Optimizing Federated Presence with View Sharing
draft-ietf-simple-view-sharing-01

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 January 15, 2009.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

Presence federation refers to the exchange of presence information between systems. One of the primary challenges in presence federation is scale. With a large number of watchers in one domain obtaining presence for many presentities in another, the amount of notification traffic is large. This document describes an extension to the Session Initiation Protocol (SIP) event framework, called view sharing. View sharing can substantially reduce the amount of
traffic, but requires a certain level of trust between domains. View sharing allows the amount of presence traffic between domains to achieve the theoretical lower bound on information exchange in any presence system.

Table of Contents

1. Introduction ........................................... 3
2. Overview of Operation .................................. 4
3. RLS Behavior ............................................ 7
   3.1. On Receipt of a Resource List Subscription Request ... 8
      3.1.1. Authentication and Authorization .................. 8
      3.1.2. No Active Back-End Subscription .................. 8
      3.1.3. Active Back-End Subscription ..................... 9
   3.2. Processing NOTIFY Requests ........................... 10
      3.2.1. Processing ACL-Infos .............................. 10
      3.2.2. Processing Presence Documents ................... 11
      3.2.3. Processing Back-End Terminations ................ 12
4. Presence Agent Behavior .................................. 12
   4.1. Authentication and Authorization ....................... 12
   4.2. Processing Initial SUBSCRIBE Requests ................. 12
   4.3. SUBSCRIBE Refreshes .................................. 13
   4.4. Policy Changes ...................................... 13
   4.5. Presence State Changes ............................... 15
5. ACL Format .............................................. 15
   5.1. Document Structure and Semantics .................... 15
   5.2. Trust Considerations when Construcing ACLs .......... 17
   5.3. Example Documents ................................... 18
   5.4. Rule Determination Algorithm ......................... 19
   5.5. XML Schema .......................................... 21
6. Performance Analysis ..................................... 21
7. Requirements Analysis .................................... 22
8. Security Considerations ................................. 24
9. IANA Considerations ..................................... 25
   9.1. MIME Type Registration ............................... 25
   9.2. URN Sub-Namespace Registration ...................... 26
   9.3. Schema Registration .................................. 27
10. Acknowledgements ....................................... 27
11. References ............................................. 27
    11.1. Normative References ............................... 27
    11.2. Informative References ............................. 28
Authors’ Addresses ......................................... 29
Intellectual Property and Copyright Statements ............... 30
1. Introduction

Presence refers to the ability, willingness and desire to communicate across differing devices, mediums and services [RFC2778]. Presence is described using presence documents [RFC3863] [RFC4479], exchanged using a SIP-based event package [RFC3856].

Presence federation refers to the interconnection of disparate systems for the purposes of sharing presence information. This interconnection involves passing of subscriptions from one system to another, and then the passing of notifications in the opposite direction.

[I-D.ietf-simple-interdomain-scaling-analysis] has analyzed the amount of traffic, in terms of messages and in terms of bytes, which flow between systems in various federated use cases. These numbers demonstrate that presence traffic can be a substantial source of overhead. The root cause of this scale challenge is the so-called multiplicative effect of presence data. If there are N users, each of which have B buddies on their buddy list, and each buddy changes state L times per hour, the amount of notification traffic is proportional to N*B*L. For example, in the case of two extremely large public IM providers that federate with each other (each with 20 million users), [I-D.ietf-simple-interdomain-scaling-analysis] shows that the amount of traffic due to these steady state notifications is 18.4 billion messages per day, an astoundingly large number. Overhead for subscription maintenance and refreshes brings the total to 25.6 billion per day.

The overhead for SIP-based presence can be reduced using SIP optimizations. In particular, [I-D.ietf-sip-subnot-etags] can reduce the amount of traffic due to refreshes and polls. However, this optimization targets the overhead, and doesn’t address the core scaling problem – the multiplicative effect of presence data.

For this reason, there is a clear need to improve the scale of SIMPLE in federated environments.

[I-D.ietf-sipping-presence-scaling-requirements] has laid out a set of requirements for optimizations. The essence of these requirements are that the extension should improve performance, while being backwards compatible and supporting the privacy and policy requirements of users.

This document defines a mechanism called view sharing in support of those requirements. The key idea with view sharing is that when there are many watchers in a domain to a single presentity in another domain, each of which is actually going to get the exact same presence document, the domain of the watchers extends a single
subscription to the domain of the presentity, and the domain of the presentity sends a single copy of the presence document back to the domain of the watcher.

In the case of a pair of large providers that are peering with each other, this mechanism can result in a significant savings. Assuming a symmetrical system whereby the average buddies per watcher is \( B \) and the average number of watchers for a user is also \( B \), if most buddies are in one domain or the other, this optimization can reduce the overall subscription overhead and notification traffic by a factor of \( B/2 \). In cases where there are a large number of small domains, this mechanism is less useful. Of course, in such cases, the amount of traffic between any pair of domains is small anyway.

