Internet Engineering Task Force
Internet Draft
draft-rosenberg-impp-presence-01.txt
March 2, 2001
Expires: September 2001

SIMPLE WG
Rosenberg et al.
Various Places

SIP Extensions for Presence

STATUS OF THIS MEMO

Abstract

This document proposes an extension to SIP for subscriptions and notifications of user presence. User 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 user presence are supported by defining an event package within the general SIP event notification framework. This protocol is also compliant with the Common Presence and Instant Messaging (CPIM) framework.

1 Introduction

Presence is (indirectly) defined in RFC2778 [1] as subscription to and notification of changes in the communications state of a user.

This communications state consists of the set of communications means, communications address, and status of that user. A presence protocol is a protocol for providing such a service over the Internet or any IP network.

This document proposes an extension to the Session Initiation Protocol (SIP) [2] for presence. This extension is a concrete instantiation of the general event notification framework defined for SIP [3], and as such, makes use of the SUBSCRIBE and NOTIFY methods defined there. User presence is particularly well suited for SIP. SIP registrars and location services already hold user presence information; it is uploaded to these devices through REGISTER messages, and used to route calls to those users. Furthermore, SIP networks already route INVITE messages from any user on the network to the proxy that holds the registration state for a user. As this state is user presence, those SIP networks can also allow SUBSCRIBE requests to be routed to the same proxy. This means that SIP networks can be reused to establish global connectivity for presence subscriptions and notifications.

This extension 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 user presence. The entity is defined as a logical one, since it is generally co-resident with another entity, and can even move around during the lifetime of a subscription.

This extension is also compliant with the Common Presence and Instant Messaging (CPIM) framework that has been defined in [4]. This allows SIP for presence to easily interwork with other presence systems compliant to CPIM.

2 Definitions

This document uses the terms as defined in [1]. Additionally, the following terms are defined and/or additionally clarified:

Presence User Agent (PUA): A Presence User Agent manipulates presence information for a presentity. In SIP terms, this means that a PUA generates REGISTER requests, conveying some kind of information about the presentity. We explicitly allow multiple PUAs per presentity. This means that a user can have many devices (such as a cell phone and 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.

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 complete knowledge of the presence state of a presentity. Typically, this is accomplished by co-locating the PA with the proxy/registrar, or the presence user agent of the presentity. A PA is always addressable with a SIP URL.

Presence Server: A presence server is a logical 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. This protocol means can be SIP REGISTER requests, but other mechanisms are allowed. When acting as a proxy, the SUBSCRIBE requests are proxied to another entity that may act as a PA.

Presence Client: A presence client is a presence agent that is colocated 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.

3 Overview of Operation

In this section, we present an overview of the operation of this extension.

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 either a presence URL or a SIP URL. The subscription is carried along SIP proxies as any other INVITE would be. It eventually arrives at a presence server, which can either terminate the subscription (in which case it acts as the presence agent for the presentity), or proxy it on to a presence client. If the presence client handles the subscription, it is effectively acting as the presence agent for the presentity. The decision 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 presence client) 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. Once authorized, the presence agent sends a 202 Accepted response. It also sends an immediate NOTIFY message containing the state of the presentity. As the state of the presentity changes, the PA generates NOTIFYs for all

subscribers.

The SUBSCRIBE message effectively establishes a session with the presence agent. As a result, the SUBSCRIBE can be record-routed, and rules for tag handling and Contact processing mirror those for INVITE. Similarly, the NOTIFY message is handled in much the same way a re-INVITE within a call leg is handled.

4 Naming

A presentity is identified in the most general way through a presence URI [4], which is of the form pres:user@domain. These URIs are protocol independent. Through a variety of means, these URIs can be resolved to determine a specific protocol that can be used to access the presentity. Once such a resolution has taken place, the presentity can be addressed with a sip URL of nearly identical form: sip:user@domain. The protocol independent form (the pres: URL) can be thought of as an abstract name, akin to a URN, which is used to identify elements in a presence system. These are resolved to concrete URLs that can be used to directly locate those entities on the network.

When subscribing to a presentity, the subscription can be addressed using the protocol independent form or the sip URL form. In the SIP context, "addressed" refers to the request URI. It is RECOMMENDED that if the entity sending a SUBSCRIBE is capable of resolving the protocol independent form to the SIP form, this resolution is done before sending the request. However, if the entity is incapable of doing this translation, the protocol independent form is used in the request URI. Performing the translation as early as possible means that these requests can be routed by SIP proxies that are not aware of the presence namespace.

The result of this naming scheme is that a SUBSCRIBE request is addressed to a user the exact same way an INVITE request would be addressed. This means that the SIP network will route these messages along the same path an INVITE would travel. One of these entities along the path may act as a PA for the subscription. Typically, this will either be the presence server (which is the proxy/registrar where that user is registered), or the presence client (which is one of the user agents associated with that presentity).

SUBSCRIBE messages also contain logical identifiers that define the originator and recipient of the subscription (the To and From header fields). Since these identifiers are logical ones, it is RECOMMENDED that these use the protocol independent format whenever possible. This also makes it easier to interwork with other systems which recognize these forms.

The Contact, Record-Route and Route fields do not identify logical entities, but rather concrete ones used for SIP messaging. As such, they MUST use the SIP URL forms in both SUBSCRIBE and NOTIFY.

5 Presence Event Package

The SIP event framework [3] defines an abstract SIP extension for subscribing to, and receiving notifications of, events. It leaves the definition of many additional aspects of these events to concrete extensions, also known as event packages. This extension qualifies as an event package. This section fills in the information required by [3].

5.1 Package Name

The name of this package is "presence". This name MUST appear within the Event header in SUBSCRIBE request and NOTIFY request. This section also serves as the IANA registration for the event package "presence".

TODO: Define IANA template in sub-notify and fill it in here.

