong to the group. This characteristic is called its functional primitive. The first step in the BLISS process is to identify feature groups and their functional primitives that are narrow enough so they are meaningful, yet broad enough that they are not overly constraining. This is not exact, and the initial definitions do not need to be exact. They can be refined as the BLISS process proceeds. Indeed, in many cases, investigations can start with a single feature - for example call park - and analysis can proceed with just one. As work proceeds, the definition of the feature group can be broadened. In the case of CFNA, clearly a functional primitive of "call forwarding features that execute on no-answer" is too narrow. A functional primitive of "features that handle an initial INVITE" is too broad. An ideal starting point would probably be, "features that result in a retargeting or response operation that depend on user-specified criteria". This covers all of the call forwarding variations, but also includes features like Do-Not-Disturb.

Each feature group should be defined in a similar way, through the definition of a functional primitive by which one could decide whether or not a particular feature was included. As part of this definition, the group can consider specific features and agree whether or not they are covered by the primitive. For example, would "send call to voicemail" be covered by the functional primitive "features that result in a retargeting or response operation that depend on user-specified criteria"? The answer is yes in this case. Discussion of what features are covered by a functional primitive is part of the discussion in this phase.

Care must be taken not to define the functional primitive in such a way as to eliminate the possibility of any but a defined and enumerated set of features from being included. The functional primitive should clearly cover features which are in existence today, and of interest, but allow for future ones that could be covered by the primitive. This avoids the perils of enumeration as discussed in Section 4.
5.2. Phase II - Gather Data

With the functional primitive identified and a shared understanding of which features fit within it, the next step is for working group participants to document how their implementations implement features in the group.

This can be done any number of ways. Ideally, call flows would be collected that document the mechanism implemented by each vendor. However, experience has shown that vendors frequently consider this information proprietary or sensitive. An alternate model is to define a survey which asks high level questions about how the feature or feature group is implemented. Yet another model is to merely ask vendors to submit freeform text which describes their implementation.

It is a decision of the working group as to whether to actually publish the collected information as an RFC, use them as a working internet draft, or just keep them on a web page. The gathered data is not an output of the BLISS process; they are only an intermediate step. If the information is to be published as an RFC, it is suggested that a single document be published for each functional primitive. The title of the document would be something like, "Enumeration of Existing Practices for Foo" where "Foo" is some moniker for the functional primitive. Such a document must be clear that it is NOT a best practice. It would strictly be informational.

5.3. BLISS Phase III - Problem Definition

With current practice for a particular feature group collected, the next step in the process is to analyze the data. The analysis considers each permutation of implementation of logic from the data gathered in the previous phase, and determines which combinations work, and which ones do not.

General speaking, this analysis is performed by taking the components associated with the feature (for example, in the case of CFNA, there are four components - three UA and one proxy), and for each one considering what happens when it implements one of the logical behaviors identified in the cases identified from the previous phase. Thus, if four variations on a feature have been submitted to the group, and that feature has four components, there are 16 possible deployment scenarios that can be considered. In practice, many of these are equivalent or moot, and therefore the number in practice will be much smaller. The group should work to identify those cases that are going to be of interest, and then based on the logic in each component, figure out where interoperability failures occur.

This phase can be accomplished using documents that contain flows, or
can be purely a thinking exercise carried out on the mailing list or in a design team. In all likelihood, it will depend on the feature group and the level of complexity. Regardless of the intermediate steps, the end goal of this phase should be an enumeration of combinations with known interoperability problems. One possible output would look exactly like the contents of Section 3.1.5, which describe several failure modes that are possible.

5.4. BLISS Phase 4 - Minimum Interop Definition

The final step in the BLISS process is to repair the interoperability failures identified in the previous phase. This is done by coming up with a set of recommendations on behaviors of various components, such that, were those rules to be followed, those interoperability failure cases would not have occurred.

In some cases, these recommendations identify a place in the network where something has to happen. Again, considering our CFNA example, the primary recommendation that needs to be made is where the logic for call handling should happen - in the UA, in the proxy, or both. This is likely to be a contentious topic, and the right thing will certainly be a function of participant preference and use cases that are considered important. But, no one ever said life is easy.

