Internet Engineering Task Force pipr WG
Internet Draft
draft-rosenberg-sip-pip-00.txt
November 13, 1998
Expires: May 1999

J.Rosenberg,H.Schulzrinne
Bell Laboratories,Columbia U.

SIP For Presence

STATUS OF THIS MEMO

ABSTRACT

We describe an extension to SIP for subscription, notification, fetching, and indication of presence events. The extensions consist of two new methods, SUBSCRIBE and NOTIFY.

1 Introduction

An event notification service allows a user (called the subscriber) to subscribe to some entity. Associated with the entity is some state. The subscription is a request to be informed about changes to the state. When state changes occur, a notification is delivered asynchronously to the subscriber. The applicability of the service is extremely broad; events could include things like network management events, presence information, device status, system failures, etc. Subscriptions can be as simple as "notify me when person X logs in"

or as complicated as "notify me when event X in state machine Y occurs if the day is Tuesday and the temperature in Zimbabwe is 85 degrees fahrenheit".

Furthermore, different events and subscriptions will vary in their requirements for reliability, scalability (in terms of number of subscribers for some event), timeliness (in terms of the latency between an event and delivery of the notification to the subscribers), and control (in terms of the complexity of the description of events which may be subscribed to). For example, subscribing to a service which notifies you when concert tickets become available requires a supporting protocol to be scalable, and may mandate multicast. However, reliability is not a concern. On the other hand, subscribing to a service which notifies you when the temperature in the nuclear reactor hits some threshold requires absolute reliability, but scalability is less of a concern.

Due to this breadth of requirements, we do not believe it practical to develop a broad, all-encompassing notification service in the context of the current Internet. Our focus, however, is to solve the problem in the more restrictive context of presence.

We define presence (or presence information) as the means of communication a user is capable or willing to take part in, and may also include contact or address information for those means, preferences about which means to use and when, and state about availability at those means.

A presence event occurs when a user logs in or out of a computer, changes their preferences about reachability at some location, such as a phone or pager, or changes their status at some location. More generally, a presence event is any event which changes the current presence information.

Note that this definition is broader than just "logged in" or "logged out".

2 Architecture

An infrastructure for presence can be broken into three elements and several protocol components. The elements are:

- The Subscriber: an element which asks for notification of the presence of another user. Usually a subscriber is a human.
- The Publisher: The user who has been subscribed to.
- The presence server: a server which performs notification of

presence, and receives subscription requests. This element may or may not be co-resident with the machine on which the publisher is located. For reasons of reliability and availability, a separate server may perform the actual notifications and subscription processing, but this is not required.

The presence server is a key element in a notification system. By allowing it to be a physically separate entity, a number of advantages are gained. The server can perform a number of services for the users it represents, such as proxy encryption, access control and authentication, notification routing, logging, firewalling, and so on. In effect, a presence server is a proxy for the publisher. It is natural to have several presence servers along the protocol message paths. For example, notification requests may pass through a server at both the subscriber and publisher domains. This allows for policy and logging operations to be provided by administrators in both domains.

The server also plays a critical role in providing naming and call routing services. This is discussed in more detail below.

The protocol components in the presence system are:

- Subscription component: a mechanism for conveying the subscription from the subscriber to the presence server. The subscription names the publisher, and may contain additional information about under what conditions the subscriber would like to be notified about presence information related to the publisher. For example, the subscriber may only wish to know when the publisher can be contacted via video.

- Fetching component: a mechanism allowing a user to request the current status of another user, without actually instantiating a subscription. This is effectively a poll, whereas a subscription sets up an asynchronous notification.

- Publication component: When the publisher and presence server are not co-resident, some means is required for informing the presence server about changes in state of the publisher.