Example:

Event: presence

5.2 SUBSCRIBE bodies

The body of a SUBSCRIBE request MAY contain a body. The purpose of the body depends on its type. In general, subscriptions will normally not contain bodies. The request URI, which identifies the presentity, combined with the event package name, are sufficient for user presence.

We anticipate that document formats could be defined to act as filters for subscriptions. These filters would indicate certain user presence events that would generate notifies, or restrict 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 users instant message inbox changes. It might also say that the content of these notifications should only contain the IM related information.

5.3 Expiration

User presence changes as a result of events that include:

- 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 minutes or hours. As such, it is 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 [3], the subscriber MAY include an alternate expiration time. Whatever the indicated expiration time, the server MAY reduce it but MUST NOT increase it.

5.4 NOTIFY Bodies

The body of the notification contains a presence document. This document describes the user presence of the presentity that was subscribed to. All subscribers MUST support the presence data format described in [fill in with IMPP document TBD], and MUST list its MIME type, [fill in with MIME type] in an Accept header present in the SUBSCRIBE request.

Other presence data formats might be defined in the future. In that case, the subscriptions MAY indicate support for other presence formats. However, they MUST always support and list [fill in with MIME type of IMPP presence document] as an allowed format.

Of course, the notifications generated by the presence agent MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

5.5 Processing Requirements at the PA

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.

A presence agent MUST authenticate all subscription requests. This authentication can be done using any of the mechanisms defined for SIP. It is not considered sufficient for the authentication to be transitive; that is, the authentication SHOULD use an end-to-end mechanism. The SIP basic authentication mechanism MUST NOT be used.

It is RECOMMENDED that any subscriptions that are not authenticated do not cause state to be established in the PA. This can be accomplished by generating a 401 in response to the SUBSCRIBE, and then discarding all state for that transaction. Retransmissions of the SUBSCRIBE generate the same response, guaranteeing reliability even over UDP.

Furthermore, 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 [5], RADIUS [6], or COPS [7], 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 was present. Appendix A provides a possible solution for such a scenario.

The result of the authorization decision by the server will be reject, accept, or pending. Pending occurs when the server cannot obtain authorization at this time, and may be able to do so at a later time, when the presentity becomes available.

Unfortunately, if the server informs the subscriber that the subscription is pending, this will divulge information about the presentity - namely, that they have not granted authorization and are not available to give it at this time. Therefore, a PA SHOULD generate the same response for both pending and accepted subscriptions. This response SHOULD be a 202 Accepted response.

If the server informs the subscriber that the subscription is rejected, this also divulges information about the presentity - namely, that they have explicitly blocked the subscription previously, or are available at this time and chose to decline the subscription. If the policy of the server is not to divulge this information, the PA MAY respond with a 202 Accepted response even though the subscription is rejected. Alternatively, if the policy of the presentity or the PA is that it is acceptable to inform the subscriber of the rejection, a 603 Decline SHOULD be used.

Note that since the response to a subscription does not contain any useful information about the presentity, privacy and integrity of SUBSCRIBE responses is not deemed important.

5.6 Generation of Notifications

Upon acceptance of a subscription, the PA SHOULD generate an

immediate NOTIFY with the current presence state of the presentity.

If a subscription is received, and is marked as pending or was rejected, the PA SHOULD generate an immediate NOTIFY. This NOTIFY should contain a valid state for the presentity, yet be one which provides no useful information about the presentity. An example of this is to provide an IM URL that is the same form as the presence URL, and mark that IM address as "not available". The reason for this process of "lying" is that without it, a subscriber could tell the difference between a pending subscription and an accepted subscription based on the existence and content of an immediate NOTIFY. The approach defined here ensures that the presence delivered in a NOTIFY generated by a pending or rejected subscription is also a valid one that could have been delivered in a NOTIFY generated by an accepted subscription.

If the policy of the presence server or the presentity is that it is acceptable to divulge information about whether the subscription succeeded or not, the immediate NOTIFY need not be sent for pending or rejected subscriptions.

Of course, once a subscription is accepted, the PA SHOULD generate a NOTIFY for the subscription when it determines that the presence state of the presentity has changed. Section 6 describes how the PA makes this determination.

For reasons of privacy, it will frequently be necessary to encrypt the contents of the notifications. This can be accomplished using the standard SIP encryption mechanisms. The encryption should be performed using the key of the subscriber as identified in the From field of the SUBSCRIBE. Similarly, integrity of the notifications is important to subscribers. As such, the contents of the notifications SHOULD be authenticated using one of the standardized SIP mechanisms. Since the NOTIFY are 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 are signed by a third party. It is RECOMMENDED that the signature be by an authority over 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.

5.7 Rate Limitations on NOTIFY

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the PA not generate notifications for a single presentity at a rate faster than once every 5 seconds.

5.8 Refresh Behavior

Since SUBSCRIBE is routed by proxies as any other method, it is possible that a subscription might fork. The result is that it might arrive at multiple devices which are configured to act as a PA for the same presentity. Each of these will respond with a 202 response to the SUBSCRIBE. Based on the forking rules in SIP, only one of these responses is passed to the subscriber. However, the subscriber will receive notifications from each of those PA which accepted the subscriptions. The SIP event framework allows each package to define the handling for this case.

The processing in this case is identical to the way INVITE would be handled. The 202 Accepted to the SUBSCRIBE will result in the installation of subscription state in the subscriber. The subscription is associated with the To and From (both with tags) and Call-ID from the 202. When notifications arrive, those from the PA's whose 202's were discarded in the forking proxy will not match the subscription ID stored at the subscriber (the From tags will differ). These SHOULD be responded to with a 481. This will disable the subscriptions from those PA. Furthermore, when refreshing the subscription, the refresh SHOULD make use of the tags from the 202 and make use of any Contact or Record-Route headers in order to deliver the SUBSCRIBE back to the same PA that sent the 202.

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 generated notifications for a subset of the presence data, is complex and difficult to manage. If such a feature is needed, it can be accomplished with a B2BUA rather than through a forking proxy.

