Network Working Group
Request for Comments: 3856
Category: Standards Track

J. Rosenberg
dynamicsoft

August 2004

A Presence Event Package for the Session Initiation Protocol (SIP)

Status of this Memo

Copyright Notice

Abstract

This document describes the usage of the Session Initiation Protocol (SIP) for subscriptions and notifications of presence. Presence is defined as the willingness and ability of a user to communicate with other users on the network. Historically, presence has been limited to "on-line" and "off-line" indicators; the notion of presence here is broader. Subscriptions and notifications of presence are supported by defining an event package within the general SIP event notification framework. This protocol is also compliant with the Common Presence Profile (CPP) framework.

Table of Contents

1. Introduction

Presence, also known as presence information, conveys the ability and willingness of a user to communicate across a set of devices. RFC 2778 [10] defines a model and terminology for describing systems that provide presence information. In that model, a presence service is a system that accepts, stores, and distributes presence information to interested parties, called watchers. A presence protocol is a protocol for providing a presence service over the Internet or any IP network.

This document proposes the usage of the Session Initiation Protocol (SIP) [1] as a presence protocol. This is accomplished through a concrete instantiation of the general event notification framework defined for SIP [2], and as such, makes use of the SUBSCRIBE and NOTIFY methods defined there. Specifically, this document defines an event package, as described in RFC 3265 [2]. SIP is particularly well suited as a presence protocol. SIP location services already contain presence information, in the form of registrations. Furthermore, SIP networks are capable of routing requests from any user on the network to the server that holds the registration state for a user. As this state is a key component of user presence, those

SIP networks can allow SUBSCRIBE requests to be routed to the same server. This means that SIP networks can be reused to establish global connectivity for presence subscriptions and notifications.

This event package is based on the concept of a presence agent, which is a new logical entity that is capable of accepting subscriptions, storing subscription state, and generating notifications when there are changes in presence. The entity is defined as a logical one, since it is generally co-resident with another entity.

This event package is also compliant with the Common Presence Profile (CPP) framework that has been defined in [3]. This allows SIP for presence to easily interwork with other presence systems compliant to CPP.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [4] and indicate requirement levels for compliant implementations.

3. Definitions

This document uses the terms as defined in RFC 2778 [10]. Additionally, the following terms are defined and/or additionally clarified:

Presence User Agent (PUA): A Presence User Agent manipulates presence information for a presentity. This manipulation can be the side effect of some other action (such as sending a SIP REGISTER request to add a new Contact) or can be done explicitly through the publication of presence documents. We explicitly allow multiple PUAs per presentity. This means that a user can have many devices (such as a cell phone and Personal Digital Assistant (PDA)), each of which is independently generating a component of the overall presence information for a presentity. PUAs push data into the presence system, but are outside of it, in that they do not receive SUBSCRIBE messages or send NOTIFY messages.

Presence Agent (PA): A presence agent is a SIP user agent which is capable of receiving SUBSCRIBE requests, responding to them, and generating notifications of changes in presence state. A presence agent must have knowledge of the presence state of a presentity. This means that it must have access to presence data manipulated by PUAs for the presentity. One way to do this is by co-locating the PA with the proxy/registrar.

Another way is to co-locate it with the presence user agent of the presentity. However, these are not the only ways, and this specification makes no recommendations about where the PA function should be located. A PA is always addressable with a SIP URI that uniquely identifies the presentity (i.e., sip:joe@example.com). There can be multiple PAs for a particular presentity, each of which handles some subset of the total subscriptions currently active for the presentity. A PA is also a notifier (defined in RFC 3265 [2]) that supports the presence event package.

Presence Server: A presence server is a physical entity that can act as either a presence agent or as a proxy server for SUBSCRIBE requests. When acting as a PA, it is aware of the presence information of the presentity through some protocol means. When acting as a proxy, the SUBSCRIBE requests are proxied to another entity that may act as a PA.

Edge Presence Server: An edge presence server is a presence agent that is co-located with a PUA. It is aware of the presence information of the presentity because it is co-located with the entity that manipulates this presence information.

4. Overview of Operation

In this section, we present an overview of the operation of this event package. The overview describes behavior that is documented in part here, in part within the SIP event framework [2], and in part in the SIP specification [1], in order to provide clarity on this package for readers only casually familiar with those specifications. However, the detailed semantics of this package require the reader to be familiar with SIP events and the SIP specification itself.

When an entity, the subscriber, wishes to learn about presence information from some user, it creates a SUBSCRIBE request. This request identifies the desired presentity in the Request-URI, using a SIP URI, SIPS URI [1] or a presence (pres) URI [3]. The SUBSCRIBE request is carried along SIP proxies as any other SIP request would be. In most cases, it eventually arrives at a presence server, which can either generate a response to the request (in which case it acts as the presence agent for the presentity), or proxy it on to an edge presence server. If the edge presence server handles the subscription, it is acting as the presence agent for the presentity. The decision at a presence server about whether to proxy or terminate the SUBSCRIBE is a local matter; however, we describe one way to effect such a configuration, using REGISTER.

