                  Publication of Partial Presence Information

Status of This Memo

Abstract

   The Session Initiation Protocol (SIP) Extension for Event State
   Publication describes a mechanism with which a presence user agent is
   able to publish presence information to a presence agent. Using the
   Presence Information Data Format (PIDF), each presence publication
   contains full state, regardless of how much of that information has
   actually changed since the previous update. As a consequence,
   updating a sizeable presence document with small changes bears a
   considerable overhead and is therefore inefficient. Especially with
   low bandwidth and high latency links, this can constitute a
   considerable burden to the system. This memo defines a solution that
   aids in reducing the impact of those constraints and increases
   transport efficiency by introducing a mechanism that allows for
   publication of partial presence information.

Table of Contents

1. Introduction

    The Session Initiation Protocol (SIP) Extension for Event State
    Publication [RFC3903] allows Presence User Agents ('PUA') to publish
    presence information of a user ('presentity'). The Presence Agent
    (PA) collects publications from one or several presence user agents,
    and generates the composite event state of the presentity.

    The baseline format for presence information is defined in the
    Presence Information Data Format (PIDF) [RFC3863] and is by default
    used in presence publication. The PIDF uses Extensible Markup
    Language (XML) [W3C.REC-xml], and groups data into elements called
    tuples. In addition, [RFC4479], [RFC4480], [RFC4481], [RFC4482], and
    [RFC5196] define extension elements that provide various additional
    features to PIDF.

    Presence publication by default uses the PIDF document format, and
    each publication contains full state, regardless of how much of the
    presence information has actually changed since the previous update.
    As a consequence, updating a sizeable presence document especially
    with small changes bears a considerable overhead and is therefore
    inefficient. Publication of information over low bandwidth and high
    latency links further exacerbates this inefficiency.

    This memo specifies a mechanism with which the PUA is after an
    initial full state publication able to publish only those parts of
    the presence document that have changed since the previous update.
    This is accomplished using the partial PIDF [RFC5262] document format

   to communicate a set of presence document changes to the PA, who then
   applies the changes in sequence to its version of the presence
   document.

   This memo is structured in the following way: Section 3 gives an
   overview of the partial publication mechanism, Section 4 includes the
   detailed specification, Section 5 includes discussion of security
   considerations, and Section 6 includes examples of partial
   publication.

2. Definitions and Document Conventions

   In this document, the key words "MUST", "MUST NOT", "REQUIRED",
   "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY",
   and "OPTIONAL" are to be interpreted as described in RFC 2119, BCP 14
   [RFC2119], and indicate requirement levels for compliant
   implementations.

   This document makes use of the vocabulary defined in the Model for
   Presence and Instant Messaging [RFC2778], the Event State Publication
   Extension to SIP [RFC3903], and the PIDF Extension for Partial
   Presence [RFC5262].

3. Overall Operation

   This section introduces the baseline functionality for presence
   publication, and gives an overview of the partial publication
   mechanism. This section is informational in nature. It does not
   contain any normative statements.

3.1. Presence Publication

   Event State Publication is specified in [RFC3903].

   The publication of presence information consists of a presence user
   agent sending a SIP PUBLISH request [RFC3903] targeted to the
   address-of-record of the presentity, and serviced by a presence agent
   or compositor. The body of the PUBLISH request carries full event
   state in the form of a presence document.

   The compositor processes the PUBLISH request and stores the presence
   information. It also assigns an entity-tag that is used to identify
   the publication. This entity-tag is returned to the PUA in the
   response to the PUBLISH request.

   The PUA uses the entity-tag in the following PUBLISH request for
   identifying the publication that the request is meant to refresh,
   modify or remove. Presence information is stored in an initial

publication, and maintained using the refreshing and modifying
publications. Presence information disappears either by explicitly
removing it or when it meets its expiration time.

3.2. Partial Presence Publication

The partial publication mechanism enables the PUA to update only
parts of its presence information, namely those sections of the
presence document that have changed. The initial publication always
carries full state. However, successive modifying publications to
this initial presence state can communicate state deltas, i.e., one
or more changes to the presence information since the previous
update. Versioning of these partial publications is necessary to
guarantee that the changes are applied in the correct order. The
PUBLISH method [RFC3903] already accomplishes this using entity-tags
and conditional requests, which guarantee correct ordering of
publication updates.