6 Publication

The user presence for a presentity can be obtained from any number of different ways. Baseline SIP defines a method that is used by all SIP clients - the REGISTER method. This method allows a UA to inform a SIP network of its current communications addresses (ie., Contact addresses) . Furthermore, multiple UA can independently register Contact addresses for the same SIP URL. These Contact addresses can be SIP URLs, or they can be any other valid URL.

Using the register information for presence is straightforward. The address of record in the REGISTER (the To field) identifies the presentity. The Contact headers define communications addresses that describe the state of the presentity. The use of the SIP caller preferences extension [8] is RECOMMENDED for use with UAs that are

interested in presence. It provides additional information about the Contact addresses that can be used to construct a richer presence document. The "description" attribute of the Contact header is explicitly defined here to be used as a free-form field that allows a user to define the status of the presentity at that communications address.

We also allow REGISTER requests to contain presence documents, so that the PUAs can publish more complex information.

Note that we do not provide for locking mechanisms, which would allow a client to lock presence state, fetch it, and update it atomically. We believe that this is not neeeded for the majority of use cases, and introduces substantial complexity. Most presence operations do not require get-before-set, since the SIP register mechanism works in such a way that data can be updated without a get.

The application of registered contacts to presence increases the requirements for authenticity. Therefore, REGISTER requests used by presence user agents SHOULD be authenticated using either SIP authentication mechanisms, or a hop by hop mechanism.

To indicate presence for instant messaging, the UA MAY either register contact addresses that are SIP URLs with the "methods" parameter set to indicate the method MESSAGE, or it MAY register an IM URL.

TODO: This section needs work. Need to define a concrete example of mapping a register to a presence document, once IMPP generates the document format.

6.1 Migrating the PA Function

It is important to realize that the PA function can be colocated with several elements:

- It can be co-located with the proxy server handling registrations for the presentity. In this way, the PA knows the presence of the user through registrations.

- It can be co-located with a PUA for that presentity. In the case of a single PUA per presentity, the PUA knows the state of the presentity by sheer nature of its co-location.

- It can be co-located in any proxy along the call setup path. That proxy can learn the presence state of the presentity by generating its own SUBSCRIBE in order to determine it. In this case, the PA is effectively a B2BUA.

Because of the soft-state nature of the subscriptions, it becomes possible for the PA function to migrate during the lifetime of a subscription. The most workable scenario is for the PA function to migrate from the presence server to the PUA, and back.

Consider a subscription that is installed in a presence server. Assume for the moment that the presence server can determine that a downstream UA is capable of acting as a PA for the presentity. When a subscription refresh arrives, the PA destroys its subscription, and then acts as a proxy for the subscription. The subscription is then routed to the UA, where it can be accepted. The result is that the subscription becomes installed in the PUA.

For this migration to work, the PUA MUST be prepared to accept SUBSCRIBE requests which already contain tags in the To field. Furthermore, the PUA MUST insert a Contact header into the 202, and this header MUST be used by the subscriber to update the contact address for the subscription.

TODO: Does this work? What about getting a Record-Route in place at the PUA. This might only be possible for refreshes that don't use Route or tags.

The presence server determines that a PUA is capable of supporting a PA function through the REGISTER message. Specifically, if a PUA wishes to indicate support for the PA function, it SHOULD include a contact address in its registration with a caller preferences "methods" parameter listing SUBSCRIBE.

7 Mapping to CPIM

This section defines how a SIP for presence messages are converted to CPIM, and how a CPIM messages are converted to SIP for presence. SIP to CPIM conversion occurs when a SIP system sends a SUBSCRIBE request that contains a pres URL or SIP URL that corresponds to a user in a domain that runs a different presence protocol. CPIM to SIP involves the case where a user in a different protocol domain generates a subscription that is destined for a user in a SIP domain.

Note that the process defined below requires that the gateway store subscription state. This unfortunate result is due to the need to remember the Call-ID, CSeq, and Route headers for subscriptions from the SIP side, so that they can be inserted into the SIP NOTIFY generated when a CPIM notification arrives.

7.1 SIP to CPIM

SIP for presnce is converted to CPIM through a SIP to CPIM abstract

gateway service, depicted in Figure 1.

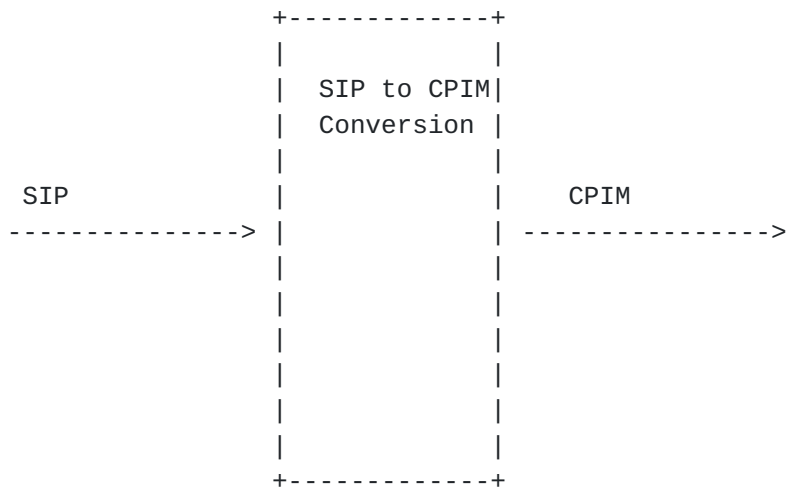


Figure 1: SIP to CPIM Conversion

The first step is that a SUBSCRIBE request is received at a gateway. The gateway generates a CPIM subscription request, with its parameters filled in as follows:

- The watcher identity in the CPIM message is copied from the From field of the SUBSCRIBE. If the From field contains a SIP URL, it is converted to an equivalent pres URL by dropping all SIP URL parameters, and changing the scheme to pres.

  This conversion may not work - what if the SIP URL has no user name. Plus, converting from a URL back to a URN in this fashion may not do it correctly.