The presence agent (whether in the presence server or edge presence server) first authenticates the subscription, then authorizes it. The means for authorization are outside the scope of this protocol, and we expect that many mechanisms will be used. If authorized, a 200 OK response is returned. If authorization could not be obtained at this time, the subscription is considered "pending", and a 202 response is returned. In both cases, the PA sends an immediate NOTIFY message containing the state of the presentity and of the subscription. The presentity state may be bogus in the case of a pending subscription, indicating offline no matter what the actual state of the presentity, for example. This is to protect the privacy of the presentity, who may not want to reveal that they have not provided authorization for the subscriber. As the state of the presentity changes, the PA generates NOTIFYs containing those state changes to all subscribers with authorized subscriptions. Changes in the state of the subscription itself can also trigger NOTIFY requests; that state is carried in the Subscription-State header field of the NOTIFY, and would typically indicate whether the subscription is active or pending.

The SUBSCRIBE message establishes a "dialog" with the presence agent. A dialog is defined in RFC 3261 [1], and it represents the SIP state between a pair of entities to facilitate peer-to-peer message exchanges. This state includes the sequence numbers for messages in both directions (SUBSCRIBE from the subscriber, NOTIFY from the presence agent), in addition to a route set and remote target URI. The route set is a list of SIP (or SIPS) URIs which identify SIP proxy servers that are to be visited along the path of SUBSCRIBE refreshes or NOTIFY requests. The remote target URI is the SIP or SIPS URI that identifies the target of the message - the subscriber, in the case of NOTIFY, or the presence agent, in the case of a SUBSCRIBE refresh.

SIP provides a procedure called record-routing that allows for proxy servers to request to be on the path of NOTIFY messages and SUBSCRIBE refreshes. This is accomplished by inserting a URI into the Record-Route header field in the initial SUBSCRIBE request.

The subscription persists for a duration that is negotiated as part of the initial SUBSCRIBE. The subscriber will need to refresh the subscription before its expiration, if they wish to retain the subscription. This is accomplished by sending a SUBSCRIBE refresh within the same dialog established by the initial SUBSCRIBE. This SUBSCRIBE is nearly identical to the initial one, but contains a tag in the To header field, a higher CSeq header field value, and possibly a set of Route header field values that identify the path of proxies the request is to take.

The subscriber can terminate the subscription by sending a SUBSCRIBE, within the dialog, with an Expires header field (which indicates duration of the subscription) value of zero. This causes an immediate termination of the subscription. A NOTIFY request is then generated by the presence agent with the most recent state. In fact, behavior of the presence agent for handling a SUBSCRIBE request with Expires of zero is no different than for any other expiration value; pending or authorized SUBSCRIBE requests result in a triggered NOTIFY with the current presentity and subscription state.

The presence agent can terminate the subscription at any time. To do so, it sends a NOTIFY request with a Subscription-State header field indicating that the subscription has been terminated. A reason parameter can be supplied which provides the reason.

It is also possible to fetch the current presence state, resulting in a one-time notification containing the current state. This is accomplished by sending a SUBSCRIBE request with an immediate expiration.

5. Usage of Presence URIs

A presentity is identified in the most general way through a presence URI [3], which is of the form pres:user@domain. These URIs are resolved to protocol specific URIs, such as the SIP or SIPS URI, through domain-specific mapping policies maintained on a server.

It is very possible that a user will have both a SIP (and/or SIPS) URI and a pres URI to identify both themself and other users. This leads to questions about how these URI relate and which are to be used.

In some instances, a user starts with one URI format, such as the pres URI, and learns a URI in a different format through some protocol means. As an example, a SUBSCRIBE request sent to a pres URI will result in learning a SIP or SIPS URI for the presentity from the Contact header field of the 200 OK to the SUBSCRIBE request. As another example, a DNS mechanism might be defined that would allow lookup of a pres URI to obtain a SIP or SIPS URI. In cases where one URI is learned from another through protocol means, those means will often provide some kind of scoping that limit the lifetime of the learned URI. DNS, for example, provides a TTL which would limit the scope of the URI. These scopes are very useful to avoid stale or conflicting URIs for identifying the same resource. To ensure that a user can always determine whether a learned URI is still valid, it is RECOMMENDED that systems which provide lookup services for presence URIs have some kind of scoping mechanism.

If a subscriber is only aware of the protocol-independent pres URI for a presentity, it follows the procedures defined in [5]. These procedures will result in the placement of the pres URI in the Request-URI of the SIP request, followed by the usage of the DNS procedures defined in [5] to determine the host to send the SIP request to. Of course, a local outbound proxy may alternatively be used, as specified in RFC 3261 [1]. If the subscriber is aware of both the protocol-independent pres URI and the SIP or SIPS URI for the same presentity, and both are valid (as discussed above) it SHOULD use the pres URI format. Of course, if the subscriber only knows the SIP URI for the presentity, that URI is used, and standard RFC 3261 processing will occur. When the pres URI is used, any proxies along the path of the SUBSCRIBE request which do not understand the URI scheme will reject the request. As such, it is expected that many systems will be initially deployed that only provide users with a SIP URI.