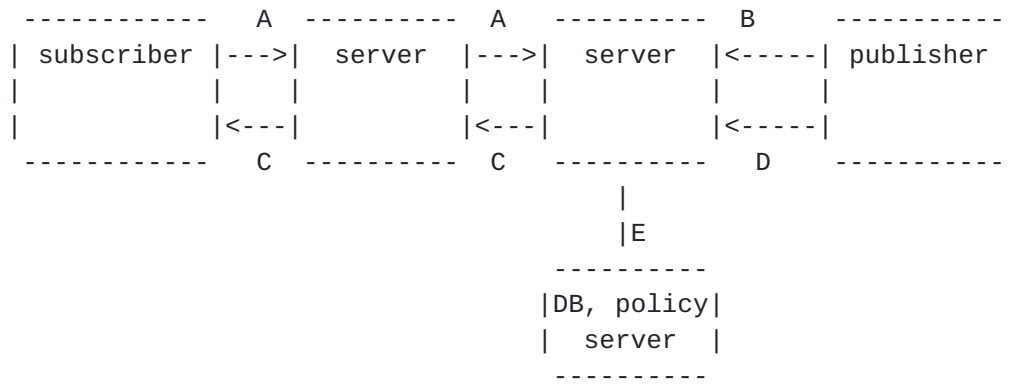
- Access Control component: The publisher needs to provide input about how subscription requests for it will be processed. These preferences may restrict the set of users which may subscribe, cause only select pieces of information to be reported, or potentially even cause false information to be reported based on the subscriber. A means is needed to express and convey these preferences from the publisher to the

presence server.

- Notification component: When there is a change in the presence state of a user, it needs to be conveyed to the subscribers of that user.

We further believe that each of these components is separable into transport and content. The general problem of transport is to provide delivery of the given information in a manner appropriate for the service. The notion of content depends on the task. For subscription, the content should describe the event to be subscribed to. This should effectively define the conditions under which a notification is delivered to the subscriber, and the desired content of that notification. For both notification and publishing, the content is a description of the event which has occured.

The overall architecture is depicted in Figure 1.



A = subscription
B = access control
C = notification
D = publishing
E = database and policy protocols

Figure 1: Architecture of Presence System

In the architectural diagram, there are two servers acting as proxies. In a real system there may be any number, including zero, along the message path. We focus our discussion on the case of two servers, one run by the subscriber's service provider, and the other by the publisher's service provider.

Note that our definition of service provider is NOT the same as ISP. The service provider is any organization which is making its presence servers available to customers. Just as a user can have separate ISP and mail services, a user may have its presence services by yet another organization.

3 Naming, Addressing and Routing

A key component of a presence system is naming. Subscribers must be named, and subscription requests delivered to them, or to the presence servers which represent them. It is critical that the publisher be able to log in to the network with different IP addresses, and for the system to still function. It is also critical that the subscriber be unaware as to whether subscription requests are actually delivered to the publisher directly, or through a chain of presence servers.

The publisher must also be named, so that the presence server of the publisher can perform access control based on this name. The publisher must also provide location information, so that notification requests can be delivered to it. As these are two seperate functions, the subscription protocol must provide a means for indicating the canonical name of the subscriber and the address for delivery of notifications as separate components.

We observe that these requirements for naming are, in fact, identical to those for email and the Session Initiation Protocol (SIP) [1]. As a result, we propose that naming be performed using URLs constructed by email-like identifiers. In cases where a users mail provider and presence provider are the same, the identifiers can also be the same. For example, a user whose email address is jack@mailcompany.com would have a presence URL of the form pip:jack@hotmailcompany.com. This saves space on business cards, and allows database queries to be based on a consistent naming structure. As presence notifications (such as "I'm online now") are often the precursor to actual communications, using the same names simplifies integration.

As a result, a key function of a presence server is the routing of notification and subscription requests based on this URL. Since users are often known by different email identifiers within different scopes, a presence server is also responsible for translation of names. For example, a user, Jack, might publish the address jack@company.com. A user sending a subscription for Jack would use the domain portion of the URI (looking up a pip.udp or pip.tcp SRV record, for example) to forward the subscription to the main presence server for the company. This server might perform some access control, and then, as a result of a lookup in a corporate database, translate the name to j.smith@sales.company.com, an internal address.