- The target identity in the CPIM message is copied from the Request-URI field of the SUBSCRIBE. This may need to be converted to a pres URL as well.

- The duration parameter in the CPIM message is copied from the Expires header in the SUBSCRIBE. If the Expires header specifies an absolute time, it is converted to a delta-time by the gateway. If no Expires header is present, one hour is assumed.

- The transID parameter in the CPIM message is constructed by appending the Call-ID, the URI in the To field, the URI in the From field, the CSeq and the tag in the From field, and the request URI, and computing a hash of the resulting string. This hash is used as the transID. Note that the request URI is included in the hash. This is to differentiate forked requests within the SIP network that may arrive at the same gateway.

The CPIM service then responds with either a success or failure. In the case of success, the SIP to CPIM gateway service generates a 202 response to the SUBSCRIBE. It adds a tag to the To field in the response, which is the same as the transID field in the success response. The 202 response also contains a Contact header, which is the value of the target from the SUBSCRIBE request. It is important that the Contact header be set to the target, since that makes sure that subscription refreshes have the same value in the request URI as the original subscription. The duration value from the CPIM success response is placed into the Expires header of the 202. The gateway stores the Call-ID and Route header set for this subscription.

If the CPIM service responds with a failure, the SIP to CPIM gateway generates a 603 response. It adds a tag to the To field in the response, which is the same as the transID field in the failure response.

When the CPIM system generates a notification request, the SIP to CPIM gateway creates a SIP NOTIFY request. The request is constructed using the standard RFC2543 [2] procedures for constructing a request within a call leg. This will result in the To field containing the watcher field from CPIM, and the From field containing the target field from the CPIM notification. The tag in the From field will contain the transID. The presence information is copied into the body of the notification. The Call-ID and Route headers are constructed from the subscription state stored in the gateway. If no notification has yet been generated for this subscription, an initial CSeq value

is selected and stored.

SUBSCRIBE refreshes are handled identically to initial subscriptions as above.

If a subscription is received with an Expires of zero, the SIP to CPIM gateway generates an unsubscribe message into the the CPIM system. The watcher parameter is copied from the From field of the SUBSCRIBE. The target parameter is copied from the Request URI field of the SUBSCRIBE. The transID is copied from the tag in the To field of the SUBSCRIBE request.

The response to an unsubscribe is either success or failure. In the case of success, a 202 response is constructed in the same fashion as above for a success response to a CPIM subscriber. All subscription state is removed. In the case of failure, a 603 response is constructed in the same fashion as above, and then subscription state is removed, if present.

7.2 CPIM to SIP

CPIM to SIP conversion occurs when a CPIM subscription request arrives on the CPIM side of the gateway. This scenario is shown in Figure 2.

The CPIM subscription request is converted into a SIP SUBSCRIBE request. To do that, the first step is to determine if the subscribe is for an existing subscription. That is done by taking the target in the CPIM subscription request, and matching it against targets for existing subscriptions. If there are none, it is a new subscription, otherwise, its a refresh.

If its a new subscription, the gateway generates a SIP SUBSCRIBE request in the following manner:

- The From field in the request is set to the watcher field in the CPIM subscription request
- The To field in the request is set to the target field in the CPIM subscription request
- The Expires header in the SUBSCRIBE request is set to the duration field in the CPIM subscription request
- The tag in the From field is set to the transID in the CPIM subscription request.

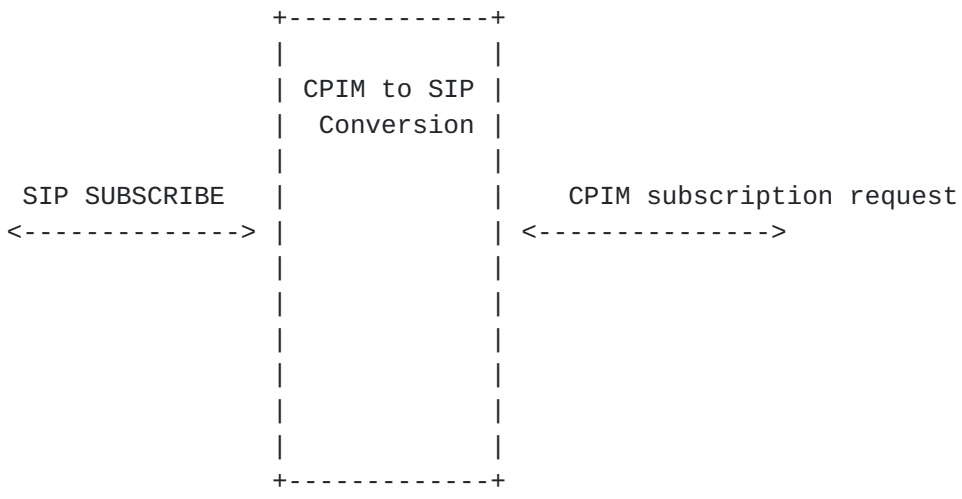


Figure 2: CPIM to SIP Conversion

This SUBSCRIBE message is then sent.

If the subscription is a refresh, a SUBSCRIBE request is generated in the same way. However, there will also be a tag in the To field, copied from the subscription state in the gateway, and a Route header, obtained from the subscription state in the gateway.

When a response to the SUBSCRIBE is received, a response is sent to the CPIM system. The duration parameter in this response is copied from the Expires header in the SUBSCRIBE response (a conversion from an absolute time to delta time may be needed). The transID in the response is copied from the tag in the From field of the response. If the response was 202, the status is set to indeterminate. If the response was any other 200 class response, the status is set to sucess. For any other final response, the status is set to failure.

If the response was a 200 class response, subscription state is

established. This state contains the tag from the To field in the SUBSCRIBE response, and the Route header set computed from the Record-Routes and Contact headers in the 200 class response. The subscription is indexed by the presentity identification (the To field of the SUBSCRIBE that was generated).