2. Overview of Operation

The extensions works in the environment shown in Figure 1. The environment assumes two domains. There are some number of watchers (\( W_1 - W_3 \)) in the domain on the left, which we call the watching domain. All of those watchers are interested in the presence of a single presentity \( P_1 \) in the domain on the right, which we call the serving domain. The watchers all make use of a resource list server (RLS) [RFC4662] which stores their buddy lists and performs the buddy list expansion. Consequently, when each watcher subscribes to their buddy list on the RLS, in absence of any optimizations, the RLS will generate three separate subscriptions to \( P_1 \), each of which reaches the presence server in the serving domain.
Of course, in practice each domain will act as both a watching domain and a serving domain, thus implementing both sides of the system.

The initial SUBSCRIBE generated by the RLS includes a SIP option tag "view-share" in the Supported header field, indicating that the RLS supports the view sharing extension. If the presence server also supports the extension, it makes use of it and includes an indication of this fact in the Require header field in the SUBSCRIBE response and in NOTIFY requests it generates.
View sharing requires a level of trust between the two domains. Consequently, the connection between them utilizes TLS with mutual authentication. The presence server verifies that the certificate presented in the mutual authentication matches the domain of the watcher.

If this is the first subscription from domain 1 for that particular presentity, the presence server accepts the subscription (assuming the watcher is authorized of course). The notifications sent to the RLS include two separate pieces of state. One is the actual presence state for the presentity. The other is an Access Control List (ACL) document. This document describes the set of other watchers from the originating domain, if any, who are authorized to see exactly the same presence document - in other words, the set of users that share the same view. Should one of those watchers seek the presence of that presentity, the RLS from the originating domain does not need to generate a back-end subscription; rather, it just uses the presence document it is receiving from the original subscription, and passes it to both watchers. The ACL can also list users in the originating domain that are authorized to subscribe to that presentity, but who will end up receiving a different view. Should one of those watchers subscribe, the RLS knows that it must perform a back-end subscription to obtain that view. The ACL can also list watchers in the originating domain that are not authorized at all, in which case the RLS could immediately reject their subscriptions. Finally, if the ACL says nothing about a particular watcher, it means that the presence server has elected to say nothing about what view this watcher will receive. Consequently, the RLS must perform a back-end subscription should that watcher subscribe to the presentity.

Other subsequent subscriptions to the same presentity from the same originating domain are processed in a similar way. However, the presence server in the serving domain will keep track of the set of subscriptions to the same presentity from the same RLS which are to receive the same view. When a presence notification is to be sent, instead of sending it on all subscriptions, the notification is sent on just a single subscription.

Should the authorization policies in the serving domain change, an updated ACL is sent, informing the watching domain of the new policies. This may require the watching domain to extend a back-end subscription to obtain a view, or may change the view an existing watcher is receiving, and so on.

The ACL allows the serving domain a great deal of flexibility in the level of trust it imparts to the watching domain. The following are all possible approaches that the serving domain can utilize:
No Trust: When a presence server receives the subscription, it elects not to use this mechanism at all using the negotiation techniques defined here.

Minimal Trust: When a watcher subscribes to a presentity, the ACL generated for that subscription includes only that watcher, along with an identifier for their view. Consequently, for each watcher in domain 1 there will be a backend subscription to domain 2. However, should multiple watchers share the same view, the presence server in domain 2 sends a single presence document on one of the subscriptions, and the RLS uses this for all of the other watchers with the same view. With this approach, domain 2 never discloses the list of authorized watchers ahead of time, and it has full knowledge of each watcher that is subscribed. However, it gets the performance benefits of reducing the amount of notification traffic.

Partial Trust: When a watcher subscribes to a presentity, the ACL generated for that subscription includes that watcher and all other watchers authorized for that same view. Consequently, there will only be one backend subscription from the RLS to the presence server for each view. However, the full set of authorized watchers is not disclosed ahead of time, only those that will get the same view. With partial trust, the presence server will not know the full set of watchers currently subscribed.

Full Trust: When a watcher subscribes to a presentity, the ACL generated for that subscription includes that watcher and all other watchers that are authorized for that view, and all other views, along with a rule that says that all other watchers get rejected. In this case, as with partial trust, there is only one backend subscription from the RLS to the presence server for each view. The full set of watchers is disclosed ahead of time as well. The presence server will not know the full set of watchers currently subscribed.

Depending on the level of trust, the mechanism trades off inter-domain messaging traffic for increased processing load in the RLS to handle the ACL documents.

3. RLS Behavior

This section defines the procedures that are to be followed by the RLS. It is important to note that, even though this specification defines an extension to the SIP events framework, that extension is only useful for the back-end subscriptions generated by an RLS. The extension defined here is not applicable or useful for individual
users generating subscriptions. Indeed, if it were utilized by individual users, it has the potential for violations of user privacy. See Section 8 for a discussion.

3.1. On Receipt of a Resource List Subscription Request

When the RLS receives a subscription to a resource list which includes some presentity P in another domain, it follows the rules defined here.

3.1.1. Authentication and Authorization

First, the RLS MUST check a configured list of peer domains for which this extension is to be applied. Because of the potential privacy disclosures involved in unauthorized use of this facility, it can only be used between pairs of domains which have a pre-arranged agreement to utilize it. If the domain of the presentity P matches one of the configured list of peer domains, the RLS is permitted to utilize this extension. If not, the extension MUST NOT be used.