The domain name portion of this address is also looked up in the DNS, and the subscription request forwarded to the presence server for the sales department. This server then notes that j.smith is a local user, and it accepts the subscription.

In the example above, the subscriber for Jack won't be aware of the translation or the alternate sales server. As an alternate means of processing the subscription request, a server may elect to redirect. In this scenario, the company's main server would inform the subscriber to contact the server at sales.company.com directly.

In the example above, the main server for the company used a corporate database for the name translation. We observe that any number of means, in fact, can be used for the name translation, including LDAP [2], finger [3], whois [4], whois++ [5], or even multicast queries, with the choice being a purely local matter.

This system of name translation and call routing provides flexibility. Administrators can instantiate any kind of translation logic at servers. Basing the logic on time of day or subscriber preferences enables perrsonal mobility services. For example, a user can publish a single identifier for their presence, such as pip:jdrosen@ieee.org. They can dynamically register new addresses with the IEEE server as they move to new locations, just as is done for email and SIP.

We observe that this system for naming, addressing, and routing are identical in every regard with those used in SIP.

4 SIP For Presence

The discussion so far has presented a general framework for a presence system. We have observed that this architecture is nearly identical to SIP. The presence servers discussed here serve the same purpose as SIP servers - name translation, lookup, routing, logging, access control, and firewalling. SIP provides the same addressing and call routing architecture as proposed. SIP separates content from transport, which fits well with the presence application. This similarity is due to the fact that session initiation and presence subscription are nearly identical functions.

This section demonstrates how the various protocol components required in a presence system are easily implemented with just a few new SIP methods, and no new headers. Almost all of the existing SIP headers are applicable. In the discussion below, all of the syntax and semantics of the headers are the same as in SIP, unless noted otherwise.

4.1 Subscription

The subcription function is accomplished by sending a SIP message from the subscriber, addressed to the publisher. The message is a standard SIP message, but uses a new method, SUBSCRIBE. The message is forwarded until it reaches the server or end user which is performing notifications for the named user. The response to the request follows the standard SIP response codes. 200-class responses indicate success, 300 redirection to an alternate server, 400 client error, 500 server error, and 600 global failure. A 200-class response contains the current presence status for the user, as if it had been fetched.

The request message must contain To, From, Call-ID, Via, and CSeq fields. The To field contains the address of the publisher, and the From field contains the address of the subscriber. The Call-ID field is a unique identifier which groups transactions associated with the subscription. All messages, including notifications from the server, and changes or updates to the subscription, will use the same Call-ID. In this sense, the Call-ID is a "presence session" identifier. The CSeq field provides ordering among messages for the same Call-ID. The Via field is used for routing responses to requests.

A user may resubscribe or update the subscription at any time. In this regard, subscription requests are idempotent, as are SIP requests in general. Subscriptions are cancelled by sending a new SUBSCRIBE with an Expires: 0 in the header.

There are a number of optional headers which may be included in a SUBSCRIBE request. The most important are Contact, which provides a contact address where notifications should be sent by the server, and Expires, which indicates when the subscription expires. All of the other SIP general headers, Date, Encryption, Record-Route, and Timestamp, may be present and retain the same meaning as they do in SIP.

There are a number of request headers are relevant for transporting the presence information. In particular, Accept, Accept-Encoding, and Accept-Language, specify the allowable presence formats which are understood by the subscriber. The response to the request will contain a body that conveys the presence information. This body will be in one of the formats specified by the Accept header in the request. Basic presence information can be expressed in SIP without need for any body, using the Contact headers, exactly as in the call control extensions [6]. These extensions allow for expression of various addresses that the user can be reached at, and for each one, statements of preferences and attributes.