If an unsubscribe request is received from the CPIM side, the gateway checks if the subscription exists. If it does, a SUBSCRIBE is generated as described above. However, the Expires header is set to zero. If the subscription does not exist, the gateway generates a failure response and sends it to the CPIM system. When the response to the SUBSCRIBE request arrives, it is converted to a CPIM response as described above for the initial SUBSCRIBE response. In all cases, any subscription state in the gateway is destroyed.

When a NOTIFY is received from the SIP system, a CPIM notification request is sent. This notification is constructed as follows:

- The CPIM watcher is set to the URI in the To field of the NOTIFY.
- The CPIM target is set to the URI in the From field of the NOTIFY.
- The transID is computed using the same mechanism as for the SUBSCRIBE in Section 7.1
- The presence component of the notification is extracted from the body of the SIP NOTIFY request.

The gateway generates a 200 response to the SIP NOTIFY and sends it as well.

TODO: some call flow diagrams with the parameters

8 Firewall and NAT Traversal

It is anticipated that presence services will be used by clients and presentities that are connected to proxy servers on the other side of firewalls and NATs. Fortunately, since the SIP presence messages do not establish independent media streams, as INVITE does, firewall and NAT traversal is much simpler than described in [9] and [10].

Generally, data traverses NATs and firewalls when it is sent over TCP or TLS connections established by devices inside the firewall/NAT to devices outside of it. As a result, it is RECOMMENDED that SIP for presence entities maintain persistent TCP or TLS connections to their next hop peers. This includes connections opened to send a SUBSCRIBE,

NOTIFY, and most importantly, REGISTER. By keeping the latter connection open, it can be used by the SIP proxy to send messages from outside the firewall/NAT back to the client. It is also recommended that the client include a Contact cookie as described in [10] in their registration, so that the proxy can map the presentity URI to that connection.

Furthermore, entities on either side of a firewall or NAT should record-route in order to ensure that the initial connection established for the subscription is used for the notifications as well.

9 Security considerations

There are numerous security considerations for presence. Many are outlined above; this section considers them issue by issue.

9.1 Privacy

Privacy 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

Privacy is provided through a combination of hop by hop encryption and end to end encryption. The hop by hop mechanisms provide scalable privacy 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 also requires that both participants have public keys, which is not generally the case. Thus, both mechanisms combined are needed for complete privacy services.

SIP allows any hop by hop encryption scheme. It is RECOMMENDED that between network servers (proxies to proxies, proxies to redirect servers), transport mode ESP [11] is used to encrypt the entire message. Between a UAC and its local proxy, TLS [12] is RECOMMENDED. Similarly, TLS SHOULD be used between a presence server and the PUA.

The presence server can determine whether TLS is supported by the receiving client based on the transport parameter in the Contact header of its registration. If that registration contains the token "tls" as transport, it implies that the PUA supports TLS.

Furthermore, we allow for the Contact header in the SUBSCRIBE request to contain TLS as a transport. The Contact header is used to route subsequent messages between a pair of entities. It defines the address and transport used to communicate with the user agent. Even though TLS might be used between the subscriber and its local proxy, placing this parameter in the Contact header means that TLS can also be used end to end for generation of notifications after the initial SUBSCRIBE message has been successfully routed. This would provide end to end privacy and authentication services with low proxy overheads.

SIP encryption MAY be used end to end for the transmission of both SUBSCRIBE and NOTIFY requests. SIP supports PGP based encryption, which does not require the establishment of a session key for encryption of messages within a given subscription (basically, a new session key is established for each message as part of the PGP encryption). Work has recently begun on the application of S/MIME [13] for SIP.

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. This is supported in SIP through end to end authentication and message integrity. SIP provides PGP based authentication and integrity (both challenge-response and public key signatures), and http basic and digest authentication. HTTP Basic is NOT RECOMMENDED.

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

To prevent the replay of old subscriptions and notifications, all signed SUBSCRIBE and NOTIFY requests and responses MUST contain a Date header covered by the message signature. Any message with a date

older than several minutes in the past, or more than several minutes into the future, SHOULD be discarded.

Furthermore, all signed SUBSCRIBE and NOTIFY requests MUST contain a Call-ID and CSeq header covered by the message signature. A user agent or presence server MAY store a list of Call-ID values, and for each, the higest CSeq seen within that Call-ID. Any message that arrives for a Call-ID that exists, whose CSeq is lower than the highest seen so far, is discarded.

Finally, challenge-response authentication (http digest or PGP) MAY be used to prevent replay attacks.

9.5 Denial of service attacks

Denial of service attacks are a critical problem for an open, inter-domain, presence protocol. Here, we discuss several possible attacks, and the steps we have taken to prevent them.

9.5.1 Smurf attacks through false contacts

Unfortunately, presence is a good candidate for smurfing 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 is to send subscriptions to a large number of users, and in the Contact header (which is where notifications are sent), place the address of the target.

The only reliable way to prevent these attacks is through authentication and authorization. End users will hopefully not accept subscriptions from random unrecognized users. Also, the presence client software could be programmed to warn the user when the Contact header in a SUBSCRIBE is from a domain which does not match that of the From field (which identifies the subscriber).

Also, note that as described in [3], 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.