SUBSCRIBE messages also contain logical identifiers that define the originator and recipient of the subscription (the To and From header fields). These headers can take either a pres or SIP URI. When the subscriber is aware of both a pres and SIP URI for its own identity, it SHOULD use the pres URI in the From header field. Similarly, when the subscriber is aware of both a pres and a SIP URI for the desired presentity, it SHOULD use the pres URI in the To header field.

The usage of the pres URI instead of the SIP URI within the SIP message supports interoperability through gateways to other CPP-compliant systems. It provides a protocol-independent form of identification which can be passed between systems. Without such an identity, gateways would be forced to map SIP URIs into the addressing format of other protocols. Generally, this is done by converting the SIP URI to the form <foreign-protocol-scheme>:<encoded SIP URI>@<gateway>. This is commonly done in email systems, and has many known problems. The usage of the pres URI is a SHOULD, and not a MUST, to allow for cases where it is known that there are no gateways present, or where the usage of the pres URI will cause interoperability problems with SIP components that do not support the pres URI.

The Contact, Record-Route and Route fields do not identify logical entities, but rather concrete ones used for SIP messaging. SIP [1] specifies rules for their construction.

6. Presence Event Package

The SIP event framework [2] defines a SIP extension for subscribing to, and receiving notifications of, events. It leaves the definition of many aspects of these events to concrete extensions, known as

event packages. This document qualifies as an event package. This section fills in the information required for all event packages by RFC 3265 [2].

6.1. Package Name

The name of this package is "presence". As specified in RFC 3265 [2], this value appears in the Event header field present in SUBSCRIBE and NOTIFY requests.

Example:

Event: presence

6.2. Event Package Parameters

The SIP event framework allows event packages to define additional parameters carried in the Event header field. This package, presence, does not define any additional parameters.

6.3. SUBSCRIBE Bodies

A SUBSCRIBE request MAY contain a body. The purpose of the body depends on its type. Subscriptions will normally not contain bodies.

The Request-URI, which identifies the presentity, combined with the event package name, is sufficient for presence.

One type of body that can be included in a SUBSCRIBE request is a filter document. These filters request that only certain presence events generate notifications, or would ask for a restriction on the set of data returned in NOTIFY requests. For example, a presence filter might specify that the notifications should only be generated when the status of the user's instant inbox [10] changes. It might also say that the content of these notifications should only contain the status of the instant inbox. Filter documents are not specified in this document, and at the time of writing, are expected to be the subject of future standardization activity.

Honoring of these filters is at the policy discretion of the PA.

If the SUBSCRIBE request does not contain a filter, this tells the PA that no filter is to be applied. The PA SHOULD send NOTIFY requests at the discretion of its own policy.

6.4. Subscription Duration

User presence changes as a result of many events. Some examples are:

- Turning on and off of a cell phone
- Modifying the registration from a softphone
- Changing the status on an instant messaging tool

These events are usually triggered by human intervention, and occur with a frequency on the order of seconds to hours. As such, subscriptions should have an expiration in the middle of this range, which is roughly one hour. Therefore, the default expiration time for subscriptions within this package is 3600 seconds. As per RFC 3265 [2], the subscriber MAY specify an alternate expiration in the Expires header field.

6.5. NOTIFY Bodies

As described in RFC 3265 [2], the NOTIFY message will contain bodies that describe the state of the subscribed resource. This body is in a format listed in the Accept header field of the SUBSCRIBE, or a package-specific default if the Accept header field was omitted from the SUBSCRIBE.

In this event package, the body of the notification contains a presence document. This document describes the presence of the presentity that was subscribed to. All subscribers and notifiers MUST support the "application/pidf+xml" presence data format described in [6]. The subscribe request MAY contain an Accept header field. If no such header field is present, it has a default value of "application/pidf+xml". If the header field is present, it MUST include "application/pidf+xml", and MAY include any other types capable of representing user presence.

6.6. Notifier Processing of SUBSCRIBE Requests

Based on the proxy routing procedures defined in the SIP specification, the SUBSCRIBE request will arrive at a presence agent (PA). This subsection defines package-specific processing at the PA of a SUBSCRIBE request. General processing rules for requests are covered in Section 8.2 of RFC 3261 [1], in addition to general SUBSCRIBE processing in RFC 3265 [2].

User presence is highly sensitive information. Because the implications of divulging presence information can be severe, strong requirements are imposed on the PA regarding subscription processing, especially related to authentication and authorization.

6.6.1. Authentication

A presence agent MUST authenticate all subscription requests. This authentication can be done using any of the mechanisms defined in RFC 3261 [1]. Note that digest is mandatory to implement, as specified in RFC 3261.