Most of the remaining request headers - Hide, Max-Forwards, Organization, Priority, Proxy-Authorization, Proxy-Require, Route, Response-Key, have the same semantics as in SIP. The Subject header makes less sense for the SUBSCRIBE method. The Require header is used to indicate extensions which must be understood by the server. For presence, the token org.ietf.presence should be included in the Require header.

The subscribe message will normally not contain a body, and thus the entity headers are not required. However, future versions of the presence extensions might allow a body. The body might contain more complex subscriptions, such as "notify me when the user logs in to a cell phone, but no other times".

An example SUBSCRIBE message is depicted in Figure 2.

```
SIP/2.0 SUBSCRIBE pip:jdrosen@bell-labs.com
Via: SIP/2.0/UDP erlang.bell-phone.com:5060
To: pip:jdrosen@bell-labs.com
From: pip:hgs@cs.columbia.edu
Call-ID: 098y0na08fy0h@112.33.58.22
Contact: pip:hgs@play.cs.columbia.edu:488
Accept: text/presence
Organization: Bell Laboratories
Content-Length: 0
```

Figure 2: Example SUBSCRIBE message

This message then traverses some number of servers, eventually arriving at the one which handles presence subscriptions for jdrosen. A success response might look like Figure 3.

The im URL is just illustrative, and reflects what an address for Instant Messaging might look like.

Note that the response includes the currently available presence information for the publisher. The basic information is presented using the SIP Contact header and the call control extensions. Alternatively, the response could contain a body with presence information, as shown in Figure 4.

The presence format listed is just an example - text/presence is not

```
200 OK SIP/2.0
Via: SIP/2.0/UDP proxy.bell-labs.com,
 SIP/2.0/UDP erlang.bell-phone.com:5060
To: pip:jdrosen@bell-labs.com;tag=98asbd987
From: pip:hgs@cs.columbia.edu
Call-ID: 098y0na08fy0h@112.33.58.22
Contact: mailto:jdrosen@bell-labs.com;q=0.8
Contact: im:jdrosen@cs.columbia.edu;q=0.7;mobility=fixed
Contact: sip:jdrosen@alum.mit.edu;q=0.6
Content-Length: 0
```

Figure 3: Example SUBSCRIBE response

```
200 OK SIP/2.0
Via: SIP/2.0/UDP proxy.bell-labs.com,
 SIP/2.0/UDP erlang.bell-phone.com:5060
To: pip:jdrosen@bell-labs.com;tag=98asbd987
From: pip:hgs@cs.columbia.edu
Call-ID: 098y0na08fy0h@112.33.58.22
Content-Length: 58
Content-Type: text/presence

mail-address = jdrosen@bell-labs.com
phone-address = 555-1212
phone-status = busy
im-address = jdrosen@cs.columbia.edu
```

Figure 4: Response to SUBSCRIBE with body

defined. Various new formats can be defined, or existing formats, such as the VCard XML format [7] can be used.

4.2 Notification and Publication

The publication component allows a publisher to inform the notification server about its updated contact and presence information. We believe that there are many possible ways in which publication can take place. For example, a system can be co-resident with a Network Access Server (NAS), and presence can be represented as simply logged in or logged out. In this case, there is no separate protocol for conveying publication of presence information. It is done as a natural consequence of logging into the server.

Other mechanisms include SIP registrations, H.323 gatekeeper registrations, telnets, picking up the phone, sensors in the door (or in the chair) of someones office. All of these are valid means to convey some form of presence to the notification server.

We propose that if an explicit publication means is needed, it should use the notification messages. This is because publication is really a form of notification, with the notification is between the publisher and the notification server. This is a natural extension of the serverless case, where the publisher would be responsible for sending the notification requests to the subscriber. It also makes simple notification servers easy. They can effectively proxy, without modification, a notification request from the subscriber. The only change needed is to modify the request URI and fork the request. This is a normal operation for a SIP server.

The process of publication is supported through the NOTIFY method, a new method added to SIP for this purpose. A notification request is sent by a publisher, and targeted to the notification server just as a SIP register message is targeted to a registrar (there is no username). No additional header fields are needed to support publications and notifications through SIP.