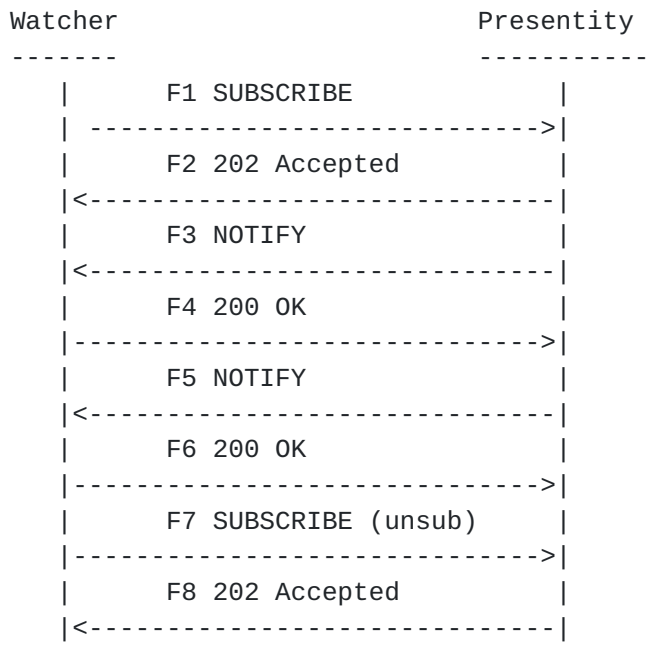
10 Example message flows

The following subsections exhibit example message flows, to further clarify behavior of the protocol.

10.1 Client to Client Subscription with Presentity State Changes

This call flow illustrates subscriptions and notifications that do not involve a presence server.

The watcher subscribes to the presentity, and the subscription is accepted, resulting in a 202 Accepted response. The presentity subsequently changes state (is on the phone), resulting in a new notification. The flow finishes with the watcher canceling the subscription.



Message Details

F1 SUBSCRIBE watcher -> presentity

```
SUBSCRIBE sip:presentity@pres.example.com SIP/2.0
Via: SIP/2.0/UDP  watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Expires: 600
Accept: application/xpidf+xml
Event: presence
Contact: sip:user@watcherhost.example.com
```

F2 202 Accepted presentity->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>;tag=88a7s
Call-ID: 3248543@watcherhost.example.com
Cseq: 1 SUBSCRIBE
Event: presence
Expires: 600
Contact: sip:presentity@pres.example.com
```

F3 NOTIFY Presentity->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>;tag=88a7s
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 1 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:presentity@example.com">
  <tuple destination="sip:presentity@example.com" status="open"/>
</presence>
```

F4 200 OK watcher->presentity

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 1 NOTIFY
```

F5 NOTIFY Presentity->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 2 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:presentity@example.com">
  <tuple destination="sip:presentity@example.com" status="closed"/>
</presence>
```

F6 200 OK watcher->presentity

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 2 NOTIFY
```

F7 SUBSCRIBE watcher -> presentity

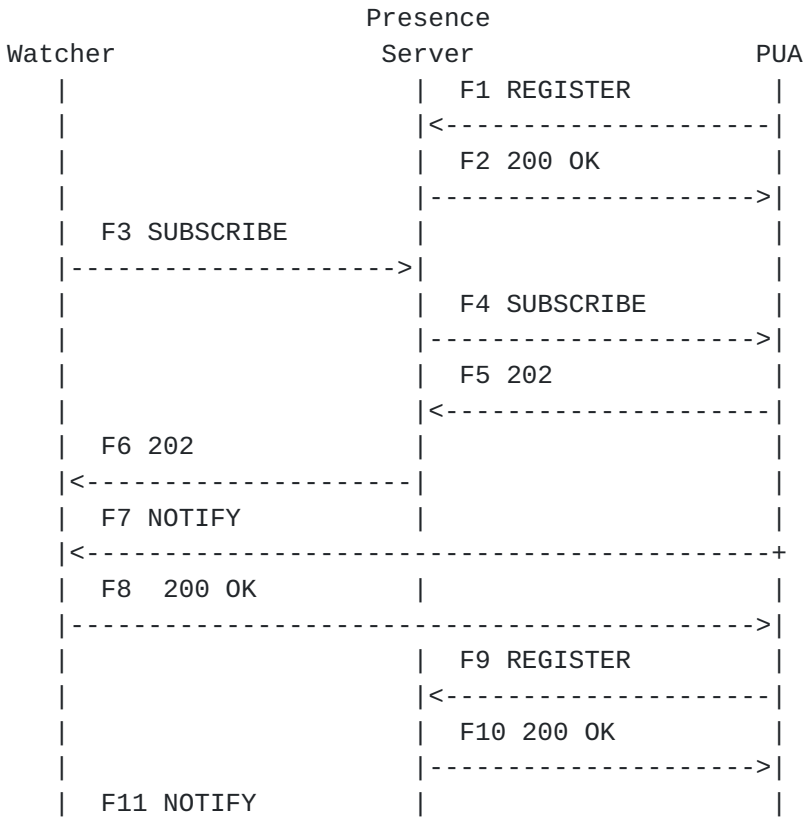
```
SUBSCRIBE sip:presentity@pres.example.com SIP/2.0
Via: SIP/2.0/UDP  watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>
Call-ID: 3248543@watcherhost.example.com
Event: presence
CSeq : 2 SUBSCRIBE
Expires: 0
Accept: application/xpidf+xml
Contact: sip:user@watcherhost.example.com
```

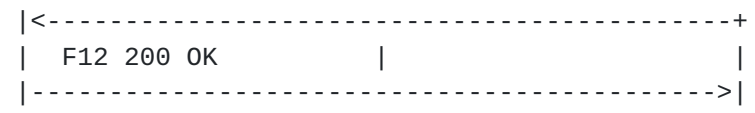
F8 202 Accepted presentity->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>
Call-ID: 3248543@watcherhost.example.com
Event: presence
Cseq: 2 SUBSCRIBE
Expires:0
```

10.2 Presence Server with Client Notifications

This call flow shows the involvement of a presence server in the handling of subscriptions. In this scenario, the client has indicated that it will handle subscriptions and thus notifications. The message flow shows a change of presence state by the client and a cancellation of the subscription by the watcher.