In single-domain systems, where the subscribers all have shared secrets with the PA, the combination of digest authentication over Transport Layer Security (TLS) [7] provides a secure and workable solution for authentication. This use case is described in Section 26.3.2.1 of RFC 3261 [1].

In inter-domain scenarios, establishing an authenticated identity of the subscriber is harder. It is anticipated that authentication will often be established through transitive trust. SIP mechanisms for network asserted identity can be applied to establish the identity of the subscriber [11].

A presentity MAY choose to represent itself with a SIPS URI. By "represent itself", it means that the user represented by the presentity hands out, on business cards, web pages, and so on, a SIPS URI for their presentity. The semantics associated with this URI, as described in RFC 3261 [1], require TLS usage on each hop between the subscriber and the server in the domain of the URI. This provides additional assurances (but no absolute guarantees) that identity has been verified at each hop.

Another mechanism for authentication is S/MIME. Its usage with SIP is described fully in RFC 3261 [1]. It provides an end-to-end authentication mechanism that can be used for a PA to establish the identity of the subscriber.

6.6.2. Authorization

Once authenticated, the PA makes an authorization decision. A PA MUST NOT accept a subscription unless authorization has been provided by the presentity. The means by which authorization are provided are outside the scope of this document. Authorization may have been provided ahead of time through access lists, perhaps specified in a web page. Authorization may have been provided by means of uploading of some kind of standardized access control list document. Back end authorization servers, such as a DIAMETER [12] server, can also be

used. It is also useful to be able to query the user for authorization following the receipt of a subscription request for which no authorization information has been provided. The "watcherinfo" event template package for SIP [8] defines a means by which a presentity can become aware that a user has attempted to subscribe to it, so that it can then provide an authorization decision.

Authorization decisions can be very complex. Ultimately, all authorization decisions can be mapped into one of three states: rejected, successful, and pending. Any subscription for which the client is authorized to receive information about some subset of presence state at some points in time is a successful subscription. Any subscription for which the client will never receive any information about any subset of the presence state is a rejected subscription. Any subscription for which it is not yet known whether it is successful or rejected is pending. Generally, a pending subscription occurs when the server cannot obtain authorization at the time of the subscription, but may be able to do so at a later time, perhaps when the presentity becomes available.

The appropriate response codes for conveying a successful, rejected, or pending subscription (200, 403 or 603, and 202, respectively) are described in RFC 3265 [2].

If the resource is not in a meaningful state, RFC 3265 [2] allows the body of the initial NOTIFY to be empty. In the case of presence, that NOTIFY MAY contain a presence document. This document would indicate whatever presence state the subscriber has been authorized to see; it is interpreted by the subscriber as the current presence state of the presentity. For pending subscriptions, the state of the presentity SHOULD include some kind of textual note that indicates a pending status.

Polite blocking, as described in [13], is possible by generating a 200 OK to the subscription even though it has been rejected (or marked pending). Of course, an immediate NOTIFY will still be sent. The contents of the presence document in such a NOTIFY are at the discretion of the implementor, but SHOULD be constructed in such a way as to not reveal to the subscriber that their request has actually been blocked. Typically, this is done by indicating "offline" or equivalent status for a single contact address.

6.7. Notifier Generation of NOTIFY Requests

RFC 3265 details the formatting and structure of NOTIFY messages. However, packages are mandated to provide detailed information on when to send a NOTIFY, how to compute the state of the resource, how

to generate neutral or fake state information, and whether state information is complete or partial. This section describes those details for the presence event package.

A PA MAY send a NOTIFY at any time. Typically, it will send one when the state of the presentity changes. The NOTIFY request MAY contain a body indicating the state of the presentity. The times at which the NOTIFY is sent for a particular subscriber, and the contents of the body within that notification, are subject to any rules specified by the authorization policy that governs the subscription. This protocol in no way limits the scope of such policies. As a baseline, a reasonable policy is to generate notifications when the state of any of the presence tuples changes. These notifications would contain the complete and current presence state of the presentity as known to the presence agent. Future extensions can be defined that allow a subscriber to request that the notifications contain changes in presence information only, rather than complete state.

In the case of a pending subscription, when final authorization is determined, a NOTIFY can be sent. If the result of the authorization decision was success, a NOTIFY SHOULD be sent and SHOULD contain a presence document with the current state of the presentity. If the subscription is rejected, a NOTIFY MAY be sent. As described in RFC 3265 [2], the Subscription-State header field indicates the state of the subscription.

The body of the NOTIFY MUST be sent using one of the types listed in the Accept header field in the most recent SUBSCRIBE request, or using the type "application/pidf+xml" if no Accept header field was present.

The means by which the PA learns the state of the presentity are also outside the scope of this recommendation. Registrations can provide a component of the presentity state. However, the means by which a PA uses registrations to construct a presence document are an implementation choice. If a PUA wishes to explicitly inform the presence agent of its presence state, it should explicitly publish the presence document (or its piece of it) rather than attempting to manipulate their registrations to achieve the desired result.