Next, the RLS MUST send the SUBSCRIBE request over a mutually authenticated TLS connection. The RLS MUST check that the subjectAltName in the certificate of its peer contains a domain name that is a match for the domain of the URI of the presentity. If they are not a match, view sharing cannot be utilized for this subscription.

The procedures followed by the RLS after this point depend on whether the RLS already has a backend subscription to the presentity that is in the active state, and for which an ACL has been received.

3.1.2. No Active Back-End Subscription

The RLS MUST generate a back-end subscription to obtain the state of the presentity. The RLS MUST include a Supported header field in the request with the option tag "view-share". The Accept header field MUST be present in the request and MUST include the "application/aclinfo+xml" MIME type amongst the list of supported types. The RLS MUST include an +sip.instance Contact header field parameter [I-D.ietf-sip-outbound] to uniquely identify the RLS instance.

Note that it is possible that two watchers, in a short period of time, both subscribe to their resource lists, both of which include presentity P. This will cause the RLS to generate two back-end subscriptions at around the same time. The RLS is forced to generate the second back-end subscription because it doesn’t have an active back-end subscription that has yet generated an ACL. Once both subscriptions become active and generate ACLs, if the watchers are
receiving the same view and both ACLs contain both watchers, the RLS SHOULD terminate one of the back-end subscriptions.

3.1.3. Active Back-End Subscription

In this case, the RLS already has at least one back-end subscription to the target presentity \( P \), and it has received at least one ACL for that presentity. It has received a resource list subscription from watcher \( W \) which includes presentity \( P \). Based on the procedures of Section 3.2.1, the RLS will keep, for each presentity, the list of the most recent ACLs received on each back-end subscription currently in place. This is called the current ACL list.

For each ACL \( A_i \) in the current ACL list, the RLS performs the rule determination algorithm of Section 5.4 to compute the rule ID \( R \) for the watcher \( W \). This represents the view that the watcher is supposed to receive.

Next, the RLS goes through all subscriptions it currently has for presentity \( P \). For each one, it takes the identity of the watcher for that actual subscription. The identity for the watcher for that actual subscription is equal to the asserted identity included in the back-end subscription. For example, if SIP Identity [RFC4474] is utilized, this would be the URI present in the From header field of the back-end SUBSCRIBE. Call the watcher identity for each subscription \( W_j \).

Next, the RLS computes the rule determination algorithm of Section 5.4 to compute the rule ID \( R_j \) for the watcher \( W_j \) on each subscription \( j \). This represents the rule ID for the view being delivered on that subscription.

Then, processing depends on the values of \( R \) and \( R_j \):

- If \( R \) is null, it means that the ACL doesn’t specify the view for this watcher. The RLS MUST generate a back-end subscription to presentity \( P \), and MUST use watcher \( W \) as the identity in the back-end subscription.

- If \( R \) is not null, but the associated rule is blocked, it means that the watcher will be rejected. The RLS SHOULD NOT perform another back-end subscription, and must act as if it had created a back-end subscription which was rejected.

- If \( R \) is not null, and there is at least one subscription \( j \) for which \( R_j = R \), it means that subscription \( j \) is already generating notifications for the view that watcher \( W \) is supposed to receive. In that case, the RLS SHOULD NOT generate a back-end subscription.
for P on behalf of W. Rather, it should treat the existing back
end subscription j as if it were the back-end subscription for W,
and follow the guidelines of RFC 4662 [RFC4662] based on that.
Subscription j is called the generating subscription for watcher
W, and the actual watcher associated with subscription j, Wj, is
called the generating watcher Wgen for watcher W.

- If R is not null, but there is no subscription j for which Rj=R,
it means that the RLS is not yet receiving the view that watcher W
requires. The RLS MUST generate a back-end subscription to
presentity P, and MUST use watcher W as the identity in the back-
end subscription.

### 3.2. Processing NOTIFY Requests

If a NOTIFY request arrives with a Require header field that includes
the "view-share" option tag, it MUST be processed according to the
rules of this specification.

#### 3.2.1. Processing ACL-Infos

If the contents of the NOTIFY are of type "application/aclinfo+xml",
the subscriber processes the body as described here.

For each presentity that the RLS has at least one back-end
subscription for, the RLS MUST remember the most recent aclinfo
received on each back-end subscription. This is called the current
ACL list for the presentity. This set of aclinfo is used in the
processing of subscription requests, as described in Section 3.1.3.

The serving domain can change policies at any time. When it does, it
sends an updated ACL on one or more subscriptions. The RLS MUST
store this ACL. Furthermore, the ACL might impact the views being
received by watchers, and may impact the state of the back-end
subscriptions.