   Note that the partial PIDF format [RFC5262] contains the 'version'
   attribute that could be used for versioning as well. However, we
   chose not to introduce an additional versioning mechanism to
   partial publish, since that would only add ambiguity and a
   potentially undefined error case if the two versioning mechanisms
   were to somehow contradict.

To initialize its publication of presence information, the PUA first
publishes a full state initial publication. The consequent modifying
publications can carry either state deltas or full state. Both
initial and modifying partial presence publications are accomplished
using the 'application/pidf-diff+xml' content type [RFC5262], with
the former using the <pidf-full> root element, and the latter using
the <pidf-diff> or <pidf-full> root elements, respectively.

While the <pidf-full> encapsulates a regular PIDF document, the
<pidf-diff> can contain one or more operations for adding new
elements or attributes (<add> elements), replacing elements or
attributes whose content has changed (<replace> elements), or
indications of removal of certain elements or attributes (<remove>
elements). The PUA is free to decide the granularity by which
changes in presence information are communicated to the composer. It
may very well happen that there are enough changes to be communicated
that it is more efficient to send a full state publication instead of
a set of state deltas.

When the presence compositor receives a partial publication, it
applies the included patch operations in sequence. The resulting
changed (or patched) presence document is then submitted to the
composition logic in the same manner as with a full state presence

publication. Similarly, any changes to the publication expiration
apply to the full, patched presence publication. In other words,
there is no possibility to roll back to an earlier version, except by
submitting a full state publication.

4. Client and Server Operation

Unless otherwise specified in this document, the presence user agent
and presence agent behavior are as defined in [RFC3903].

4.1. Content-Type for Partial Publications

The entities supporting the partial publication extension described
in this document MUST support the 'application/pidf-diff+xml' content
type defined in the partial PIDF format [RFC5262], in addition to the
baseline 'application/pidf+xml' content type defined in [RFC3863].

Listing the partial PIDF content type in the Accept header field of a
SIP response is an explicit indication of support for the partial
publication mechanism. The PUA can learn server support either as a
result of an explicit query, i.e., in a response to an OPTIONS
request, or by trial-and-error, i.e., after a 415 error response is
returned to an attempted partial publication.

4.2. Generation of Partial Publications

Whenever a PUA decides to begin publication of partial presence
information, it first needs to make an initial publication. This
initial publication always carries full state. After the initial
publication, presence information can be updated using modifying
publications; the modifications can carry state deltas as well as
full state. Finally, the publication can be terminated by explicit
removal, or by expiration.

Both the initial and modifying publications make use of the partial
presence document format [RFC5262], and all follow the normal rules
for creating publications, as defined in RFC 3903 [RFC3903], Section
4.

If the initial PUBLISH request returns a 415 (Unsupported Media
Type), it means that the compositor did not understand the partial
publication format. In this case, the PUA MUST follow normal
procedures for handling a 400-class response, as specified in Section
8.1.3.5 of [RFC3261]. Specifically, the PUA SHOULD retry the
publication using the default PIDF content type, namely 'application/
pidf+xml'. In addition, to find out a priori whether a specific
presence compositor supports partial presence publication, the PUA
MAY use the OPTIONS method, as described in [RFC3261].

To construct a full-state publication, the PUA uses the following
process:

o The Content-Type header field in the PUBLISH request MUST be set
   to the value 'application/pidf-diff+xml'.

o The document in the body of the request is populated with a <pidf-
   full> root element that includes the 'entity' attribute set to
   identify the presentity.

o Under the <pidf-full> root element exists all of the children of a
   PIDF [RFC3863] <presence> element. This document contains the
   full state of which the PUA is aware, and MAY include elements
   from any extension namespace.

To construct a partial publication, the following process is
followed:

o The Content-Type header field in the PUBLISH request MUST be set
   to the value 'application/pidf-diff+xml'.

o The document in the body of the request is populated with a <pidf-
   diff> root element that includes the 'entity' attribute
   identifying the presentity.

o Under the <pidf-diff> root element exists a set of patch
   operations that communicate the changes to the presentity's
   presence information. These operations MUST be constructed in
   sequence, and as defined in the partial PIDF format [RFC5262].