For reasons of privacy, it will frequently be necessary to encrypt the contents of the notifications. This can be accomplished using S/MIME. The encryption can be performed using the key of the subscriber as identified in the From field of the SUBSCRIBE request. Similarly, integrity of the notifications is important to subscribers. As such, the contents of the notifications MAY provide authentication and message integrity using S/MIME. Since the NOTIFY is generated by the presence server, which may not have access to the

key of the user represented by the presentity, it will frequently be the case that the NOTIFY is signed by a third party. It is RECOMMENDED that the signature be by an authority over the domain of the presentity. In other words, for a user pres:user@example.com, the signator of the NOTIFY SHOULD be the authority for example.com.

6.8. Subscriber Processing of NOTIFY Requests

RFC 3265 [2] leaves it to event packages to describe the process followed by the subscriber upon receipt of a NOTIFY request, including any logic required to form a coherent resource state.

In this specification, each NOTIFY contains either no presence document, or a document representing the complete and coherent state of the presentity. Within a dialog, the presence document in the NOTIFY request with the highest CSeq header field value is the current one. When no document is present in that NOTIFY, the presence document present in the NOTIFY with the next highest CSeq value is used. Extensions which specify the use of partial state for presentities will need to dictate how coherent state is achieved.

6.9. Handling of Forked Requests

RFC 3265 [2] requires each package to describe handling of forked SUBSCRIBE requests.

This specification only allows a single dialog to be constructed as a result of emitting an initial SUBSCRIBE request. This guarantees that only a single PA is generating notifications for a particular subscription to a particular presentity. The result of this is that a presentity can have multiple PAs active, but these should be homogeneous, so that each can generate the same set of notifications for the presentity. Supporting heterogeneous PAs, each of which generates notifications for a subset of the presence data, is complex and difficult to manage. Doing so would require the subscriber to act as the aggregator for presence data. This aggregation function can only reasonably be performed by agents representing the presentity. Therefore, if aggregation is needed, it MUST be done in a PA representing the presentity.

Section 4.4.9 of RFC 3265 [2] describes the processing that is required to guarantee the creation of a single dialog in response to a SUBSCRIBE request.

6.10. Rate of Notifications

RFC 3265 [2] requires each package to specify the maximum rate at which notifications can be sent.

A PA SHOULD NOT generate notifications for a single presentity at a rate of more than once every five seconds.

6.11. State Agents

RFC 3265 [2] requires each package to consider the role of state agents in the package, and if they are used, to specify how authentication and authorization are done.

State agents are core to this package. Whenever the PA is not co-located with the PUA for the presentity, the PA is acting as a state agent. It collects presence state from the PUA, and aggregates it into a presence document. Because there can be multiple PUA, a centralized state agent is needed to perform this aggregation. That is why state agents are fundamental to presence. Indeed, they have a specific term that describes them - a presence server.

6.11.1. Aggregation, Authentication, and Authorization

The means by which aggregation is done in the state agent is purely a matter of policy. The policy will typically combine the desires of the presentity along with the desires of the provider. This document in no way restricts the set of policies which may be applied.

However, there is clearly a need for the state agent to have access to the state of the presentity. This state is manipulated by the PUA. One way in which the state agent can obtain this state is to subscribe to it. As a result, if there were 5 PUA manipulating presence state for a single presentity, the state agent would generate 5 subscriptions, one to each PUA. For this mechanism to be effective, all PUA SHOULD be capable of acting as a PA for the state that they manipulate, and that they authorize subscriptions that can be authenticated as coming from the domain of the presentity.

The usage of state agents does not significantly alter the way in which authentication is done by the PA. Any of the SIP authentication mechanisms can be used by a state agent. However, digest authentication will require the state agent to be aware of the shared secret between the presentity and the subscriber. This will require some means to securely transfer the shared secrets from the presentity to the state agent.

The usage of state agents does, however, have a significant impact on authorization. As stated in Section 6.6, a PA is required to authorize all subscriptions. If no explicit authorization policy has been defined, the PA will need to query the user for authorization. In a presence edge server (where the PUA is co-located with the PUA), this is trivially accomplished. However, when state agents are used (i.e., a presence server), a means is needed to alert the user that an authorization decision is required. This is the reason for the watcherinfo event template-package [8]. All state agents SHOULD support the watcherinfo template-package.

6.11.2. Migration

On occasion, it makes sense for the PA function to migrate from one server to another. For example, for reasons of scale, the PA function may reside in the presence server when the PUA is not running, but when the PUA connects to the network, the PA migrates subscriptions to it in order to reduce state in the network. The mechanism for accomplishing the migration is described in Section 3.3.5 of RFC 3265 [2]. However, packages need to define under what conditions such a migration would take place.