The RLS computes the set of watchers Wi which have a resource list
subscription that includes the presentity P for whom an updated ACL
has just been received. For each Wi, it performs the view
determination algorithm (see Section 5.4 on the current ACL set. Let
Ri be the view associated with watcher Wi. If Ri has not changed
from prior to the receipt of the new ACL, no action is taken.
However, if it has changed, the RLS takes the set of current back-end
subscriptions, and for each subscription j, computes the view
determination algorithm for its associated watcher Wj, to produce Rj.
The action to take depends on what has changed:
o If Ri is now null, it means that the serving domain has changed the views associated with watcher Wi, and this new view is not known to the RLS. The RLS MUST generate a new back-end subscription on behalf of watcher Wi for presentity P to obtain this view.

o If Ri is now a blocked rule, it means that the serving domain has now blocked Wi from obtaining the presence of the presentity. The RLS must act as if it had a back-end subscription on behalf of watcher Wi which was terminated.

o If Ri is not null and not blocked, and if there is an Rj which matches the new Ri, it means that the serving domain has changed the views associated with watcher Wi, and changed them to another view already being received by the RLS. The RLS MUST treat this back-end subscription j as if it were the back-end subscription to presentity P for watcher Wi. If the most recent presence document received on this back-end subscription is not a semantic match for the presence document most recently delivered to Wi for presentity P, the RLS MUST send this most recent presence document to watcher Wi.

o If Ri is not null and not blocked, but there is no Rj which matches the new Ri, it means that the serving domain has changed the views associated with watcher Wi, and this new view is not one currently being delivered to the RLS. The RLS MUST generate a new back-end subscription on behalf of watcher Wi for presentity P to obtain this view.

Furthermore, if there are now two back-end subscriptions j1 and j2 for which Aj1 = Aj2, the RLS SHOULD terminate one of those two subscriptions. Two ACL documents are considered equal if they enumerate the same set of rules with the same members for each rule.

3.2.2. Processing Presence Documents

If the contents of the NOTIFY is a presence document, the RLS follows the procedures defined here.

Let Wj be the watcher on the subscription j on which the presence document was just received. Let Rj be the results of running the rule determination algorithm on Wj using the current ACL set. Next, the RLS takes the set of watchers Wi which have presentity P on their buddy lists. The RLS then runs the rule determination algorithm on each Wi using the current ACL set, producing Ri for each watcher Wi. For each Ri that is equal to Rj, the RLS MUST utilize the presence document just received as if the back-end subscription j was in fact for watcher Wi. This will typically cause that presence document to
be sent in a NOTIFY request to each such watcher, though each watcher may have some kind of filtering policy which causes the RLS to modify the document prior to delivery.

3.2.3. Processing Back-End Terminations

If the NOTIFY request from the serving domain terminates the back-end subscription, it may be because the watcher W_j associated with that subscription is no longer permitted to view the presence of the presentity.

The ACL associated with the subscription MUST be removed from the current ACL set. The procedures of Section 3.2.1 MUST be performed to adjust back-end subscriptions, if needed.

4. Presence Agent Behavior

When a presence agent receives a SUBSCRIBE request containing a Supported header with the value "view-share", and it wishes to utilize view sharing for this subscription, it follows the procedures defined here.

4.1. Authentication and Authorization

First, the presence agent MUST have received the SUBSCRIBE request over a mutually authenticated TLS connection. If it had not, view sharing cannot be utilized for this subscription. The presence agent MUST check that the subjectAltName in the certificate of its peer contains a domain name that is a match for the domain of the URI of the watcher. If they are not a match, view sharing cannot be utilized for this subscription.

Assuming they are a match, the presence agent MUST check a configured list of peer domains for which this extension is to be applied. Because of the potential privacy disclosures involved in unauthorized use of this facility, it can only be used between pairs of domains which have a pre-arranged agreement to utilize it. If the domain of the watcher W matches one of the configured list of peer domains, the presence agent is permitted to utilize this extension. If not, the extension MUST NOT be used.

4.2. Processing Initial SUBSCRIBE Requests

First, the subscription is processed as it normally would be, including authorization and policy around the presence document to be delivered to the watcher. Furthermore, if the presence agent wishes to utilize view sharing for this subscription, it MUST include a
Require header field in the first NOTIFY request (and indeed any subsequent ones) it sends confirming this subscription, and that NOTIFY MUST contain the "view-share" option tag.

Furthermore, the initial state sent by the presence agent MUST include an ACL document. It is formatted according to the rules and considerations in Section 5.

The initial state sent by the presence agent might include an actual presence document. In particular, a presence document MUST be sent if one of the following is true:

- There is only one subscription from the watching domain to this presentity that has the view associated with the watcher.
- There is more than one subscription from the watching domain to this presentity with the same view, but the +sip.instance parameter for the remote target (as conveyed in the Contact header field of the SUBSCRIBE) differs. In other words, these subscriptions are from the same domain, but from different RLS in the watching domain. Each RLS in the watching domain needs to get their own copy of the view for a particular presentity.

If one of these conditions is not true, the presence agent SHOULD NOT send an initial presence document on this subscription.

If an ACL and a presence document are to be delivered, they MUST be delivered in a separate NOTIFY request (unless the subscriber indicated support for multipart, in which case the content MAY be included in a single NOTIFY with multipart content).

4.3. SUBSCRIBE Refreshes

When the presence agent receives a SUBSCRIBE refresh, it MUST send the most recent ACL document, and if presence documents are being sent for this subscription, the most recent presence document.