In other cases, these recommendations take the form of a specification that needs to be implemented. For example, CFNA can be implemented using CPL, in which case both the UA and proxy need to support it. If the group should decide that CPL is the main way to implement these features, the recommendation should clearly state that CPL is required in both places.

Indeed, if a particular functional primitive requires any functionality to be present in any node that goes beyond the "common" functions in RFC 3261, the recommendations need to state that. For example, if a particular feature can be implemented using S/MIME, and the group decides that S/MIME is the required everywhere for this feature to work, that recommendation should be clearly stated.

In some cases, only a part of a specification is required in order for the features in a feature group to be interoperable. In that case, the group should identify which parts it is. In the example of CPL, RFC 3880 [RFC3880], the ability to support non-signalling controls is not necessary to achieve an implementation of this feature group. So, the recommendation could be that this part is not required.

Another key part of the recommendations that get made in this phase, are recommendations around capability discovery. If a decision is
made that says there are multiple different ways that a feature can work, and it is necessary to know which one is in use, some kind of capability exchange is required. Consider once more CFNA. If the recommendation of the group is that all proxies have to implement the logic associated with the feature, but phones can also optionally do it, the UA needs to determine whether it has to be responsible for this feature or not. Otherwise, the failure mode in Section 3.1.5.2 may still happen. This particular problem can be resolved, for example, by the use of a feature tag in the Require header field that would inform the proxy whether it should or should not provide the feature. The BLISS recommendations for this phase need to include these kinds of things, if they are necessary for the feature group.

The recommendations in this phase, covering specific protocols or pieces of protocols, places where functionality needs to reside, and capability negotiations and controls, are all the final output of the BLISS process. If the group has done its job well, with these recommendations, a (potentially large) class of features will interoperate, yet there will be room for innovation.

6. Structure of the BLISS Final Deliverable

This section describes a recommended template for the final BLISS deliverable - the recommendations of Section 5.4.

There will typically be a document produced per functional primitive. The title of the document must clearly articulate the functional primitive that is being addressed. For example, if the functional group is forwarding, an appropriate title would be, "Best Practices for Interoperability of Forwarding Features in the Session Initiation Protocol". It is important that the feature group be well articulated in the title, so that implementors seeking guidance on these features can find it.

Similarly, the abstract of the document is very important. It has to contain several sentences that more clearly articulate the functional primitive definition. In addition, the abstract should contain example features, by name or description, that are defined by the functional primitive. Again, this is important so that people looking to understand why feature foo doesn’t work, can find the right specification that tells them what they need to do to make it work.

The body of the document needs to first clearly and fully define the functional primitive. It must then enumerate features that are in the group. Next, the document should summarize the problems that have arisen in practice that led to the interoperability failures.
This would basically be a summarization of the results of phase III of the BLISS process. If the feature group were call forwarding, this part of the document would discuss how the primary problem is where in the network the actual feature logic lives – UA or proxy, and that the interop problems occur because of inconsistent choices between UA and proxy. The final part of the document is explicit recommendations. This would typically be broken out by component types – a section for UA, a section for proxies or "servers" more generally (so that it is clear that B2BUAs aren’t excused from the interoperability requirements). This section would clearly state the requirements for this feature group – specifications, portions of specifications, and capability behaviors that are required.

7. Security Considerations

Interoperability of security functions is also a critical part of the overall interoperability problem, and must be considered as well.

8. IANA Considerations

There are no IANA considerations associated with this specification.

9. Acknowledgements

I’d like to thank Shida Schubert, Jason Fischl, and John Elwell for actually running the BLISS process and providing feedback on its effectiveness.

10. Informative References


[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer


Author’s Address

Jonathan Rosenberg
Cisco
Edison, NJ
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net
Full Copyright Statement

Copyright (C) The IETF Trust (2008).

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 DISCLAIMS 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 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.

Acknowledgment

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