A PA MAY choose to migrate subscriptions at any time, through configuration, or through dynamic means. The REGISTER request provides one dynamic means for a presence server to discover that the function can migrate to a PUA. Specifically, if a PUA wishes to indicate support for the PA function, it SHOULD use the callee capabilities specification [9] to indicate that it supports the SUBSCRIBE request method and the presence event package. The combination of these two define a PA. Of course, a presence server can always attempt a migration without these explicit hints. If it fails with either a 405 or 489 response code, the server knows that the PUA does not support the PA function. In this case, the server itself will need to act as a PA for that subscription request. Once such a failure has occurred, the server SHOULD NOT attempt further migrations to that PUA for the duration of its registration. However, to avoid the extra traffic generated by these failed requests, a presence server SHOULD support the callee capabilities extension.

Furthermore, indication of support for the SUBSCRIBE request and the presence event package is not sufficient for migration of subscriptions. A PA SHOULD NOT migrate the subscription if it is composing aggregated presence documents received from multiple PUA.

7. Learning Presence State

Presence information can be obtained by the PA in many ways. No specific mechanism is mandated by this specification. This section overviews some of the options, for informational purposes only.

7.1. Co-location

When the PA function is co-located with the PUA, presence is known directly by the PA.

7.2. REGISTER

A UA uses the SIP REGISTER method to inform the SIP network of its current communications addresses (i.e., Contact addresses). Multiple UA can independently register Contact addresses for the same address-of-record. This registration state represents an important piece of the overall presence information for a presentity. It is an indication of basic reachability for communications.

Usage of REGISTER information to construct presence is only possible if the PA has access to the registration database, and can be informed of changes to that database. One way to accomplish that is to co-locate the PA with the registrar.

The means by which registration state is converted into presence state is a matter of local policy, and beyond the scope of this specification. However, some general guidelines can be provided. The address-of-record in the registration (the To header field) identifies the presentity. Each registered Contact header field identifies a point of communications for that presentity, which can be modeled using a tuple. Note that the contact address in the tuple need not be the same as the registered contact address. Using an address-of-record instead allows subsequent communications from a watcher to pass through proxies. This is useful for policy processing on behalf of the presentity and the provider.

A PUA that uses registrations to manipulate presence state SHOULD make use of the SIP callee capabilities extension [9]. This allows the PUA to provide the PA with richer information about itself. For example, the presence of the methods parameter listing the method "MESSAGE" indicates support for instant messaging.

The q values from the Contact header field [1] can be used to establish relative priorities amongst the various communications addresses in the Contact header fields.

The usage of registrations to obtain presence information increases the requirements for authenticity and integrity of registrations. Therefore, REGISTER requests used by presence user agents MUST be authenticated.

7.3. Uploading Presence Documents

If a means exists to upload presence documents from PUA to the PA, the PA can act as an aggregator and redistributor of those documents. The PA, in this case, would take the presence documents received from each PUA for the same presentity, and merge the tuples across all of those PUA into a single presence document. Typically, this aggregation would be accomplished through administrator or user defined policies about how the aggregation should be done.

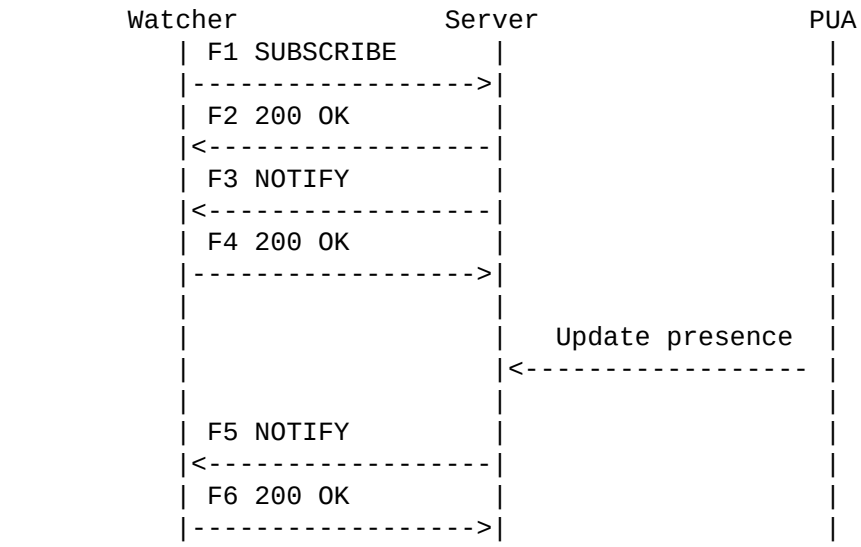
The specific means by which a presence document is uploaded to a presence agent are outside the scope of this specification. When a PUA wishes to have direct manipulation of the presence that is distributed to subscribers, direct uploading of presence documents is RECOMMENDED.

8. Example Message Flow

This message flow illustrates how the presence server can be responsible for sending notifications for a presentity. This flow assumes that the watcher has previously been authorized to subscribe to this resource at the server.