4.4. Policy Changes

There are several different types of policy changes that can occur:

- If the definitions for a particular rule change, the presence agent MUST assign a new rule ID for that rule. For each subscription to a presentity which contained that rule, the presence agent MUST send an updated ACL which includes a rule with this new rule ID.
If some watcher W was previously associated with rule X and is now associated with rule Y, the presence agent checks if it has any subscriptions from watcher W. If it does, it MUST send an updated ACL on that subscription. Based on the rules in Section 5, this ACL will contain rule Y and will at least include W amongst the list of members. Furthermore, if there were subscriptions from other watchers, but the presence agent had previously sent an ACL on the subscription which was a match for W, it MUST send an updated ACL on that subscription. This updated ACL MAY omit a statement about rule Y or MAY include it. However, the updated ACL MUST NOT claim that watcher W will receive rule X.

If some watcher W was previously associated with rule X and is now blocked, the presence agent checks if it has any subscriptions from watcher W. If it does, it MUST terminate the back-end subscription. If it doesn’t, but it has a subscription from some other watcher which had included a rule that was a match for W, the presence agent MUST send an updated ACL on that subscription. This updated ACL MAY omit any statement about watcher W, or MAY include them as part of a blocked rule in that ACL.

If some watcher W was previously blocked and is now permitted and associated with some rule X, the presence agent checks if it had any subscriptions from some other watcher which included a blocked rule that matched watcher W. If it had, it MUST send an updated ACL on that subscription. That updated ACL MAY omit any statement about watcher W, or MAY indicate that watcher W is now associated with rule X.

Of course, a policy change will also potentially alter the presence documents that are associated with a view. If so, the presence agent MUST send an updated document on a subscription if one of the following is true:

- There is only one subscription from the watching domain to this presentity that has the view associated with the watcher.

- There is more than one subscription from the watching domain to this presentity with the same view, but the User-Agent header field in the request differs between them.

If neither is true, the presence agent MUST select one subscription amongst the several which share the same presentity, view, and User-Agent header field, and sent an updated notification on that subscription. The choice of subscriptions is arbitrary and MAY change for each notification.
4.5. Presence State Changes

If the state of some presentity changes, the presence agent may need to send an updated notification on a subscription. The presence agent MUST send an update on a subscription if one of the following is true:

- There is only one subscription from the watching domain to this presentity that has the view associated with the watcher.
- There is more than one subscription from the watching domain to this presentity with the same view, but the User-Agent header field in the request differs between them.

If neither is true, the presence agent MUST select one subscription amongst the several which share the same presentity, view, and User-Agent header field, and sent an updated notification on that subscription. The choice of subscriptions is arbitrary and MAY change for each notification.

5. ACL Format

An ACL document is an XML [W3C.REC-xml-20001006] document that MUST be well-formed and MUST be valid according to schemas, including extension schemas, available to the validater and applicable to the XML document. ACL documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying ACL documents and document fragments. The namespace URI for elements defined by this specification is a URN [RFC2141], using the namespace identifier ‘ietf’ defined by RFC 2648 [RFC2648] and extended by RFC 3688 [RFC3688]. This URN is:

urn:ietf:params:xml:ns:aclinfo

5.1. Document Structure and Semantics

An ACL document informs a watching domain of the set of views that can be received by that domain, and associates specific watchers with specific views. It is very important to understand that the ACL document does not convey the actual processing that will be applied by the serving domain. It does not indicate, for example, whether geolocation is present in a presence document, or which rich presence [RFC4480] data elements will be conveyed. It merely provides grouping - indicating which watchers from the watching domain will receive the same view.

Each ACL document starts with the enclosing root element <acl-list>.
This contains the list of rules defined by the ACL. Each rule is represented by the <rule> element. Each rule represents a specific view, which is generated by the presence server based on its authorization, composition and filtering policies. Each rule is associated with a rule ID, which is a mandatory attribute of the <rule> element. This ID is scoped within a single presentity. That is, the IDs for two rules for different presentities are unrelated.

The <rule> element also contains an optional "blocked" boolean attribute. If "true", it means that the rule specifies that the associated set of watchers will be rejected, should they subscribe. This can be used by the watching domain to avoid performing back-end subscriptions to users which will only be blocked anyway.

Each <rule> contains the set of users that will receive the corresponding view. This can be described by an enumerated set or by a default. If it is an enumerated set, the <rule> is followed by a sequence of <member> elements, each of which contains a SIP URI for the watcher that will receive that view.

The default view is specified by including a single child element for <rule> - <other>. The default view applies to all watchers except those enumerated by other rules. For this reason, an ACL document which contains a default view MUST include the rule IDs and associated members for all other views that are delivered to watchers. For example, consider a presentity that has three views. View 1 is delivered to watchers A and B. View 2 is delivered to watcher C. View 3 is delivered to everyone else. An ACL document that includes the default view must also include views 1 and 2 with watchers A, B, and C.

In contrast, an ACL document that does not include a default does not need to include all views, and it does not need to include all members for a particular view. Using the example above, it is valid to include an ACL document which includes only view 1 with watcher 1.