The mandatory headers are To, From, Call-ID, and CSeq. The To field is set the same way as in a SIP REGISTER message - it is the address of the entity whose information is being updated, NOT the target of the registration. The From field is the entity actually sending the notification (this need not be the same as the To field). The Call-ID is the same as the one in the SUBSCRIBE which triggered it. Cseq is randomly chosen, but must be larger than the CSeq in a previous message from the server with the same Call-ID.

As with the response to the SUBSCRIBE request, the NOTIFY request may either contain a payload which describes the current presence state of the user, or this information may be conveyed in the Contact headers. A presence server may need to translate any presence bodies if some of the subscribers don't understand the format used by the publisher. The entire state, not pieces of it, are always sent.

The request URI of a publication contains just the address of the notification server. The notification server will fork the notification (after any appropriate modifications), and send copies to each of the subscribers by changing the request URI in each of the copies. The Request URI contains the value from the Contact header received in each registration.

The Expires header may be used in notifications to indicate that the information has a finite lifetime.

The remaining request header fields have their standard meanings. Note that the Requires header is needed here as well.

The response to a NOTIFY is a 200-class response if it has been received correctly. A 400-class response is returned if a subscriber receives a notification, but it had never subscribed to that user.

An example notification sent from a publisher to its notification server is depicted in Figure 5.

```
SIP/2.0 NOTIFY pip:dnrc.bell-labs.com
To: pip:jdrosen@bell-labs.com
From: pip:jdrosen@dnrc.bell-labs.com
Call-ID: 098n08ayfp@10.0.0.1
CSeq: 0
Via: SIP/2.0/UDP machine.dnrc.bell-labs.com
Contact: sip:jdrosen@bell-labs.com:5061;expires=3600
```

Figure 5: Example NOTIFY request

This is then received at the server. Assume there was a single subscription, the example in the previous section. The notification server would then generate a single notification, an example of which is shown in Figure 6.

```
SIP/2.0 NOTIFY pip:hgs@play.cs.columbia.edu
To: pip:jdrosen@bell-labs.com
From: pip:jdrosen@dnrc.bell-labs.com
Call-ID: 098n08ayfp@10.0.0.1
CSeq: 0
Via: SIP/2.0/UDP machine.dnrc.bell-labs.com
Contact: sip:jdrosen@bell-labs.com:5061
```

Figure 6: Example NOTIFY response

This would then be sent to port 488 of the machine play, delivering it to the subscriber.

The NOTIFY request can also be sent by multicast as a configuration-less means of publication. It is sent to the all-SIP-servers

multicast address (224.1.0.75), and the server which is acting as a proxy for that user will accept it, send a 200 OK response, and perform the notification. As with multicast REGISTER requests, this avoids the need for knowledge about the server.

4.3 Fetching

Fetching presence allows a subscriber to synchronously obtain the current state of the publisher, without establishing any registration. This is accomplished easily without any additional methods. Since the response to a SUBSCRIBE method contains the current presence information, a user can fetch the current state by sending a SUBSCRIBE with an Expires header with a time of 0. This will return the current state without causing a registration.

To fetch the presence information without modifying an existing registration, the fetch should use a different Call-ID than the original registration.

5 Access Control

Access control is an optional part of the presence system. It allows a publisher to express preferences to the server about how and when notifications get delivered to subscribers. These preferences could include things like time of day preferences ("tell people I'm online only on Tuesdays"), per-subscriber preferences ("don't tell Joe anything"), preferences based on communications means ("only advertise my telephone number"), and combinations therein ("on Tuesdays, don't tell Joe about my telephone presence").

To support such a service, there needs to be a syntax, carried from the publisher to the server, which expresses these simple rules. Currently, such a syntax is under development [8]. The context is focused towards telephony preferences, but is easily extended to support presence preferences as well. For example, Figure 7 shows an example XML based script for controlling call routing.