Message Details

F1 REGISTER PUA->server

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 1 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="open"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F2 200 OK server->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 1 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="open"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F3 SUBSCRIBE watcher->server

```
SUBSCRIBE sip:resource@example.com SIP/2.0
Via: SIP/2.0/UDP  watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Expires: 600
Event: presence
Accept: application/xpidf+xml
Contact: sip:user@watcherhost.example.com
```

F4 SUBSCRIBE server->PUA

```
SUBSCRIBE sip:id@pua.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com:5060
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Event: presence
Expires: 600
Accept: application/xpidf+xml
Contact: sip:user@watcherhost.example.com
```

F5 202 Accepted PUA->server

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP server.example.com:5060
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Event: presence
Expires: 600
```

F6 200 OK server->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Event: presence
Expires: 600
```

F7 NOTIFY PUA->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="open"/>
</presence>
```

F8 200 OK watcher->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 NOTIFY
```

F9 REGISTER PUA->server

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="busy"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F10 200 OK server->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="busy"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F11 NOTIFY PUA->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 2 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120
```

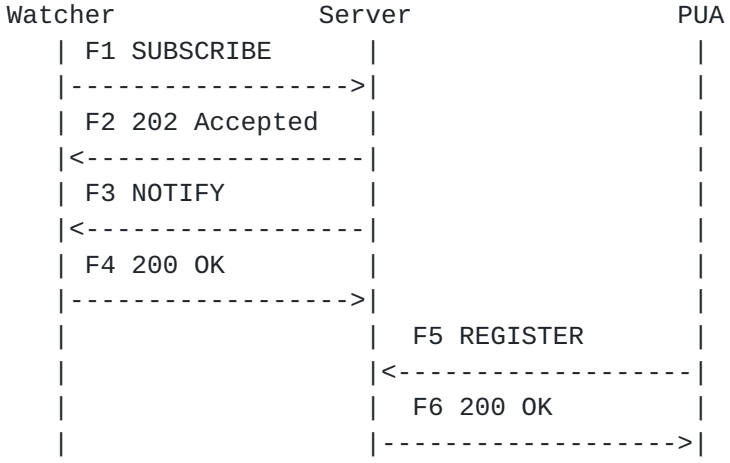
```
<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="busy"/>
</presence>
```

F12 200 OK watcher->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 2 NOTIFY
```

10.3 Presence Server Notifications

This message flow illustrates how the presence server can be the responsible for sending notifications for a presentity. The presence server will do this if the presentity has not sent a registration indicating an interest in handling subscriptions. This flow assumes that the watcher has previously been authorized to subscribe to this resource at the server.



Message Details

F1 SUBSCRIBE watcher->server

```
SUBSCRIBE sip:resource@example.com SIP/2.0
Via: SIP/2.0/UDP watcherhost.example.com:5060
To: <pres:resource@example.com>
From: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 1 SUBSCRIBE
Event: presence
Accept: application/xpidf+xml
Contact: <sip:user@watcherhost.example.com>
Expires: 600
```

F2 202 OK server->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
To: <pres:resource@example.com>;tag=ffd2
From: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 1 SUBSCRIBE
Event: presence
Expires: 600
Contact: sip:example.com
```

F3 NOTIFY server-> watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com:5060
```

```
From: <pres:resource@example.com>;tag=ffd2
To: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
Event: presence
CSeq: 1 NOTIFY
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="open"/>
</presence>
```

F4 200 OK

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server.example.com:5060
From: <pres:resource@example.com>;tag=ffd2
To: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 1 NOTIFY
```

F5 REGISTER

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 110@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="Away from keyboard"
Expires: 600
```

F6 200 OK

```
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 110@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ; description="Away from keyboard"
          ; expires=600
```

F7 NOTIFY

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com:5060
From: <pres:resource@example.com>;tag=ffd2
To: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 2 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="Away from
keyboard"/>
</presence>
```

F8 200 OK

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server.example.com:5060
From: <sip:resource@example.com>;tag=ffd2
To: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 2 NOTIFY
```

11 Open Issues

The following is the list of known open issues:

- This draft recommends that the To and From field are populated with presence URLs rather than sip URLs. Is that reasonable? Will this lead to incompatibilities in proxies? Is there any issues with CPIM if the SIP URL format is used? This depends on what components of a message are signed in CPIM.

- Rate limitations on NOTIFY: do we want that? How do we pick a value? 5 seconds is arbitrary.

- Merging of presence data from multiple PA has been removed. Is that OK?

- Placing IM URLs in the Contact header of a REGISTER: is that OK? What does it mean?

- The process of migrating subscriptions from a presence server to PUA is not likely to work in the case where subscription refreshes use tags and Route headers. So, we have a choice. Either migration is disallowed, and we keep with leg oriented subscriptions, or migration is allowed, and there is no tags or Route's associated with subscriptions.

- Converting SIP URLs back to pres URLs.

- The SIP to CPIM and CPIM to SIP gateways are not stateless, because of the need to maintain Route, Call-ID, CSeq, and other parameters. Perhaps we can ask CPIM to define a token value which is sent in a CPIM request and returned in a CPIM response. Would that help?

- Need to specify how to take Contacts from REGISTER and build a presence document. One obvious thing is that the contact addresses don't go in there directly; you probably want to put the address of record, otherwise calls might not go through the proxy.

12 Changes from -00

The document has been completely rewritten, to reflect the change from a sales pitch and educational document, to a more formal protocol specification. It has also been changed to align with the SIP event architecture and with CPIM. The specific protocol changes resulting from this rewrite are:

- The Event header must now be used in the SUBSCRIBE and NOTIFY requests.
- The SUBSCRIBE message can only have a single Contact header. -00 allowed for more than one.
- The From and To headers can contain presence URIs.
- The Request-URI can contain a presence URI.
- Subscriptions are responded to with a 202 if they are pending or accepted.
- Presence documents are not returned in the body of the SUBSCRIBE response. Rather, they are sent in a separate NOTIFY. This more cleanly separates subscription and notification, and is mandated by alignment with CPIM.
- Authentication is now mandatory at the PA. Authorization is now mandatory at the PA.
- Fake NOTIFY is sent for pending or rejected subscriptions.
- A rate limit on notifications was introduced.
- Merging of presence data has been removed.
- The subscriber rejects notifications received with tags that don't match those in the 202 response to the SUBSCRIBE. This means that only one PA will hold subscription state for a particular subscriber for a particular presentity.
- IM URLs allowed in Contacts in register
- CPIM mappings defined.
- Persistent connections recommended for firewall traversal.