If two URI are present within <member> elements within the same <rule>, it represents a contract by the presence server that both users MUST get the same view. Formally, if the presence server were to receive a subscription from each watcher, both subscriptions would be accepted or both would be rejected, and if accepted, each subscription would receive semantically identical presence documents at approximately the same time.

Even if two users will receive the same view, a presence server MAY assign each to a different view ID. There is no requirement that two unique views actually contain different presence data. The only requirement is that, if two users are listed within the same rule,
that they do in fact receive the same view.

An ACL document delivered in a subscription from watcher W MUST include the view associated with watcher W and MUST include watcher W explicitly in a <member> element or implicitly by presence of an <other> element.

5.2. Trust Considerations when Constructing ACLs

The semantics above give very little guidance about what a presence server should include in an ACL. The amount of information to convey depends on the level of trust between the watching and serving domains.

Optimal performance is achieved when the ACL document for a presentity includes all views that the server might ever deliver, and includes all members for each view, including any defaults and blocked rules. However, this informs the watching domain of the set of allowed and blocked watchers, and associated groupings amongst watchers.

Slightly worse performance is achieved when the ACL document for a presentity sent in a subscription from watcher W includes only a single view - the one for watcher W - along with the full set of watchers which will also receive that view, assuming it is not the default view. If the view is the default view, the document can include just watcher W. This approach will cause back-end subscriptions from every watcher that will receive the default, but it discloses less information to the watching domain. In particular, the full set and number of views is never known by the watching domain. The fact that a view is default is never known by the watching domain. The full set of users that are permitted to view the presence of the presentity is never disclosed to the watching domain. The performance of this approach is still reasonably good when the default rule is blocked. However it is much less effective when the default is not blocked, and many watchers receive the default.

Another choice for construction of ACL documents is to include, in a subscription from watcher W, a single rule containing the rule ID for the view that watcher W will receive, along with a single member - W. This approach will still result in a back-end subscription from each watcher. However, a single notification is sent for each view, rather than one per watcher. The benefit of this construction is that it provides the watching domain no additional information about the authorization policies of the presentity than if this extension were not in use at all.
5.3. Example Documents

The example document in Figure 2 shows the case when there is maximum trust between domains. The full set of watchers, include a blocked default, is included.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid XML Studio 1.0.6.0 (http://www.liquid-technologies.com) -->
<acl-list xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <rule id="6228">
    <member>sip:user1@example.com</member>
    <member>sip:user2@example.com</member>
    <member>sip:user3@example.com</member>
    <member>sip:user4@example.com</member>
    <member>sip:user5@example.com</member>
  </rule>
  <rule id="3584">
    <member>sip:user6@example.com</member>
  </rule>
  <rule id="1735">
    <member>sip:user7@example.com</member>
    <member>sip:user8@example.com</member>
    <member>sip:user9@example.com</member>
    <member>sip:user10@example.com</member>
    <member>sip:user11@example.com</member>
  </rule>
  <rule blocked="true" id="9433">
    <other />
  </rule>
</acl-list>
```

Figure 2: Example with Maximum Trust

The example in Figure 3 shows a moderate level of trust. This ACL only shows the view associated with the watcher user1.
Figure 3: Example with Partial Trust

The example in Figure 4 shows the minimal level of trust. This ACL would be sent in a subscription to user1.

Figure 4: Example with Minimal Trust

5.4. Rule Determination Algorithm

Several steps in the processing of the ACL require that the RLS in the watching domain execute the rule determination algorithm for watcher W on an ACL set. This algorithm is a simple algorithm which takes, as input, a watcher W with a given SIP URI, and a set of ACL documents Ai, and returns as output, a rule ID R, which is the rule ID for the view that, according to the set of ACLs, watcher W should receive.

The algorithm proceeds as follows. First, each Ai is matched to W. ACL Ai is a match for watcher W if:

- ACL Ai contains a <member> tag whose URI is a match, based on URI equality, for W, or
- none of the <member> tags in Ai contain a URI that is a match, based on URI equality, for W, but there is an <other> element in Ai
If no ACL Ai matched, the algorithm returns a null result.

For each ACL Ai that matches based on the rules above, take the id of the enclosing <rule> element that contained the <member> or <other> element that caused the match. For ACL Ai, this is rule Ri. For example, consider the following ACL:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<acl-list xmlns=urn:ietf:params:xml:ns:aclinfo>
  <rule id="1">
    <member>sip:user1@example.com</member>
    <member>sip:user2@example.com</member>
  </rule>
  <rule id="2">
    <member>sip:user3@example.com</member>
  </rule>
  <rule id="3">
    <other/>
  </rule>
</acl-list>
```

If this document is A1, and the watcher is sip:user3@example.com, the associated rule R1 is 2. If the watcher is sip:user1@example.com or sip:user2@example.com, the rule R1 is 1. If the watcher is anyone else from example.com, such as sip:user4@example.com, the rule R1 is 3.