The PUA is free to decide the granularity by which changes in the
presentity's presence information are communicated to the presence
compositor. In order to reduce unnecessary network traffic, the PUA
SHOULD batch several patch operations in a single PUBLISH request.

   A reasonable granularity might be to batch state changes resulting
   from related UI events together in a single PUBLISH request. For
   example, when the user sets their status to "Away", several things
   including freetext notes, service availability, and activities
   might change as a result.

If the size of the delta state becomes more than the size of the full
state, the PUA SHOULD instead send a modifying publication carrying
full state, unless this size comparison is not possible.

   To an implementation that generates state deltas directly out of
   its internal events, it may not be trivial to determine the size
   of the corresponding full state.

4.3. Processing of Partial Publications

   For each resource, the compositor maintains a record for each of the
   publications. These are indexed using the entity-tag of the
   publications.

   Processing of publications generally follows the guidelines set in
   [RFC3903]. In addition, processing PUBLISH requests that contain
   'application/pidf-diff+xml' require some extra processing that is
   dependant on whether the request contains full or partial state.

4.3.1. Processing <pidf-full>

   If the value of the Content-Type header field is 'application/
   pidf-diff+xml', and the document therein contains a <pidf-full> root
   element, the publication contains full presence information, and the
   next step applies:

   o The compositor MUST take the received presence document under the
      <pidf-full> as the local presence document, replacing any previous
      publications.

   If any errors are encountered before the entire publication is
   completely processed, the compositor MUST reject the request with a
   500 (Server Internal Error) response, and revert back to its
   original, locally stored presence information.

4.3.2. Processing <pidf-diff>

   If the value of the Content-Type header field is 'application/
   pidf-diff+xml', and the document in the body contains a <pidf-diff>
   root element, the publication contains partial presence information
   (state delta), and the next steps apply:

   o If the publication containing the <pidf-diff> root element is a
      modifying publication (i.e., contains an If-Match header field
      with a valid entity-tag), the compositor MUST apply the included
      patch operations in sequence against its locally stored presence
      document.

   o Else, the publication is an initial publication, for which only
      <pidf-full> is allowed. Therefore, the publication MUST be
      rejected with an appropriate error response, such as a 400
      (Invalid Partial Publication).

   If a publication carrying partial presence information expires
   without the PUA refreshing it, the compositor MUST clear the entire,
   full state publication.

This means that the compositor does not keep a record of the
applied patches, and consequently (unlike some versioning
systems), the compositor does not roll back to an earlier version
if a particular partial publication were to expire.

If the compositor encounters errors while processing the
'application/pidf-diff+xml' document, it MUST reject the request with
a 400 (Bad Request) response. In addition, the compositor MAY
include diagnostics information in the body of the response, using an
appropriate error condition element defined in Section 5.1. of
[RFC5261].

If any other errors are encountered before the entire partial
publication is completely processed, including all of the patch
operations in the 'application/pidf-diff+xml' body, the compositor
MUST reject the request with a 500 (Server Internal Error) response,
and revert back to its original, locally stored presence information.

5. Security Considerations

This specification relies on protocol behavior defined in [RFC3903].
General security considerations related to Event State Publication
are extensively discussed in that specification and all the
identified security considerations apply to this document in
entirety. In addition, this specification adds no new security
considerations.

6. Examples

The following message flow (Figure 1) shows an example of a presence
system that applies the partial publication mechanism.

First, the PUA sends an initial publication that contains full state.
In return, it receives a 200 OK response containing an entity-tag.
This entity-tag serves as a reference with which the initial full
state can be updated using partial publications containing state
deltas.