Obtaining Authorization

When a subscription arrives at a PA, the subscription needs to be authorized by the presentity. In some cases, the presentity may have provided authorization ahead of time. However, in many cases, the subscriber is not pre-authorized. In that case, the PA needs to query the presentity for authorization.

In order to do this, we define an implicit subscription at the PA. This subscription is for a virtual presentity, which is the "set of

subscriptions for presentity X", and the subscriber to that virtual presentity is X itself. Whenever a subscription is received for X, the virtual presentity changes state to reflect the new subscription for X. This state changes for subscriptions that are approved and for ones that are pending. As a result of this, when a subscription arrives for which authorization is needed, the state of the virtual presentity changes to indicate a pending subscription. The entire state of the virtual presentity is then sent to the subscriber (the presentity itself). This way, the user behind that presentity can see that there are pending subscriptions. It can then use some non-SIP means to install policy in the server regarding this new user. This policy is then used to either accept or reject the subscription.

A call flow for this is shown in Figure 3.

In the case where the presentity is not online, the problem is also straightforward. When the user logs into their presence client, it can fetch the state of the virtual presentity for X, check for pending subscriptions, and for each of them, upload a new policy which indicates the appropriate action to take.

A data format to represent the state of these virtual presentities can be found in [14].

A Acknowledgements

We would like to thank the following people for their support and comments on this draft:

Rick Workman	Nortel
Adam Roach	Ericsson
Sean Olson	Ericsson
Billy Biggs	University of Waterloo
Stuart Barkley	UUNet
Mauricio Arango	SUN
Richard Shockey	Shockey Consulting LLC
Jorgen Bjorkner	Hotsip
Henry Sinnreich	MCI Worldcom
Ronald Akers	Motorola

B Authors Addresses

Jonathan Rosenberg

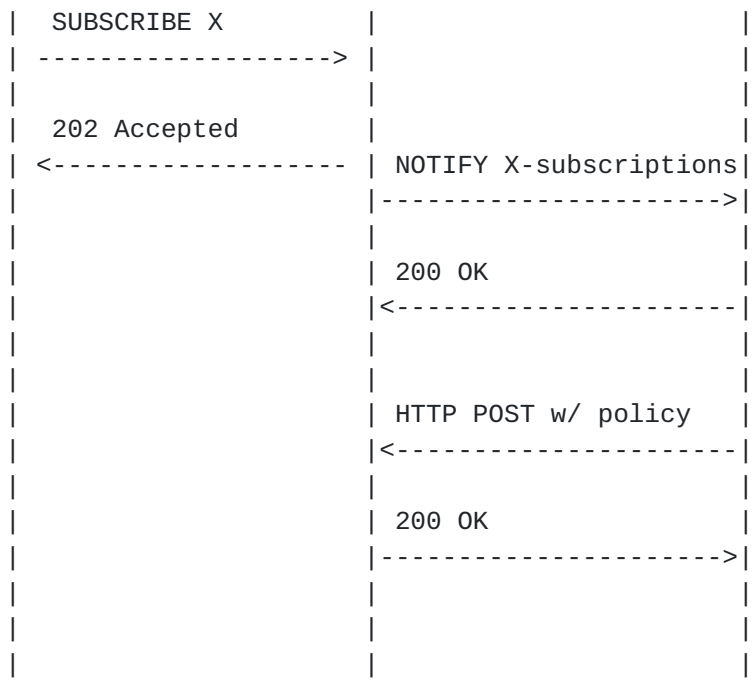


Figure 3: Sequence diagram for online authorization

dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Dean Willis
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
email: dwillis@dynamicsoft.com

Robert Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
email: rsparks@dynamicsoft.com

Ben Campbell
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
email: bcampbell@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

Jonathan Lennox
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: lennox@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
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
email: dgurle@microsoft.com

David Oran
Cisco Systems
170 West Tasman Dr.
San Jose, CA 95134
email: oran@cisco.com

C Bibliography

[1] M. Day, J. Rosenberg, and H. Sugano, "A model for presence and instant messaging," Request for Comments 2778, Internet Engineering Task Force, Feb. 2000.

[2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[3] A. Roach, "Event notification in SIP," Internet Draft, Internet Engineering Task Force, Oct. 2000. Work in progress.

[4] D. Crocker et al. , "A common profile for instant messaging (CPIM)," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[5] P. Calhoun, A. Rubens, H. Akhtar, and E. Guttman, "DIAMETER base protocol," Internet Draft, Internet Engineering Task Force, Sept. 2000. Work in progress.

[6] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (RADIUS)," Request for Comments 2865, Internet Engineering Task Force, June 2000.

[7] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry, "The COPS (common open policy service) protocol," Request for Comments 2748, Internet Engineering Task Force, Jan. 2000.

[8] H. Schulzrinne and J. Rosenberg, "SIP caller preferences and callee capabilities," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[9] J. Rosenberg, D. Drew, and H. Schulzrinne, "Getting SIP through firewalls and NATs," Internet Draft, Internet Engineering Task Force, Feb. 2000. Work in progress.

[10] J. Rosenberg and H. Schulzrinne, "SIP traversal through enterprise and residential NATs and firewalls," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[11] S. Kent and R. Atkinson, "IP encapsulating security payload (ESP)," Request for Comments 2406, Internet Engineering Task Force, Nov. 1998.

[12] T. Dierks and C. Allen, "The TLS protocol version 1.0," Request for Comments 2246, Internet Engineering Task Force, Jan. 1999.

[13] B. Ramsdell and Ed, "S/MIME version 3 message specification," Request for Comments 2633, Internet Engineering Task Force, June 1999.

[14] J. Rosenberg et al. , "An XML based format for watcher information," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

Table of Contents