If all Ri are equal, denote R = Ri. Thus, R is the rule ID associated with this watcher. Normally, all Ri will be equal. However, during transient periods of changes in authorization state, it is possible that inconsistent ACL documents exist. In that case, R is assigned the value Ri from the ACL Ai which is the most recently received among all ACL.
5.5. XML Schema

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid XML Studio 1.0.6.0 (http://www.liquid-technologies.com) -->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="acl-list">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:element name="rule">
          <xs:complexType>
            <xs:choice>
              <xs:element name="other" />
              <xs:sequence minOccurs="1" maxOccurs="unbounded">
                <xs:element name="member" type="xs:anyURI" />
              </xs:sequence>
            </xs:choice>
            <xs:attribute name="id" type="xs:integer" use="required" />
            <xs:attribute default="false" name="blocked" type="xs:boolean" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

6. Performance Analysis

This section considers the performance improvement of the mechanism when it is maximally exercised. In that case, the full ACL, including blocked senders, is returned in the first subscription to a presentity. This analysis assumes there is a single, monolithic presence server serving each domain.

The optimizations improve ramp-up, steady state, and termination message loads. In particular, each of those loads, without the optimization described here, is proportional to \( C_04 \), the total number of federated presentities per watcher. If we assume symmetry, such that the number of federated presentities per watcher is equal to the number of watchers per federated presentity, then each of the load figures is reduced by \( C_04 \). That is, the system behaves identically to the case where there is a single watcher per federated presentity, and assuming symmetric, the same as if there is a single federated presentity per watcher - e.g., \( C_04 = 1 \).

Consider then the very large network peering model in...
In this model, the assumption is two large peering domains with 20 million users each, with a value of 10 for C04. With this optimization, the number of steady state notifications due to presence state changes drops from 18.4 billion per day to 1.84 billion per day. The number of messages per second overall is reduced from 654,167 per second to 65,417 per second. Still a big number, of course, but it can’t actually get much smaller.

Indeed, it can be readily shown that, assuming the federated domains do not actually share raw presence inputs and the actual policies that govern operation of their servers, no protocol can do better (constants, such as message size and the need for protocol responses and acknowledgements aside). Consider a domain with N presentities. Each presentity changes state P times per hour. Every time the state changes, the domain applies its authorization and composition policies. The resulting presence document cannot be known to the watching domain. Thus, there must be at least one message from the serving to watching domain, per view, in order to inform it of that view. This means that the steady state rate of messages can never be better than N*P, and this is exactly the factor governing the rate of messages when this optimization is applied.

7. Requirements Analysis

This section analyzes the requirements in [I-D.ietf-sipping-presence-scaling-requirements] to show how they are met by the mechanism proposed here.

REQ-001: The solution should not hinder the ability of existing SIMPLE clients and/or servers from peering with a domain or client implementing the solution. No changes may be required of existing servers to interoperate. This requirement is met by usage of the Supported and Require mechanisms and SIP which negotiate its usage.

REQ-002: It does NOT constrain any existing RFC functional or security requirements for presence. The mechanism does not change anything that is possible without it. It does, however, introduce new privacy considerations, described below in Section 8.

REQ-003: Systems that are not using the new additions to the protocol should operate at the same level as they do today. This requirement is met by usage of the Supported and Require mechanisms in SIP.
REQ-004: The solution does not limit the ability for presentities to present different views of presence to different watchers. This requirement is met by usage of the ACL document, which allows the serving domain to associate a watcher with any view it likes, and to change it over time.

REQ-005: The solution does not restrict the ability of a presentity to obtain its list of watchers. The mechanism does allow a presence server to know the list of watchers, at the expense of non-optimal performance. In particular, it will receive a subscription from each watcher. However, it only generates one notification per view on presence changes. The fully optimized solution will result in a loss of knowledge of the set of watchers. However, it is a policy decision at the presence agent about whether it would like to make this tradeoff.

REQ-006: The solution MUST NOT create any new or make worse any existing privacy holes. This requirement is met, but only when carefully provisioned. See Section 8.

REQ-007: It is highly desirable for any presence system (intra or inter-domain) to scale linearly as number of watchers and presentities increase linearly. When the most optimal technique is used, there is always one subscription per view per presentity, independent of the number of watchers in the remote domain or the number of averages buddies per buddy list. Since the number of views is not proportional to the number of users, the total traffic volume in a domain is linear with its number of presentities, and is independent of the number of users in the peering domain.

REQ-008: The solution SHOULD NOT require significantly more state in order to implement the solution. The mechanism requires storage of the ACL, which has a size exactly equal to the number of subscriptions that would be required if the extension were not in place. Thus the memory usage is not worsened compared to the baseline.

REQ-009: It MUST be able to scale to tens of millions of concurrent users in each domain and in each peer domain. The analysis in Section 6 shows that, when fully utilized, this mechanism is the best that can possibly be achieved in any system that does not actually share policies and raw presence data.

REQ-010: It MUST support a very high level of watcher/presentity intersections in various intersection models. The mechanism is optimized for this case.
REQ-011: Protocol changes MUST NOT prohibit optimizations in different deployment models esp. where there is a high level of cross subscriptions between the domains. Since standard SIP techniques are utilized to negotiate the extension, other mechanisms can be defined in the future.