The script tells the proxy server that when a call arrives for it, it should be proxied to sip:bob@mci.com. If its busy, the caller should be redirected to joe@mit.edu. If there is no answer, if the call is from hgs@cs.columbia.edu, the call should be proxied to the phone number +1 917 555-1212, otherwise the caller should be redirected to contact sip:bill@att.com. A simple extension, and example of which is given in figure 8 could effectively provide a similar service for notifications.

```
<call>
  <proxy dest="sip:bob@mci.com" timeout="8s">
    <busy>
      <redirect dest="sip:joe@mit.edu"/>
    </busy>
    <timeout>
      <condition from="hgs@cs.columbia.edu">
        <match>
          <gateway dest="phone:+19175551212"/>
        </match>
        <nomatch>
          <redirect dest="sip:bill@att.com"/>
        </nomatch>
      </condition>
    </timeout>
  </proxy>
</call>
```

Figure 7: SIP Call Processing Script

```
<notify>
  <condition uri="hgs@cs.columbia.edu">
    <match>
      <proxy/>
    </match>
    <nomatch>
      <condition contact="phone:5551212">
        <match>
          <no-proxy/>
        </match>
        <nomatch>
          <proxy>
        </nomatch>
      </condition>
    </nomatch>
  </condition>
</notify>
```

Figure 8: PIP Call Processing Example

This script has the effect of causing all notifications for hgs@cs.columbia.edu to be delivered. However, notifications for everyone else are only delivered if they don't contain a phone contact, otherwise they are dropped.

The Call Processing Language for iptel is ideal for caller, time of day, and simple rule based preferences. More complex policies for delivery of notifications will require more flexible means of expression.

6 Aggregating Subscriptions

Since SIP readily supports hopping requests and responses through many servers, and also supports loop detection, it can be used to provide aggregation of subscriptions. Consider the configuration in Figure 9.

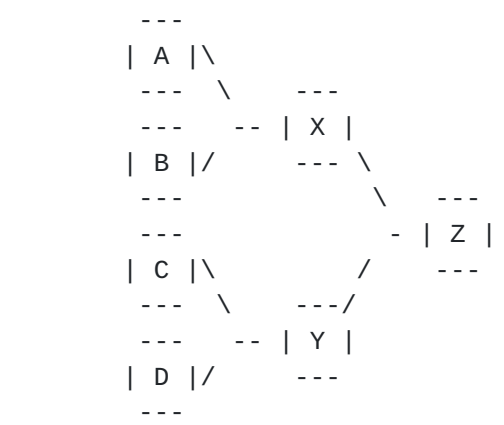


Figure 9: Aggregation Scenario

A, B, C, and D are subscribers. X and Y are intermediate proxy servers, and Z is the subscription server. A, B, C, and D all generate subscriptions for the same user. However, instead of passing these subscriptions to Z, servers X and Y only pass the first subscription. Furthermore, X and Y replace the Contact header in the subscription with their own address. X and Y also keep a list of all users who subscribe to a given user. So, when B subscribes to a user for whom A has already subscribed, X makes a note of it but does not propagate the subscription. Z will think that only X and Y have subscribed to a given user.

When a notification is generated, it is forwarded to X and Y. They, in turn, generate additional copies, and forward them to A, B, C and D. This effectively forms a hop by hop acknowledged multicast distribution tree.

This mechanism becomes less effective when the various servers have different access control policies.

7 Event Splitting and Merging

In some systems, a presence server can take a more active role in presence notifications. For example, a simple publisher may only indicate basic events, such as "logged in", or "logged out". The presence server may take this information, and combine it with other state, to generate different presence notifications. For example, if the date was Tuesday, the server might generate events of the form "logged in on Tuesday". Similarly, a publisher may generate the events "business phone is 555-1212, home phone 888-8888" and then "available only at home", and the server may combine them to generate the aggregate event "home phone is 555-1212". Combination of multiple notifications is referred to merging.