In this flow, the PUA informs the server about the updated presence information through some non-SIP means.

When the value of the Content-Length header field is "..." this means that the value should be whatever the computed length of the body is.



Message Details

F1 SUBSCRIBE watcher->example.com server

```
SUBSCRIBE sip:resource@example.com SIP/2.0
Via: SIP/2.0/TCP watcherhost.example.com;branch=z9hG4bKnashds7
To: <sip:resource@example.com>
From: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
CSeq: 17766 SUBSCRIBE
Max-Forwards: 70
Event: presence
Accept: application/pidf+xml
Contact: <sip:user@watcherhost.example.com>
Expires: 600
Content-Length: 0
```

F2 200 OK example.com server->watcher

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP watcherhost.example.com;branch=z9hG4bKnashds7
  ;received=192.0.2.1
To: <sip:resource@example.com>;tag=ffd2
From: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
CSeq: 17766 SUBSCRIBE
Expires: 600
Contact: sip:server.example.com
Content-Length: 0
```

F3 NOTIFY example.com server-> watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/TCP server.example.com;branch=z9hG4bKna998sk
From: <sip:resource@example.com>;tag=ffd2
To: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
Event: presence
Subscription-State: active;expires=599
Max-Forwards: 70
CSeq: 8775 NOTIFY
Contact: sip:server.example.com
Content-Type: application/pidf+xml
Content-Length: ...

[PIDF Document]
```

F4 200 OK watcher-> example.com server

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP server.example.com;branch=z9hG4bKna998sk
  ;received=192.0.2.2
From: <sip:resource@example.com>;tag=ffd2
To: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
CSeq: 8775 NOTIFY
Content-Length: 0
```

F5 NOTIFY example.com server -> watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/TCP server.example.com;branch=z9hG4bKna998sl
From: <sip:resource@example.com>;tag=ffd2
To: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
CSeq: 8776 NOTIFY
Event: presence
Subscription-State: active;expires=543
Max-Forwards: 70
Contact: sip:server.example.com
Content-Type: application/pidf+xml
Content-Length: ...

[New PIDF Document]
```