REQ-012: New functionalities and extensions to the presence protocol SHOULD take into account scalability with respect to the number of messages, state size and management and processing load. That is exactly what this extension targets.

REQ-013: The solution SHOULD allow for arbitrary federation topologies including direct peering and intermediary routing. The mechanism is optimized for direct peering. It can work in intermediary routing cases as well.

8. Security Considerations

The principal question with the specification is whether it alters the privacy characteristics of a non-optimized federated system.

Consider first the case where the serving domain is using the minimal trust model. In that case, the ACL provided to the watching information does not carry any information that the watching domain doesn’t already know. It merely points out when two watchers share the same view. This is something that the watching domain could have already ascertained by comparing presence documents delivered to each watcher. The ACL makes this task easier, but nonetheless the watching domain could have already ascertained it. Consequently, there is no change whatsoever in the level of privacy afforded by the optimization when this mode is used.

However, when an ACL is provided that includes other users besides the actual watcher, this provides additional information to the watching domain. This is, however, information that the watching domain could find out anyway. If it generated a subscription from each of its users to the presentity it would be able to determine who from its domain is allowed to subscribe and what view they would receive. This would be an expensive operation to be sure, but it is possible. Consequently, the optimization doesn’t really provide anything new to the originating domain, even in this case.

However, there is an attack possible when the information is divulged to an end user. Consider a watching domain that doesn’t actually implement this extension at all. A user within the domain uses a client that generates a subscription to a presentity in a remote domain. This subscription uses an outbound proxy in the watching
domain. The outbound proxy is just a proxy, and therefore doesn’t remove or modify the Supported header field in the request. The serving domain accepts the subscription and sends an ACL that contains the full set of watchers that are permitted in the originating domain. The original watcher now knows the set of other authorized buddies within their own domain, and what views they will see. While this is information that the domain overall would have access to, it is not information an end user would normally have access to. Consequently, this is a more serious privacy violation.

It is for this reason that this specification requires that both sides of the federated link be explicitly provisioned to utilize this optimization. In the attack above, the watching domain would not have set up a peering relationship with the serving domain. If it had, it would have an RLS and would not have permitted the user to directly subscribe in this way. Thus, when the subscription is received by the serving domain, it will find that it has no agreement with the originating domain, and would not utilize view sharing. This thwarts the attack.

This remedy is not optimal because it requires on provisioning to prevent. There does not appear to be any easy cryptographic means to prevent it, however.

In all cases, view sharing requires secure authentication and encryption between the domains that use it. This is provided by TLS.

9. IANA Considerations

There are several IANA considerations associated with this specification.

9.1. MIME Type Registration

This specification requests the registration of a new MIME type according to the procedures of RFC 2048 [RFC2048] and guidelines in RFC 3023 [RFC3023].

MIME media type name: application

MIME subtype name: aclinfo+xml

Mandatory parameters: none
Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [RFC3023].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [RFC3023].

Security considerations: See Section 10 of RFC 3023 [RFC3023] and Section 8 of RFC XXXX [[NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number of this specification]].

Interoperability considerations: none.

Published specification: RFC XXXX [[NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number of this specification]]

Applications which use this media type: This document type has been used to support subscriptions to lists of users [RFC4662] for SIP-based presence [RFC3856].

Additional Information:

Magic Number: None

File Extension: .acl

Macintosh file type code: "TEXT"

Personal and email address for further information: Jonathan Rosenberg, jdrosen@jdrosen.net

Intended usage: COMMON

Author/Change controller: The IETF.

9.2.  URN Sub-Namespace Registration

This section registers a new XML namespace, as per the guidelines in RFC 3688 [RFC3688].

URI: The URI for this namespace is urn:ietf:params:xml:ns:aclinfo.

Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).
9.3. Schema Registration

This section registers an XML schema per the procedures in [RFC3688].


Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org),
Jonathan Rosenberg (jdrosen@jdrosen.net).

The XML for this schema can be found as the sole content of
Section 5.5.

10. Acknowledgements

The authors would like to thank Avshalom Houri and Michael Froman for
their comments on this document.

11. References

11.1. Normative References

Initiation Protocol (SIP) Event Notification Extension for

[RFC4474]  Peterson, J. and C. Jennings, "Enhancements for
Authenticated Identity Management in the Session Initiation Protocol (SIP)


11.2. Informative References


[I-D.ietf-simple-interdomain-scaling-analysis]

[I-D.ietf-sip-subnot-etags]
draft-ietf-sip-subnot-etags-02 (work in progress), February 2008.

[I-D.ietf-sipping-presence-scaling-requirements]
Houri, A., Parameswar, S., Aoki, E., Singh, V., and H. Schulzrinne, "Scaling Requirements for Presence in SIP/SIMPLE",
draft-ietf-sipping-presence-scaling-requirements-00 (work in progress), February 2008.

Authors’ Addresses

Jonathan Rosenberg
Cisco
Edison, NJ
US

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

Steve Donovan
Cisco
Richardson, TX
US

Email: stdonova@cisco.com

Kathleen McMurry
Cisco
Richardson, TX
US

Email: kmcmurry@cisco.com