Similarly, a publisher may generate a single event, such as "logged out", and the server may generate multiple events as a result, such as "not available at home" and "not available at home". This is referred to as splitting.

The operations of merging and splitting are available to a server. They may be used to provide additional services to users of the system. The merging operation can also be used to provide additional scalability, by reducing the number of messages sent by a server.

Of course, a publisher which makes use of end to end authentication will eliminate the possibility of merging and splitting at a server.

8 Reliability and Scalability

SIP works with either TCP or UDP (or both, in the case of multiple servers, each using a different protocol). In the case of UDP, it provides its own simple reliability mechanisms. These mechanisms are based on a simple retransmit timer at the client. This is appropriate for a message oriented protocol such as SIP. It is much less complex than TCP, since the flow and congestion control aspects are not needed.

The reduced complexity directly improves scalability. A server can handle more SIP subscriptions through UDP than TCP. Furthermore, servers which simply act as proxies, without processing or changing a

message, can forward requests and the responses they generate without any state.

In cases where there are large numbers of subscribers, multicast may be appropriate. Since SIP works with UDP, additional headers can be defined to support multicast notifications. SIP can work with existing reliable multicast protocols, or new means for SIP in particular are easily defined.

These features of SIP make it ideal for providing reliability in a scalable fashion over the operating range of a presence protocol.

9 Security

SIP supports http's basic and digest authentication [9], in addition to PGP or other public key based schemes. SIP also supports end-to-end and proxy encryption, and hiding of key hop-by-hop headers. These can all be reused as is for the presence extensions.

10 Requirements

We considered the requirements in [10] to determine how many are met by SIP with these extensions:

4.1 A client MUST be able to communicate its presence information, either directly or via intermediaries such as servers, to other clients.

SIP, with the NOTIFY method, supports such presence notification. Presence is conveyed by URLs and URL parameters. SIP supports proxy and redirect through many servers, with headers for loop detection and prevention.

4.2 All clients MUST implement some common presence format for presence information.

Our proposal is to define the URL as a basic notion of presence.

4.3 The common presence format MUST include a means to represent an individual name (a personal name in the case of a person), and organizational or other disambiguating information.

URL's contain such information.

4.4 The common presence format MUST include a means to represent contact information, such as email address, telephone number, postal address, or the like.

URL's provide exactly this. They exist for email (mailto), telephone (phone), multimedia (sip), to name a few.

4.5 The common presence format MUST include a means to represent at least the following conditions: active, inactive, unavailable, do not disturb.

These are easily supported through URL parameters, such as those described in the SIP call control extensions.

4.6 There MUST be a means of extending the common presence format to represent additional information not included in the common format, without undermining or rendering invalid the fields of the common format.

SIP allows for bodies to be transported in all of its messages. Simple presence can be conveyed with URL's with paramaters. More complex notions of presence are supported through bodies carried in any SIP message. SIP makes use of the Accept, Content-Type, and Accept-Language headers to provide naming and negotiation of the types of bodies which are supported. This allows for numerous formats for presence to be defined.

4.7 A client MUST be able to indicate its interest in the presence information of other clients, even when those other clients are not available or not reachable.

This is supported by the SUBSCRIBE method proposed here. SIP deliveres requests to either proxy servers or user agent servers (end system). Thus, they can be delivered whether or not the end user is logged in and available.

4.8 When a client changes its presence information, and another client has indicated interest in the presence information of that client, the interested client MUST receive the changed information rapidly enough that the delay is not objectionable. For most applications, this implies a delay of no more than a few seconds.

These kind of delay requirements are identical to those for initiating multimedia sessions (when a user answers the phone, the caller must be notified rapidly). SIP has carefully tuned its timing to meet these objectives.

4.9 The protocol MUST provide a means so that a client receiving an update can be confident that it represents the correct presence information (that is, it has not been corrupted or delayed).

SIP supports message integrity of end to end headers and the body.