F6 200 OK

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP server.example.com;branch=z9hG4bKna998sl
 ;received=192.0.2.2
From: <sip:resource@example.com>;tag=ffd2
To: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@watcherhost.example.com
CSeq: 8776 NOTIFY
Content-Length: 0
```

9. Security Considerations

There are numerous security considerations for presence. RFC 2779 [13] outlines many of them, and they are discussed above. This section considers them issue by issue.

9.1. Confidentiality

Confidentiality encompasses many aspects of a presence system:

- Subscribers may not want to reveal the fact that they have subscribed to certain users
- Users may not want to reveal that they have accepted subscriptions from certain users
- Notifications (and fetch results) may contain sensitive data which should not be revealed to anyone but the subscriber

Confidentiality is provided through a combination of hop-by-hop encryption and end-to-end encryption. The hop-by-hop mechanisms provide scalable confidentiality services, disable attacks involving traffic analysis, and hide all aspects of presence messages. However, they operate based on transitivity of trust, and they cause message content to be revealed to proxies. The end-to-end mechanisms do not require transitivity of trust, and reveal information only to the desired recipient. However, end-to-end encryption cannot hide all information, and is susceptible to traffic analysis. Strong end-to-end authentication and encryption can be done using public keys, and end-to-end encryption can be done using private keys [14]. Both hop-by-hop and end-to-end mechanisms will likely be needed for complete privacy services.

SIP allows any hop by hop encryption scheme, but TLS is mandatory to implement for servers. Therefore, it is RECOMMENDED that TLS [7] be used between elements to provide this function. The details for usage of TLS for server-to-server and client-to-server security are detailed in Section 26.3.2 of RFC 3261 [1].

SIP encryption, using S/MIME, MAY be used end-to-end for the transmission of both SUBSCRIBE and NOTIFY requests.

9.2. Message Integrity and Authenticity

It is important for the message recipient to ensure that the message contents are actually what was sent by the originator, and that the recipient of the message be able to determine who the originator really is. This applies to both requests and responses of SUBSCRIBE and NOTIFY. NOTIFY requests are particularly important. Without authentication and integrity, presence documents could be forged or modified, fooling the watcher into believing incorrect presence information.

RFC 3261 provides many mechanisms to provide these features. In order for the PA to authenticate the watcher, it MAY use HTTP Digest (Section 22 of RFC 3261). As a result, all watchers MUST support HTTP Digest. This is a redundant requirement, however, since all SIP user agents are mandated to support it by RFC 3261. To provide authenticity and integrity services, a watcher MAY use the SIPS scheme when subscribing to the presentity. To support this, all PA MUST support TLS and SIPS as if they were a proxy (see Section 26.3.1 of RFC 3261).

Furthermore, SMIME MAY be used for integrity and authenticity of SUBSCRIBE and NOTIFY requests. This is described in Section 23 of RFC 3261.

9.3. Outbound Authentication

When local proxies are used for transmission of outbound messages, proxy authentication is RECOMMENDED. This is useful to verify the identity of the originator, and prevent spoofing and spamming at the originating network.

9.4. Replay Prevention

Replay attacks can be used by an attacker to fool a watcher into believing an outdated presence state for a presentity. For example, a document describing a presentity as being "offline" can be replayed, fooling watchers into thinking that the user is never online. This may effectively block communications with the presentity.

SIP S/MIME can provide message integrity and authentication over SIP request bodies. Watchers and PAs MAY implement S/MIME signatures to prevent these replay attacks. When it is used for that purpose, the presence document carried in the NOTIFY request MUST contain a timestamp. In the case of PIDF, this is accomplished using the timestamp element, as described in Section 6 of [6]. Tuples whose timestamp is older than the timestamp of the most recently received presence document SHOULD be considered stale, and discarded.

Finally, HTTP digest authentication (which MUST be implemented by watchers and PAs) MAY be used to prevent replay attacks, when there is a shared secret between the PA and the watcher. In such a case, the watcher can challenge the NOTIFY requests with the auth-int quality of protection.

9.5. Denial of Service Attacks Against Third Parties

Denial of Service (DOS) attacks are a critical problem for an open, inter-domain, presence protocol. Unfortunately, presence is a good candidate for Distributed DoS (DDOS) attacks because of its amplification properties. A single SUBSCRIBE message could generate a nearly unending stream of notifications, so long as a suitably dynamic source of presence data can be found. Thus, a simple way to launch an attack against a target is to send subscriptions to a large number of users, and in the Contact header field (which is where notifications are sent), place the address of the target. RFC 3265 provides some mechanisms to mitigate these attacks [2]. If a NOTIFY is not acknowledged or was not wanted, the subscription that generated it is removed. This eliminates the amplification properties of providing false Contact addresses.

Authentication and authorization at the PA can also prevent these attacks. Typically, authorization policy will not allow subscriptions from unknown watchers. If the attacks are launched from watchers unknown to the presentity (a common case), the attacks are mitigated.

9.6. Denial Of Service Attacks Against Servers

Denial of service attacks can also be launched against a presence agent itself, in order to disrupt service to a community of users. SIP itself, along with RFC 3265 [2], describes several mechanisms to mitigate these attacks.

A server can prevent SYN-attack style attacks through a four-way handshake using digest authentication [1]. Even if the server does not have a shared secret with the client, it can verify the source IP address of the request using the "anonymous" user mechanism described in Section 22.1 of RFC 3261 [1]. SIP also allows a server to instruct a client to back-off from sending it requests, using the 503 response code (Section 21.5.4 of RFC 3261 [1]). This can be used to fend off floods of SUBSCRIBE requests launched as a result of a distributed denial of service attack.

10. IANA Considerations

This specification registers an event package, based on the registration procedures defined in RFC 3265 [2]. The following is the information required for such a registration:

Package Name: presence

Package or Template-Package: This is a package.

Published Document: RFC 3856

Person to Contact: Jonathan Rosenberg, jdrosen@jdrosen.net.

11. Contributors

The following individuals were part of the initial team that worked through the technical design of this specification:

Jonathan Lennox
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003

EMail: lennox@cs.columbia.edu

Robert Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024

EMail: rsparks@dynamicsoft.com

Ben Campbell

EMail: ben@nostrum.com

Dean Willis
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024

EMail: dwillis@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003

EMail: schulzrinne@cs.columbia.edu

Christian Huitema
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

EMail: huitema@microsoft.com

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

EMail: bernarda@microsoft.com

David Gurle
Reuters Corporation

EMail: David.Gurle@reuters.com

David Oran
Cisco Systems
170 West Tasman Dr.
San Jose, CA 95134

EMail: oran@cisco.com

12. Acknowledgements

We would like to thank Rick Workman, Adam Roach, Sean Olson, Billy Biggs, Stuart Barkley, Mauricio Arango, Richard Shockey, Jorgen Bjorkner, Henry Sinnreich, Ronald Akers, Paul Kyzivat, Ya-Ching Tan, Patrik Faltstrom, Allison Mankin and Hisham Khartabil for their comments and support of this specification.

13. Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, H., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.

[4] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Peterson, J., "Address Resolution for Instant Messaging and Presence", RFC 3861, August 2004.

[6] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.

[7] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[8] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", RFC 3857, August 2004.

[9] Schulzrinne, H. Rosenberg, J., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

14. Informative References

[10] Day, M., Rosenberg, J., and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[11] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", Work in Progress, May 2004.

[12] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[13] Day, M., Aggarwal, S., Mohr, G., and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.

[14] Gutmann, P., "Password-Based Encryption for CMS", RFC 3211, December 2001.

15. Author’s Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054

EMail: jdrosen@dynamicsoft.com

16. Full Copyright Statement

Intellectual Property

Acknowledgement

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