Then at some point the resource state changes, and the PUA assembles
these changes into a set of patch operations. It then sends a
modifying publication containing the patch operations, using the
entity-tag as a reference to the publication against which the
patches are to be applied. The compositor applies the received patch
operations to its local presence document in sequence, and returns a
200 OK, which includes a new entity-tag.

```
                                    Presence Agent /
                   PUA                  Compositor
                     | (M1) PUBLISH          |
                     |---------------------------->|
                     | (M2) 200 OK          |
                     |<----------------------------|
                     |                      |
                     |                      |
                     |                      |
                     | (M3) PUBLISH          |
                     |---------------------------->|
                     | (M4) 200 OK          |
                     |<----------------------------|
                     |                      |
                    _|_                    _|_
```
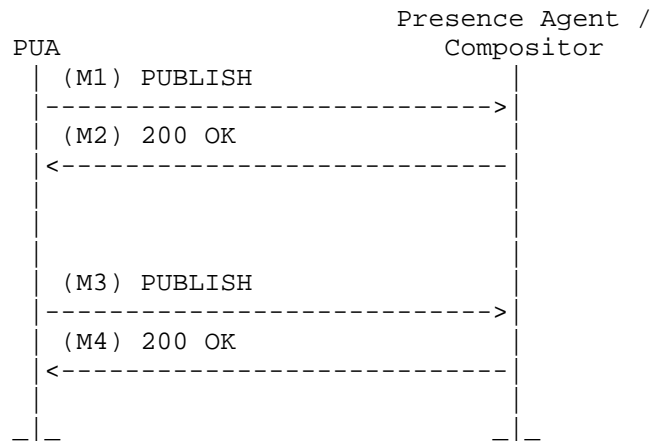
               Figure 1: Partial Publication Message Flow

   Message details:

   (M1): PUA -> Compositor

        PUBLISH sip:resource@example.com SIP/2.0
        ...
        Event: presence
        Expires: 3600
        Content-Type: application/pidf-diff+xml
        Content-Length: 1457

        <?xml version="1.0" encoding="UTF-8"?>
        <p:pidf-full xmlns="urn:ietf:params:xml:ns:pidf"
               xmlns:p="urn:ietf:params:xml:ns:pidf-diff"
               xmlns:r="urn:ietf:params:xml:ns:pidf:rpid"
               xmlns:c="urn:ietf:params:xml:ns:pidf:caps"
               entity="pres:someone@example.com">

         <tuple id="sg89ae">
          <status>
           <basic>open</basic>
           <r:relationship>assistant</r:relationship>
          </status>
          <c:servcaps>
           <c:audio>true</c:audio>
           <c:video>false</c:video>
           <c:message>true</c:message>
          </c:servcaps>
          <contact priority="0.8">tel:09012345678</contact>
         </tuple>
```

```
           <tuple id="cg231jcr">
            <status>
             <basic>open</basic>
            </status>
            <contact priority="1.0">im:pep@example.com</contact>
           </tuple>

           <tuple id="r1230d">
            <status>
             <basic>closed</basic>
             <r:activity>meeting</r:activity>
            </status>
            <r:homepage>http://example.com/~pep/</r:homepage>
            <r:icon>http://example.com/~pep/icon.gif</r:icon>
            <r:card>http://example.com/~pep/card.vcd</r:card>
            <contact priority="0.9">sip:pep@example.com</contact>
           </tuple>

           <note xml:lang="en">Full state presence document</note>
           <r:person>
            <r:status>
             <r:activities>
              <r:on-the-phone/>
              <r:busy/>
             </r:activities>
            </r:status>
           </r:person>

           <r:device id="urn:esn:600b40c7">
            <r:status>
             <c:devcaps>
              <c:mobility>
               <c:supported>
                <c:mobile/>
               </c:supported>
              </c:mobility>
             </c:devcaps>
            </r:status>
           </r:device>

          </p:pidf-full>
```

```
(M2): Compositor -> PUA

      SIP/2.0 200 OK
      ...
      SIP-ETag: 61763862389729
      Expires: 3600
      Content-Length: 0

(M3): PUA -> Compositor

      PUBLISH sip:resource@example.com SIP/2.0
      ...
      Event: presence
      SIP-If-Match: 61763862389729
      Expires: 3600
      Content-Type: application/pidf-diff+xml
      Content-Length: 778

      <?xml version="1.0" encoding="UTF-8"?>
      <p:pidf-diff xmlns="urn:ietf:params:xml:ns:pidf"
                 xmlns:p="urn:ietf:params:xml:ns:pidf-diff"
                 xmlns:r="urn:ietf:params:xml:ns:pidf:rpid"
                 entity="pres:someone@example.com">

       <p:add sel="presence/note" pos="before"><tuple id="ert4773">
        <status>
         <basic>open</basic>
        </status>
        <contact priority="0.4">mailto:pep@example.com</contact>
        <note xml:lang="en">This is a new tuple inserted
              between the last tuple and note element</note>
       </tuple>

       </p:add>
       <p:replace sel="*/tuple[@id='r1230d']/status/basic/text()"
        >open</p:replace>

       <p:remove sel="*/r:person/r:status/r:activities/r:busy"/>

       <p:replace sel="*/tuple[@id='cg231jcr']/contact/@priority"
        >0.7</p:replace>

      </p:pidf-diff>
```

```
(M4): Compositor -> PUA

     SIP/2.0 200 OK
     ...
     SIP-ETag: 18764920981476
     Expires: 3600
     Content-Length: 0
```

7. Acknowledgements

   The authors would like to thank Atle Monrad, Christian Schmidt,
   George Foti, Fridy Sharon-Fridman, and Avshalom Houri for review
   comments.

8. References

8.1. Normative References

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
                  Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC3903]   Niemi, A., "Session Initiation Protocol (SIP)
                  Extension for Event State Publication", RFC 3903,
                  October 2004.

   [RFC3863]   Sugano, H., Fujimoto, S., Klyne, G., Bateman, A.,
                  Carr, W., and J. Peterson, "Presence Information Data
                  Format (PIDF)", RFC 3863, August 2004.

   [RFC3261]   Rosenberg, J., Schulzrinne, H., Camarillo, G.,
                  Johnston, A., Peterson, J., Sparks, R., Handley, M.,
                  and E. Schooler, "SIP: Session Initiation Protocol",
                  RFC 3261, June 2002.

   [RFC5262]   Lonnfors, M., Costa-Requena, J., Leppanen, E., and H.
                  Khartabil, "Presence Information Data Format (PIDF)
                  Extension for Partial Presence", RFC 5262, September
                  2008.

   [RFC5261]   Urpalainen, J., "An Extensible Markup Language (XML)
                  Patch Operations Framework Utilizing XML Path Language
                  (XPath) Selectors", RFC 5261, September 2008.

8.2. Informative References

   [RFC2778]   Day, M., Rosenberg, J., and H. Sugano, "A Model for
                   Presence and Instant Messaging", RFC 2778,
                   February 2000.

   [RFC4479]   Rosenberg, J., "A Data Model for Presence", RFC 4479,
                   July 2006.

   [RFC4480]   Schulzrinne, H., Gurbani, V., Kyzivat, P., and J.
                   Rosenberg, "RPID: Rich Presence Extensions to the
                   Presence Information Data Format (PIDF)", RFC 4480,
                   July 2006.

   [RFC4481]   Schulzrinne, H., "Timed Presence Extensions to the
                   Presence Information Data Format (PIDF) to Indicate
                   Status Information for Past and Future Time
                   Intervals", RFC 4481, July 2006.

   [RFC4482]   Schulzrinne, H., "CIPID: Contact Information for the
                   Presence Information Data Format", RFC 4482,
                   July 2006.

   [RFC5196]   Lonnfors, M. and K. Kiss, "Session Initiation Protocol
                   (SIP) User Agent Capability Extension to Presence
                   Information Data Format (PIDF)", RFC 5196, September
                   2008.

   [W3C.REC-xml]   Bray, T., Paoli, J., Sperberg-McQueen, C., and E.
                   Maler, "Extensible Markup Language (XML) 1.0 (2nd
                   ed)", W3C REC-xml, October 2000,
                   <http://www.w3.org/TR/REC-xml>.

Authors' Addresses

   Aki Niemi
   Nokia
   P.O. Box 407
   NOKIA GROUP, FIN 00045
   Finland

   Phone: +358 71 8008000
   EMail: aki.niemi@nokia.com

   Mikko Lonnfors
   Nokia
   Itamerenkatu 11-13
   Helsinki
   Finland

   Phone: +358 71 8008000
   EMail: mikko.lonnfors@nokia.com

   Eva Leppanen
   Individual
   Lempaala
   Finland

   EMail: eva.leppanen@saunalahti.fi