4.10 The protocol MUST provide a means so that a client receiving an update can be confident that it represents the presence information of the client claimed (that is, it has not been forged).

SIP supports cryptographically strong authentication via PGP, S/MIME, or other means.

4.11 The protocol MUST provide a means for changing presence information automatically in circumstances such as broken network connections, which cannot be anticipated by a client providing its presence information.

SIP Registrations time out eventually. SIP supports clients defining the time out interval, with servers reducing it based on policy. This would allow presence information that has been registered to time out in the case of long term network failures.

11 Open Issues

1. Call-IDs: What is the scope of the Call-ID? Should it be the same for subscriptions and the notifications they generate?

2. REGISTER vs. NOTIFY: The registration message in SIP provides a similar function to NOTIFY. Both contain a description of the current addresses and communications means supported by a client. However, their function at a server is different. A NOTIFY method is an FYI - its propagated by the server to subscribers. In most cases, the publisher won't even know who the subscribers are. In the case of REGISTER, the message establishes call routing state in a single proxy server. The information is not propagated. For this reason, usage of different methods seems appropriate. However, there is no restriction about using REGISTER as a publication means. As pointed out above, a publisher can use any mechanism to notify the server about their presence.

3. pip scheme: Is it necessary to use a new URL scheme for presence, or can the existing sip scheme be used? Having them separate makes it apparent whether the address is for presence or communications. Having a pip URL in a web page might cause a SUBSCRIBE to be sent, whereas a sip URL might cause an INVITE to be sent. On the other hand, the method tag in the SIP URL could be used to provide disambiguity.

4. BYE: Do we allow a subscription to be cancelled with a BYE? It seems appropriate.

12 Conclusion

In this document, we have discussed the presence notification problem, and presented an architectural solution to the problem. We then showed that the Session Initiation Protocol, extended with just two new methods and no new headers, can serve as a fully functional, easily extendable, integrated presence system.

13 Acknowledgements

The authors would like to thank William Nagy for his comments and input.

14 Full Copyright Statement

15 Bibliography

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Internet Draft, Internet Engineering Task Force, Sept. 1998. Work in progress.

[2] T. Howes, S. Kille, and M. Wahl, "Lightweight directory access protocol (v3)," Request for Comments (Proposed Standard) 2251, Internet Engineering Task Force, Dec. 1997.

[3] D. Zimmerman, "The finger user information protocol," Request for Comments (Draft Standard) 1288, Internet Engineering Task Force, Dec. 1991. (Obsoletes RFC1196).

[4] E. Feinler, K. Harrenstien, and M. Stahl, "NICNAME/WHOIS," Request for Comments (Draft Standard) 954, Internet Engineering Task Force, Oct. 1985. (Obsoletes RFC812).

[5] C. Weider, J. Fullton, and S. Spero, "Architecture of the whois++ index service," Request for Comments (Proposed Standard) 1913, Internet Engineering Task Force, Feb. 1996.

[6] H. Schulzrinne and J. Rosenberg, "SIP call control services," Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.

[7] F. Dawson, "The vcard v3.0 XML DTD," Internet Draft, Internet Engineering Task Force, July 1998. Work in progress.

[8] H. Schulzrinne and J. Lennox, "Call processing language requirements," Internet Draft, Internet Engineering Task Force, Aug. 1998. Work in progress.

[9] J. Franks, P. Hallam-Baker, and J. Hostetler, "An extension to HTTP: digest access authentication," Request for Comments (Proposed Standard) 2069, Internet Engineering Task Force, Jan. 1997.

[10] M. Day, "Requirements for presence and instant messaging," Internet Draft, Internet Engineering Task Force, Mar. 1998. Work in progress.

16 Authors Addresses

Jonathan Rosenberg
Lucent Technologies, Bell Laboratories
101 Crawfords Corner Rd.
Holmdel, NJ 07733
Rm. 4C-526
email: jdrosen@bell-labs.com

Henning Schulzrinne
Columbia University
M/S 0401

1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu