Presence Traffic Optimization Techniques[1]

Vishal Kumar Singh, Henning Schulzrinne
Department of Computer Science, Columbia University
{*vs2140, hgs*}@cs.columbia.edu

Markus Isomaki
Nokia
{*markus.isomaki*}@nokia.com

Piotr Boni
Verizon Communications
{*piotr.boni*}@verizon.com

28th Oct 2006

Abstract: With the growth of presence-based services, it is important to provision the network to support high traffic and load generated by presence services. Presence event distribution systems amplify a single incoming PUBLISH message into possibly numerous outgoing NOTIFY messages from the server. This can increase the network load on inter-domain links and can potentially disrupt other QoS-sensitive applications. In this document, we present existing as well as new techniques that can be used to reduce presence traffic both in inter-domain and intra-domain scenarios. Specifically, we propose two new techniques: sending common NOTIFY for multiple watchers and batched notifications. We also propose some generic heuristics that can be used to reduce network traffic due to presence.

1 Introduction

SIMPLE defines a set of specifications which describe how SIP-specific event notification can be used to manage and distribute user's presence information. SIMPLE-based presence systems distribute every incoming presence update (PUBLISH) to all the watchers of the presentity. Hence, the incoming traffic is amplified and distributed. The degree of amplification depends on the average number of watchers per presentity.

Consider the following example to understand the impact of deploying presence. A single SIP [2] call roughly generates six messages: INVITE, 180 Ringing, 200 OK, ACK, BYE, 200 OK. For every call, all the buddies of the two parties in the call can potentially be notified on 'in call' event. For each participant and for each buddy, a NOTIFY message and a 200 OK response message is generated. Assuming that each presentity has 25 watchers, the number of SIP messages generated because of presence = 2 (participants) x 2 (200 OK) x 25 (NOTIFY) =100 messages per call. We can add another 100 messages for notifications at the end of the call. This example shows how much of presence traffic can be generated just because of SIP calls as source of presence. Additionally, for a

[1] This work is supported by Verizon Labs.

mobile user, each location update can potentially generate large number of NOTIFY messages.

Presence would be an important service for personal handheld devices that often have limited network connectivity, especially when carried out of hotspot coverage. To protect user interests in wide-area connectivity where the user is charged based on the volume of traffic generated and to ensure that other applications are not affected by excessive presence traffic, there is a need to optimize presence traffic in access network, inter-domain and intra-domain scenarios. The problem has been well laid out in the SIMPLE problem statement draft [1]. In this document, we discuss different techniques that can be used to reduce the presence traffic and the scenarios in which these techniques are applicable. We also propose new techniques and heuristics which can be used to reduce presence traffic. Additionally, we do an analysis of what protocol enhancements are required to realize these proposals. We also analyze how much of bandwidth can be saved by deploying these techniques.

The remainder of this document is organized as follows: Section 2 presents the existing techniques which can be used for presence traffic optimization. Section 3 presents the proposed mechanism for reducing presence traffic. Section 4 presents the heuristics to reduce presence traffic in different scenarios. Section 5 discusses future work, followed by conclusion in Section 6.

2 Existing Presence Traffic Optimization Techniques

In this section we discuss the techniques which are proposed in existing specifications. We present a brief overview as well present the scenarios in which each of these can be useful.

2.1 RLS (Event Resource List Server)

RLS [14] is an extension to SIP-specific event notification [3] mechanism for subscribing to a list of resources. Instead of sending a SUBSCRIBE message for each resource individually, the watcher sends a single SUBSCRIBE to the server for a resource (a URI) which represents a list of resources. It then receives notifications when the state of any of the resources in the list changes. The server sends individual SUBSCRIBE messages to the resources in the list on behalf of the watcher.

This is mainly useful when the client is on a low bandwidth link, such as a GPRS link and sending individual SUBSCRIBE message to each resource is not feasible or costly. This technique does not reduce the size of PUBLISH, SUBSCRIBE or NOTIFY messages but only reduces the amount of SUBSCRIBE messages.

2.2 Conditional Subscriptions using Entity Tags

Conditional subscription [9] allows the subscriber to indicate in the subscription request that it is interested in receiving notification only if the state has changed since the previous notification. It is done based on an entity-tag in the SUBSCRIBE request, the tag being issued by the server during the previous NOTIFY. This technique reduces the NOTIFY messages during the subscription refresh requests.

This technique can be used in both inter-domain and intra-domain scenario's and would be beneficial irrespective of deployment domain and the deployment architecture. However, since it is only applicable during the subscription refreshes, the reduction achieved is not very high. It may turn out to be useful if the watcher is using a fetch (pull model) using SUBSCRIBE with Expires=0.

2.3 Watcher Filtering

A client can use watcher filtering [5] to specify conditions when it wants to receive notifications and what should be the content of the notifications. The subscriber can include these rules in the SUBSCRIBE message body.

This mechanism is mainly useful to the watchers using mobile wireless access devices. This mechanism can reduce both the size of the messages as well as the number of messages depending upon the watcher filter. However, the benefit of this mechanism to reduce presence traffic depends on client's applications capability to specify watcher filters.

2.4 Compression (SIGCOMP and Presence Dictionary)

It has been experimentally found that compression ratio using SIGCOMP [6] for SIP INVITE (without using static dictionary) is on the order of 0.36 for the initial INVITE-200 OK handshake and 0.30 for the following handshake. Compression ratio is defined as ratio of compressed message size to the uncompressed message size. For presence messages, we can expect similar compression ratio's for the initial NOTIFY-200 OK. However, we need to further investigate the actual compression ratio achievable by presence message flows using SIGCOMP with the presence static dictionary [7].

2.5 Partial PUBLISH and NOTIFY

Partial publication [10] is a mechanism using which the presence agent can send only parts of a presence document has changed since the previous update. This is done by sending a complete presence state initially and then sending only parts of presence document using the XML-patch based format specified in draft-simple-partial-pidf-format [17]. The watcher upon receiving the partial-pidf applies the changes to existing presence document to construct the complete state document. This technique does not reduces the number of messages and only affect the size of NOTIFY request body.

This mechanism can reduce the size of each message but does not reduce the message count. The analysis below shows that **use of partial-publication reduces the presence document size to ¼ times the original size on average** (averaged for all changes per day per presentity), assuming all watcher applications support partial-publication.

Following are the assumptions:

1. Average number of presence sources per presentity is 3. This will determine the number of tuples in the PIDF/RPID.
2. Type of each source, location or presence (PIDF, RPID, PIDF-LO) only.
3. Amount of change - This will determine the new size of PIDF/RPID after the initial complete state is sent.

We can assume that on an average the pidf-diff will be one-fourth of the size of the complete presence state. This is based on fact that out of 3 only 1 tuple changes every time. Since, we are transferring diff for only one tuple using XML-patch, the XML-patch body would be approximately half the size of the tuple body. So, the size of partial-publication body will be 1/4th to 1/6th of the complete presence body.

Gain Ratio = ¼ [This can vary from 1/4 to 1/6].

3 Proposed Presence Traffic Optimization Techniques

In this section, we propose additional mechanisms that can be used to reduce the presence traffic. Common NOTIFY for multiple watchers (Section 3.1) is useful in inter-domain scenarios, bundling of NOTIFY (Section 3.2) is useful both in inter-domain and access network scenarios. Common NOTIFY aggregates NOTIFY for multiple watchers and bundled NOTIFY aggregates NOTIFY from multiple presentities. Other techniques included use of timed-presence, On-demand presence and adapting the notification rate.

3.1 Common NOTIFY for Multiple Watchers

When multiple watchers from a domain (for example, domain B) SUBSCRIBE to a user in another domain (for example, domain A), a single NOTIFY per user can be sent from domain A to domain B's presence server (PS). The presence server in domain B can then distribute the NOTIFY messages to each of the watchers. This eliminates the need of sending individual NOTIFY messages from domain A's presence server to each watcher in domain B. We are assuming that presence server and resource list server (RLS) are co-located because of which NOTIFY messages are sent to presence server in domain B rather then delivered directly to the watchers of domain B.

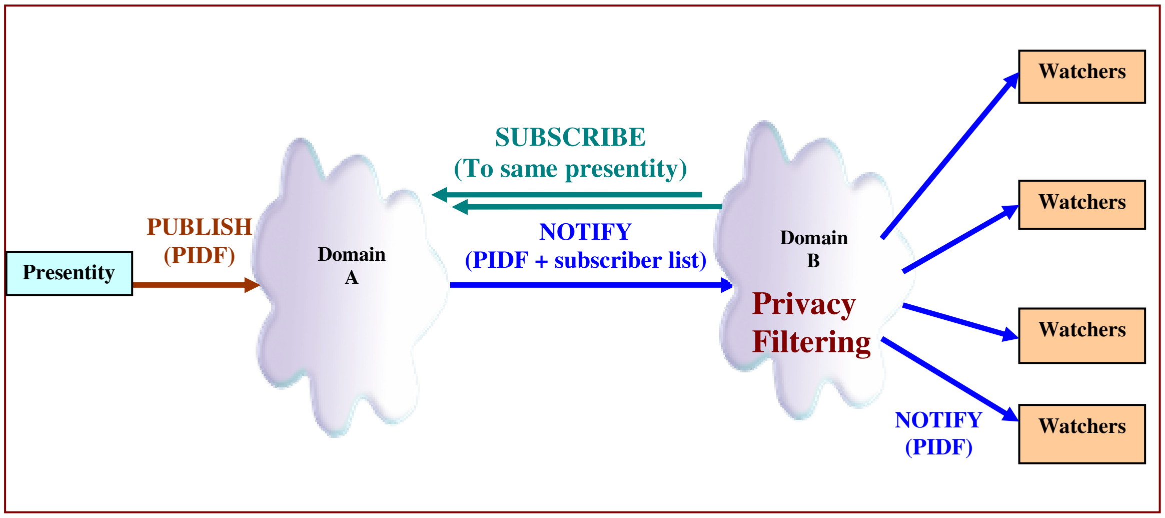


Fig. 1 Common NOTIFY for multiple watchers in an inter-domain scenario

The proposed mechanism is opposite to what a resource list server does, i.e., RLS sends SUBSCRIBE messages to a list of users based on a single SUBSCRIBE received from a watcher. In this case, the server sends a NOTIFY message to a list of watchers based on a single NOTIFY message received from another presence agent

There are three main issues namely, **privacy filtering, failure aggregation** and transfer of **watcher list** to watcher's domain so that it can distribute the NOTIFY messages. We discuss each of these issues in the subsequent sections.

3.1.1 Privacy filtering

Privacy filtering is typically done by presentity's presence server. We propose that presentity's privacy filtering task be handled by watcher domain's presence server, in this case domain B's presence server.

Since, these are two different domains; there are two possibilities about privacy filtering rules of the presentity as described below.

Per domain privacy filters: Presentity in domain A has same privacy filter rules for all the watchers in domain B. In other words, there is a domain level privacy filter specified by the presentity for users from domain B. Privacy filtering can be done by the presence server in domain A and a single NOTIFY can be sent from presence server in domain B.

Per watcher privacy filters: Presentity in domain A has different privacy filter rules for different watchers in domain B. Since, presentity in domain A has different privacy filtering rules for watchers from domain B, the privacy filter has to be applied by the presence server in domain B. Complete presence state information needs to be sent from the presentity's domain to watcher's domain. We discuss in Section 3.1.4 mechanisms to transfer privacy filters.

Further, we argue that delegating the task of privacy filtering doesn't compromise any additional privacy information when compared with normal operations. The model is very similar to e-mail trust model. Transfer of a single NOTIFY from presentity's domain to watcher's domain implies that the presence server in watcher's domain receives that information and can potentially distribute it to unauthorized watchers. [*We assumed RLS server co-located with the presence server to assume that watcher's domain presence server receives the NOTIFY*]. Thus, presentity implicitly trusts the presence server in its own domain as well as watcher's domain. The proposed mechanism extends such a trust to the presence server in domain B so that it performs the privacy filtering on behalf of presentity in domain A.

One potential issue is when presence server in domain A encrypts the presence document for each watcher using SMIME in which case the watcher domain PS cannot perform privacy filter. Hence, this kind of privacy filtering requires a layer 8 security negotiation between the presence servers of the two domains.

3.1.2 NOTIFY Failure Aggregation

The success or failure of NOTIFY message changes the subscription status of the watcher on the presentity's presence server. This requires that the domain B's presence server aggregates the success and failure responses for each watcher and send it to the presence server in domain A using another message. Alternatively, application level negative acknowledgement can be used.

3.1.3 Transferring the Watcher List

In order to distribute the NOTIFY message received from domain A, the watcher domain presence server requires the list of watchers from its domain for that presentity. We propose the following ways to achieve this.

3.1.3.1 Watcher List Sent in NOTIFY Message

The watcher list is sent from domain A's presence server to domain B's presence server in each NOTIFY message. The NOTIFY is then distributed to each watcher in the list. This has a disadvantage when the number of watcher's from domain B is very large, every NOTIFY message increases in size. An alternative could be sending the complete list initially and sending changes to the list using the XML-patch operations specified in partial-publication [17] and maintaining the list on presence server in domain B. Sending watcher-list and distributing it, is similar to multi recipient messages [11], SUBSCRIBE contained list [15] or Exploders.

3.1.3.2 Watcher List Obtained by Subscribing to WINFO Package

In this technique, the watcher's domain (domain B) presence server obtains the watcher list from domain A's PS. It also receives any changes to the watcher-list from domain A's PS by subscribing to the presentity with **presence.winfo event package**. The domain A's PS maintains and updates the watcher list as a part of its normal operation. The updates are sent whenever watcher list changes. They contain information about watchers from domain B only.

3.1.3.3 Watcher-List Created on Subscriber Side Presence Server

The watcher domain presence server maintains and updates the list of watchers per presentity based on the SUBSCRIBE requests from these watchers. Such a list is like a resource list of watchers per presentity in watcher's domain built dynamically based on SUBSCRIBE request which are not directly sent to presentity's PS.

3.1.4 Transferring Privacy Filter to Watcher Domain Presence Server

In section 3.1.1.2, we mentioned that presentity may have different privacy filters for different watchers from domain B. Thus, in order to delegate the task of privacy filtering to the watcher domain presence server, we need to transfer the privacy filter rules specific to the watchers of watcher's domain PS. We propose a SIMPLE-based mechanism to achieve this. The mechanism uses XCAP and SUBSCRIBE to do this.

Filter Download Trigger

Privacy filters can be downloaded initially when the watcher subscribes to the presentity. There can be an implicit subscription from watcher's domain presence server to presentity domain's presence server for downloading the filter rules. Whenever there is a change in the privacy filter rules for any of existing watchers or a new watcher is added, presentity domain (domain A's) updates watcher domain (domain B's) presence server to download the filter rules. The update from domain A's PS can be a NOTIFY message with **"Event = Filter Download"**. This can either contain the URL to download the new filter rules or itself contain the new rules. An XCAP-diff event can also be used. Additionally, the filters should be cleared from watcher domain either based on time-out or using a retention-expiry interval.

3.1.5 Changes Required to SIMPLE for Common NOTIFY

Following is summary of changes or additions to the SIMPLE protocol:

1. Mechanism for security association between the domains of the presence servers.
2. Mechanism for the presence server in different domains to become aware of each other's capability of handling Common NOTIFY protocol for presence. This can be achieved using OPTIONS message or an extension header field in the SUBSCRIBE.
3. Mechanism to transfer the watcher list from presentity's server to watcher's server. The watcher list only includes watchers of domain B.
4. Mechanism to transfer privacy filters of presentities to PS in domain B.
5. Mechanism to aggregate failures of NOTIFY to the watchers and feedback to the presentity's presence server.

One practical issue could be how much of CPU resources are used from watcher's domain presence server for doing privacy filtering for presentities of some other domain. The advantage of lower incoming traffic can compensate for additional CPU resources to do privacy filtering. Another issue is that transferring privacy filter across presence domain's it reveals users preferences for different watchers. Sending user's buddy list is can also be a privacy concern. However, considering that the presence server in domain B is aware of presentities complete presence information, these may be less critical.

3.1.6 Performance Impact

The proposed scheme reduces the number of NOTIFY/200 OK messages between different presence domains. Following is the number of NOTIFY and 200OK messages that will be sent across domain A and B, only considering watchers from domain B and presentities in domain A.

Average number of watchers per presentity per domain = W_c (watcher count)

Total number of presentity (per domain) = P_n

Number of watcher domains = D

Total number of watchers = D x W_c

NOTIFY/200OK = W_c x D x P_n x 2 (200 OK) x Rate of change of presence

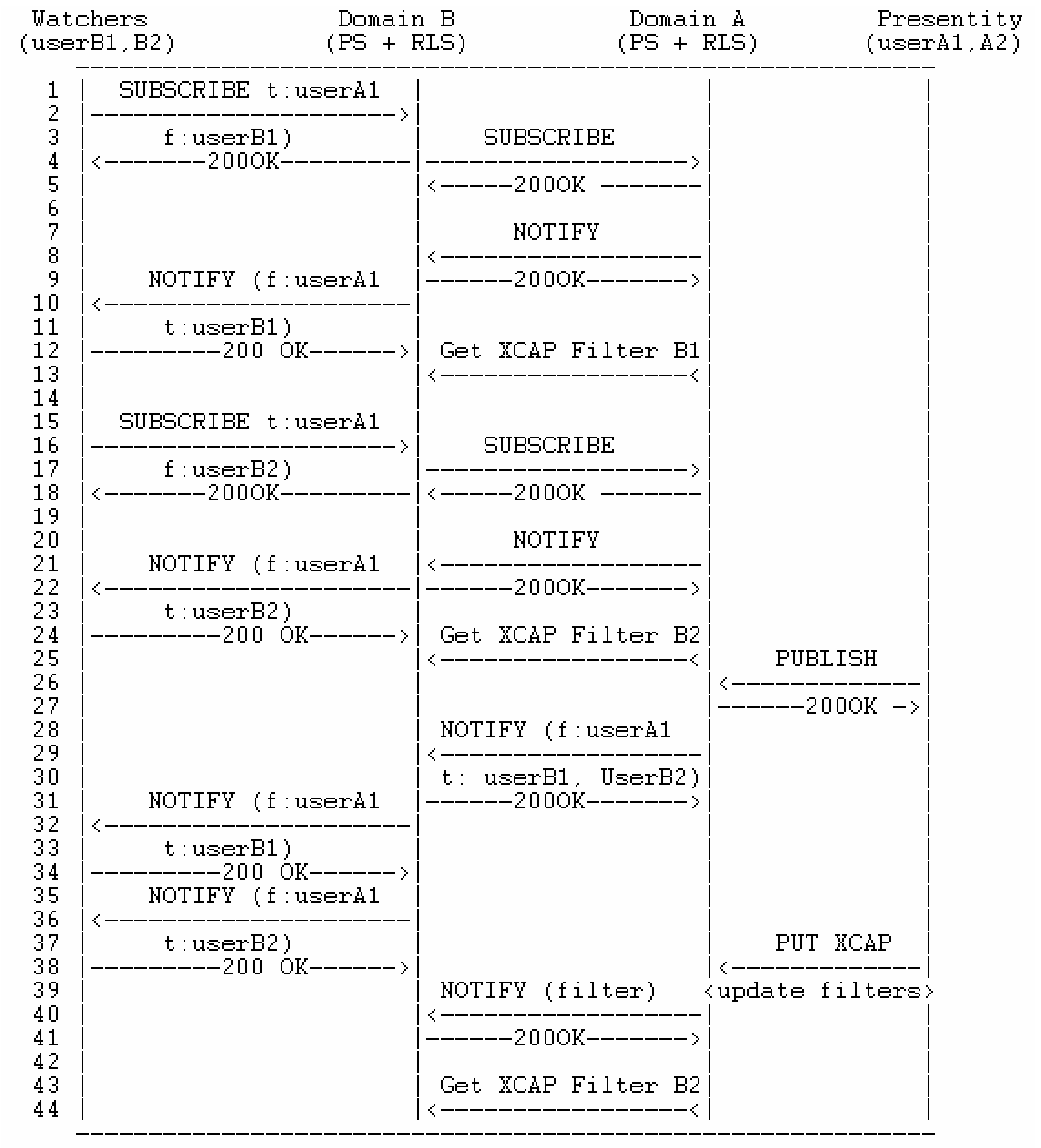
In an inter-domain scenario this is multiplied by 2 for each domain. With the proposed scheme: Average Number of NOTIFY for all watchers per presentity = 1 as a single NOTIFY will be sent. **W_c =1**

NOTIFY/200OK = D x P_n x rate x 2 (200 OK)

Total gain = 1/ W_c where W_c is number of watchers per domain

3.1.7 Message Flow Diagram

The message flow diagram below assumes watchers in domain B (userB1 and userB2), presentities in domain A (userA1, userA2). Watchers send SUBSCRIBE to their SIP proxy server which sends it to the presence server. We assume PS and RLS are collocated so that SUBSCRIBE is sent from presence server to the presentity's presence server and NOTIFY from presentity's PS is received by the presence server in watcher's domain and then distributed to the watchers. *The t: and f: are for To and From fields in SIP header.*



```
  Watchers              Domain B            Domain A          Presentity
 (userB1,B2)           (PS + RLS)          (PS + RLS)         (userA1,A2)
    -----------------------------------------------------------------------
  1 |  SUBSCRIBE t:userA1   |                    |                   |
  2 |---------------------->|                    |                   |
  3 |      f:userB1)        |     SUBSCRIBE      |                   |
  4 |<--------200OK---------|------------------->|                   |
  5 |                       |<-----200OK --------|                   |
  6 |                       |                    |                   |
  7 |                       |       NOTIFY       |                   |
  8 |                       |<-------------------|                   |
  9 |    NOTIFY (f:userA1   |------200OK-------->|                   |
 10 |<----------------------|                    |                   |
 11 |      t:userB1)        |                    |                   |
 12 |---------200 OK------->| Get XCAP Filter B1 |                   |
 13 |                       |<------------------<|                   |
 14 |                       |                    |                   |
 15 |  SUBSCRIBE t:userA1   |                    |                   |
 16 |---------------------->|     SUBSCRIBE      |                   |
 17 |      f:userB2)        |------------------->|                   |
 18 |<--------200OK---------|<-----200OK --------|                   |
 19 |                       |                    |                   |
 20 |                       |       NOTIFY       |                   |
 21 |    NOTIFY (f:userA1   |<-------------------|                   |
 22 |<----------------------|------200OK-------->|                   |
 23 |      t:userB2)        |                    |                   |
 24 |---------200 OK------->| Get XCAP Filter B2 |                   |
 25 |                       |<------------------<|     PUBLISH       |
 26 |                       |                    |<------------------|
 27 |                       |                    |------200OK ->     |
 28 |                       | NOTIFY (f:userA1   |                   |
 29 |                       |<-------------------|                   |
 30 |                       | t: userB1, UserB2) |                   |
 31 |    NOTIFY (f:userA1   |------200OK-------->|                   |
 32 |<----------------------|                    |                   |
 33 |      t:userB1)        |                    |                   |
 34 |---------200 OK------->|                    |                   |
 35 |    NOTIFY (f:userA1   |                    |                   |
 36 |<----------------------|                    |                   |
 37 |      t:userB2)        |                    |     PUT XCAP      |
 38 |---------200 OK------->|                    |<------------------|
 39 |                       | NOTIFY (filter)    <update filters>    |
 40 |                       |<-------------------|                   |
 41 |                       |------200OK-------->|                   |
 42 |                       |                    |                   |
 43 |                       | Get XCAP Filter B2 |                   |
 44 |                       |<------------------<|                   |
    -----------------------------------------------------------------------
```

Fig. 2. Message flow diagram showing Common NOTIFY for watchers in a domain.

We can see in Fig. 2 that a single NOTIFY from userA1@domainA.com is sent to watchers {userB1, userB2}@domainB.com. Also, we can see that a change in privacy filter rule causes a NOTIFY which triggers an XCAP-based download of privacy filtering rules by domain B's PS.

3.1.8 Example Messages

The following NOTIFY message contains the list of watchers and the presence document of the presentity. The RLS /presence server in B will distribute it to all the watchers in the list.

```
NOTIFY sip:rlserver.domainB.com SIP/2.0
Via: SIP/2.0/TCP rlsserver.domainA.com;branch=z9hG4bK4EPlfSFQK1
Max-Forwards: 70
From: <sip:userA1@domainA.com>;tag=zpNctbZq
To: <sip:rlsserver@domainB.com>;tag=ie4hbb8t
Call-ID: cdB34qLToC@domainA.com
CSeq: 997935769 NOTIFY
Contact: <sip:rlsserver.domainA.com>
Event: presence
Subscription-State: active;expires=7200
Content-Type: multipart/related;type="resource-lists+xml";
       start="<2BEI83@rlsserver.domainA.com >";
       boundary=" tuLLl3lDyPZX0GMr2YOo "
Content-Length: 2014

--tuLLl3lDyPZX0GMr2YOo
Content-Transfer-Encoding: binary
Content-ID: <2BEI83@rlsserver.domainA. com>
Content-Type: application/resource-lists+xml; charset="UTF-8"
   <?xml version="1.0" encoding="UTF-8"?>
   <resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <list>
       <entry uri="sip:userB1@domainB.com" />
       <entry uri="sip:userB2@domainB.com" />
     </list>
   </resource-lists>

--tuLLl3lDyPZX0GMr2YOo
Content-Transfer-Encoding: binary
Content-ID: <2BEI83@rlsserver.domainA.example.com >
Content-Type:type="application/pidf+xml;charset="UTF-8"
       start="<AAAA@rlsserver.domainB.example.com >";
       boundary=" TfZxoxgAvLqgj4wRWPDL"


--TfZxoxgAvLqgj4wRWPDL

   <?xml version="1.0" encoding="UTF-8"?>
   <presence xmlns="urn:ietf:params:xml:ns:pidf"
       entity="sip:userA1@domainA.com">
     <tuple id="z98075">
       <status>
         <basic>closed</basic>
       </status>
     </tuple>
   </presence>
```

--TfZxoxgAvLqgj4wRWPDL--

3.2 Aggregation of NOTIFY Messages (Batched Notification)

When a watcher from a domain (for example domain B) SUBSCRIBE to multiple presentities in another domain (domain A), domain A's presence server can aggregate the notification messages and send them together as a single NOTIFY message to the presence server in domain B. The presence server in domain B can then deliver the message to the watcher or create individual NOTIFY messages for different watchers and send it to them. **This reduces the number of NOTIFY/ 200 OK messages on the inter-domain link as well as access network. This aggregation of NOTIFY can be done on per watcher or per domain basis**. The RLS specification describes aggregation and throttling however, leaves it open to the implementers.

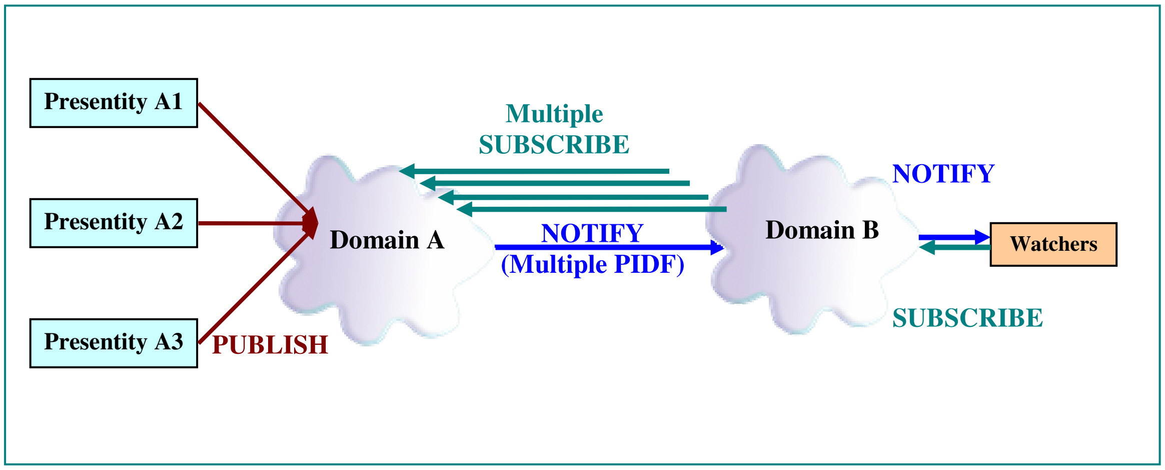


Fig. 3 Batched Notifications

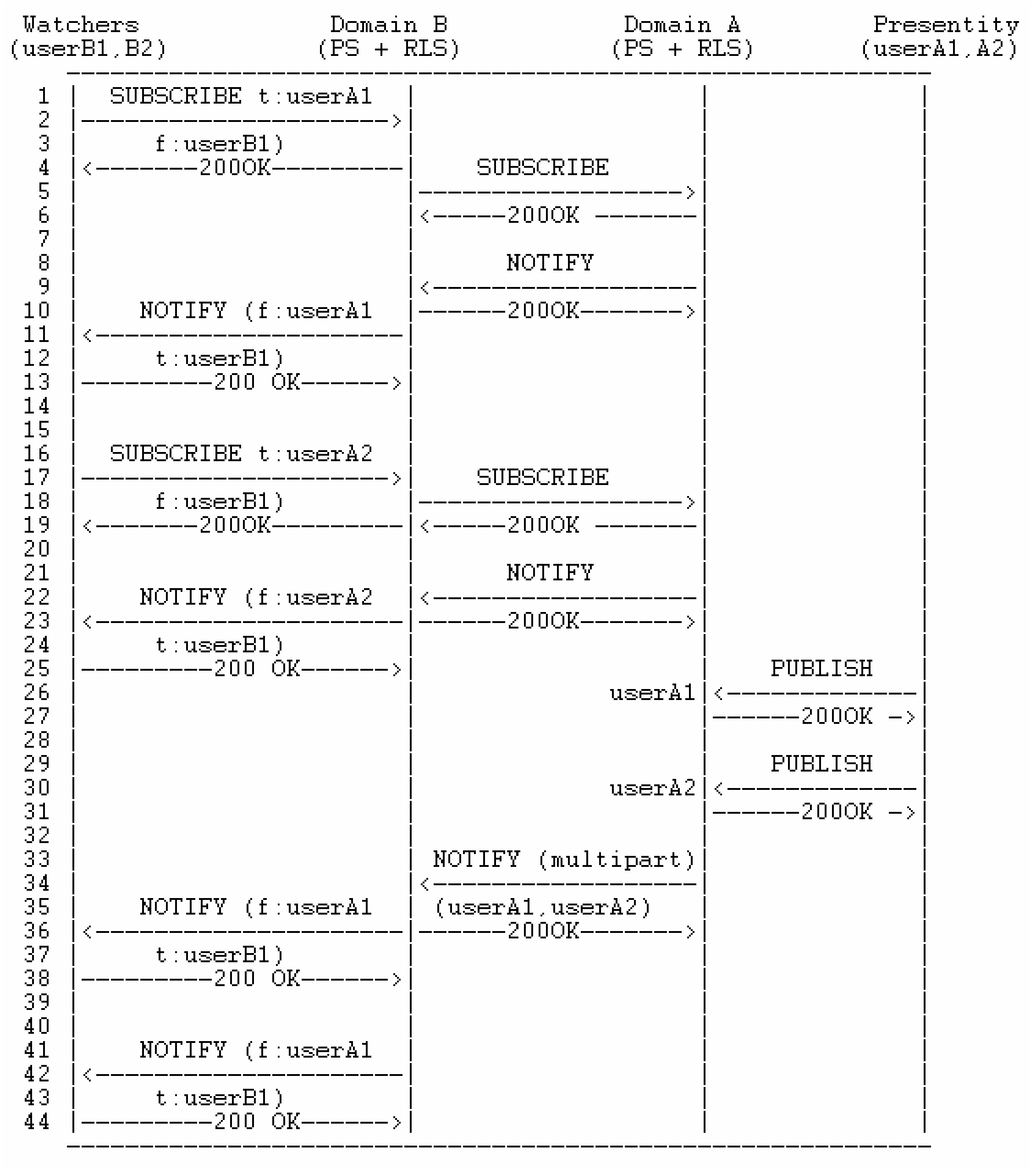
One problem in aggregation is that presence status update for presentities may not occur simultaneously. Hence, in order to bundle the NOTIFY messages for each watcher or domain, the presence server may have to delay some of the NOTIFY messages. One approach to solve this issue could be that the **watcher specifies a tolerable delay** for receiving presence state update of the presentities. The watcher can specify this delay value using the watcher filtering mechanism or a SIP-header extension in the SUBSCRIBE message. The presence server in presentity's domain can hold the NOTIFY message only for the amount of time specified.

3.2.1 Extracting and Sending Individual NOTIFY from Aggregated NOTIFY Message Body

The aggregation of NOTIFY bodies originating from different presentities to a single NOTIFY body works on the basis of Multipart (MIME). Bundling of notification imply aggregating multiple NOTIFY bodies destined to a single watcher (or watcher domain) into a single NOTIFY and delivered to watcher domain presence server. If all the NOTIFY messages are destined to a single watcher, the watcher domain presence server delivers the message directly. Otherwise, the server extracts multiple presence bodies (PIDF) from the received NOTIFY message. Each presence document (PIDF) contains an entity field which uniquely identifies the presentity; hence, there is no dependency on SIP headers to construct individual NOTIFY messages for delivering them to watchers. Delivering bundled NOTIFY messages to watchers reduces the traffic on access network also.

3.2.2 Message Flow Diagram

The message flow diagram in Fig. 4 assumes watchers in domain B (userB1, userB2) and presentities in domain A (userA1, userA2). We can see that when userA1 and userA2 send PUBLISH, a single NOTIFY is sent from domain A to domain B, which is converted to individual NOTIFY messages by presence server at domain B.



```
 Watchers               Domain B             Domain A          Presentity
(userB1,B2)            (PS + RLS)           (PS + RLS)        (userA1,A2)
    ------------------------------------------------------------
  1 |  SUBSCRIBE t:userA1   |                    |              |
  2 |---------------------->|                    |              |
  3 |      f:userB1)        |                    |              |
  4 |<-------200OK----------|    SUBSCRIBE       |              |
  5 |                       |------------------->|              |
  6 |                       |<-----200OK --------|              |
  7 |                       |                    |              |
  8 |                       |      NOTIFY        |              |
  9 |                       |<-------------------|              |
 10 |    NOTIFY (f:userA1   |------200OK-------->|              |
 11 |<----------------------|                    |              |
 12 |      t:userB1)        |                    |              |
 13 |---------200 OK------->|                    |              |
 14 |                       |                    |              |
 15 |                       |                    |              |
 16 |  SUBSCRIBE t:userA2   |                    |              |
 17 |---------------------->|    SUBSCRIBE       |              |
 18 |      f:userB1)        |------------------->|              |
 19 |<-------200OK----------|<-----200OK --------|              |
 20 |                       |                    |              |
 21 |                       |      NOTIFY        |              |
 22 |    NOTIFY (f:userA2   |<-------------------|              |
 23 |<----------------------|------200OK-------->|              |
 24 |      t:userB1)        |                    |              |
 25 |---------200 OK------->|                    |   PUBLISH    |
 26 |                       |              userA1|<-------------|
 27 |                       |                    |------200OK ->|
 28 |                       |                    |              |
 29 |                       |                    |   PUBLISH    |
 30 |                       |              userA2|<-------------|
 31 |                       |                    |------200OK ->|
 32 |                       |                    |              |
 33 |                       | NOTIFY (multipart) |              |
 34 |                       |<-------------------|              |
 35 |    NOTIFY (f:userA1   | (userA1,userA2)    |              |
 36 |<----------------------|------200OK-------->|              |
 37 |      t:userB1)        |                    |              |
 38 |---------200 OK------->|                    |              |
 39 |                       |                    |              |
 40 |                       |                    |              |
 41 |    NOTIFY (f:userA1   |                    |              |
 42 |<----------------------|                    |              |
 43 |      t:userB1)        |                    |              |
 44 |---------200 OK------->|                    |              |
    ------------------------------------------------------------
```

Fig. 4. Message flow diagram showing aggregation of NOTIFY messages

3.2.3 Subscription Termination and Failure Indication in NOTIFY Delivery

The ‘Subscription-state’ header in the NOTIFY message is used to indicate subscription termination to a watcher. Bundled notification doesn’t indicate subscription termination, hence, terminating NOTIFY messages cannot be sent using this mechanism.

Additionally, the notifier needs to know if the NOTIFY was delivered successfully or not. The subscription can be terminated if NOTIFY is not delivered successfully. The presence server in domain B should aggregate and send to PS in domain A the success or failure of NOTIFY messages.

3.2.4 Performance Impact

The advantage is observed when a single watcher subscribes to multiple presentities from another domain. The delay tolerance interval specified by the watcher should be good enough so that multiple NOTIFY messages can be bundled or aggregated.

The reduction in traffic can be seen under two scenarios, i.e., (i) when watcher logs in and subscribes to all the presentities. The NOTIFY from multiple presentities can be bundled and delivered as a single message to the watcher. (ii) In steady state, the gain can be calculated based on the delay tolerance interval, number of presentities to which a watcher is subscribed, probability of these presentities changing state in that interval. With increase in number of presentities, the probability that presentities will update presence state within a time difference of delay tolerance interval will increase and hence the inter domain traffic reduction (gain) will increase.

3.2.5 Example Messages

The following NOTIFY message contains presence documents of multiple presentities. In the example, all the presence documents are destined to a single watcher. But, this technique may work along with “Common NOTIFY for multiple watchers” described in Section 3.1.

```
NOTIFY sip:rlserver.domainB.com SIP/2.0
Via: SIP/2.0/TCP rlsserver.domainA.example.com;branch=z9hG4bK4EPlfSFQK1
Max-Forwards: 70
From: <sip:rlsserver@domainA.com>;tag=zpNctbZq
To: <sip:userA@domainB.com>;tag=ie4hbb8t
Call-ID: cdB34qLToC@ domainA.com
CSeq: 997935769 NOTIFY
Contact: <sip:rlsserver.domainA.com>
Event: presence
Subscription-State: active;expires=7200
Content-Type: multipart/related;type="rlmi+xml";
       start="<2BEI83@rlsserver.domainB.example.com >";
       boundary=" tuLLl3lDyPZX0GMr2YOo "
Content-Length: 2862

--tuLLl3lDyPZX0GMr2YOo
   Content-Transfer-Encoding: binary
   Content-ID: <2BEI83@rlsserver.domainB.example.com>
   Content-Type: application/pidf+xml;charset="UTF-8"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
    entity="sip:userA1@domainA.com">
  <tuple id="x823a4">
    <status>
      <basic>open</basic>
    </status>
    <contact priority="1.0">sip:joe@stockholm.example.org</contact>
  </tuple>
</presence>

--tuLLl3lDyPZX0GMr2YOo
Content-Transfer-Encoding: binary
Content-ID: <KKMDmv@stockholm.example.org>
Content-Type: application/pidf+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
    entity="sip:userA2@domainA.com">
  <tuple id="z98075">
    <status>
      <basic>closed</basic>
    </status>
  </tuple>
</presence>
--tuLLl3lDyPZX0GMr2YOo--
```

3.3 Timed Presence

Watchers may be interested in general availability information of certain presentities rather then getting notification for every status change of the presentity. For example, a manager may be interested in knowing if the employees under him are available or on vacation and he may not be interested in getting notification for every status change about them. Another example is that he is interested in knowing their vacation plans, customer visit plans, conference plans and long term availability information etc. This can be achieved using timed-presence [8].

An example of Timed-presence status from [8], (for sake of completeness) is below:

```
<presence xmlns="urn:ietf:params:xml:ns:pidf"
     xmlns:ts="urn:ietf:params:xml:ns:pidf:timed-status"
       entity="pres:someone@columbia.edu">
     <tuple id="c8dqui">
       <status>
         <basic>open</basic>
       </status>
       <ts:timed-status from="2006-11-04T10:20:00.000-05:00"
          until="2006-11-08T19:30:00.000-05:00">
          <ts:basic>closed</ts:basic>
       </ts:timed-status>
       <contact>sip:Vishal@cs.columbia.edu</contact>
     </tuple>
     <note>I'll be in San Diego, IETF meeting</note>
   </presence>
```

Timed-presence notification can be can be used to reduce presence by automatic switching the subscription on or off, based on timed-status obtained from timed-presence notification. However, with current watcher filtering specification it is not straightforward to automatically enable or disable notifications based on calendar information from timed-presence. Watchers cannot specify a watcher filter indicating not to send NOTIFY based on timed-status as it would require them to know the 'from'/'until' attribute in <timed-status> before hand. Watcher filtering specification does not allow watchers to specify filter rules to disable notifications based on comparison of timestamps.

A watcher application upon obtaining the <timed status> can specify a watcher filter using the 'from' and 'until' attribute in the received <timed-status>, indicating the server not to send a NOTIFY unless the <timed-status> or 'from' or 'until' attribute changes. A watcher should not blindly un-subscribe for the time specified in the <timed-status> because presentity may update the time-status and watcher may not be aware of this. Hence, watcher must specify a watcher filter which triggers a notification upon changes in elements of <timed-status>, after it has received the first <timed-status>. Once the interval for the received <timed-status> is over, the watcher application removes the filter and starts receiving notifications in a normal manner. Alternatively, differential notification can be used to know about changes in the timed-presence.

From the above discussion, it is clear that watcher filtering specification requires enhancements for timestamp based watcher filters.

4 Heuristics to Reduce Presence Traffic

In this section we propose two heuristics which can be implemented on the watcher applications or on the presence server to reduce unnecessary presence traffic.

4.1 On-demand Presence (Fetch or Pull Model)

Watchers do not want to get notified about every presence update of all the buddies at all times. Watchers may be interested in regularly receiving presence updates for some of the buddies. But for other buddies, watchers may only want to know their presence information when they want to call. This can be labeled as **on-demand presence** and can be accomplished by using fetch based SUBSCRIBE with expiration interval set to zero.

This approach does not require any changes in the SIMPLE protocol. However, it requires a mechanism in the watcher application to enable watchers to indicate that they are not interested in regular presence updates; rather they only require presence information when starting a new session. Typical use for on-demand presence is when a user initiates a session with someone with whom he does not interact on a regular basis and hence does not need to maintain a subscription.

Examples may include services, where presence status does not have to be visually exposed or known to a watcher all of the time. For example, a cell-phone associated watcher may need presence updates only when the cell-phone application (e.g., phone book) runs in the foreground on the device. Another example is a presence-based call routing in telephony, where - before the call is delivered - a watcher issues a fetch-based SUBSCRIBE to learn whether and where the callee is available.

On-demand presence model can be useful in reducing traffic both in inter-domain and intra-domain scenarios.

4.1.1 Performance Impact

Using this scheme, the number of NOTIFY messages to be sent for every presence update will be reduced to only a few watchers – those watchers who want to get presence updates always. From the example given in the introduction, we can see that it is not required to generate NOTIFY messages to all the watchers for each call.

4.2 Adapting the Notification Rates

The rate of notification can be adjusted based on statistical information about past multimedia sessions with buddies. This can be initiated by the client or can be automatically done by the server as server can procure such information based on stored call and text session information.

As a matter of fact 60-70% of the calls/IM messages are sent to 20% of the buddies [Reference required, Observation based on call detail records of my friends]. Nearly 50% of the buddies are called rarely. This may include buddies from old office, old college, and old city who are present in the buddy list but are not contacted actively. Based on such information the presence server or the client can adapt the subscription rate and use the fetch model for such buddies.

5 Future Work

We plan to evaluate in more detail the impact of each proposal on inter-domain presence traffic. We also plan to analyze the implication of these techniques on access networks. Additionally, we plan to see the amount of traffic which may be generated because of SIMPLE standard changes to realize inter-domain presence traffic scaling.

6 Conclusion

In this document we surveyed existing mechanisms to optimize presence traffic. We analyzed these mechanisms. We also proposed additional mechanisms to reduce presence traffic in inter-domain scenarios, analyzed performance gain if such mechanisms are used. Additionally we analyzed of what changes or additions to existing SIMPLE standard are required to realize the proposed mechanisms. We also proposed additional heuristics which can be employed to reduce unnecessary presence traffic.

7 References

1. Rang T., Houri A., Aoki E., Problem Statement for SIP/SIMPLE, draft-rang-simple-problem-statement-01.txt (work in progress), June 2006.

2. 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.

3. Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002
4. Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.
5. Khartabil, H., Leppanen E., Lonnfors M., Costa-Requena J., Functional Description of Event Notification Filtering, RFC 4660, Sept 2006.
6. Garcia-Martin, M., Bormann, C., Ott, J., Price, R., Roach A., "The Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Static Dictionary for Signaling Compression (SigComp)", RFC 3485, February 2003.
7. Garcia-Martin M., The Presence-specific Dictionary for the Signaling Compression (Sigcomp) Framework, draft-garcia-simple-presence-dictionary-00 (work in progress), June 2006.
8. 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.
9. Niemi, A., "An Extension to Session Initiation Protocol (SIP) Events for Issuing Conditional Subscriptions", draft-niemi-sip-subnot-etags-01 (work in progress), June 2006.
10. Niemi A., Lonnfors, M., Leppanen E., "Publication of Partial Presence Information", draft-ietf-simple-partial-publish-05 (work in progress), July 2006.
11. Garcia-Martin M., Camarillo G., Multiple-Recipient MESSAGE Requests in the Session Initiation Protocol, draft-ietf-sip-uri-list-message-00 (work in progress), Sept. 2006.
12. Rosenberg J., A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP), RFC 3857, Aug 2004.
13. Rosenberg J., The Extensible Markup Language (XML) Configuration Access Protocol, draft-ietf-simple-xcap-10 (work in progress), May 2006.
14. Roach A., Campbell B., Rosenberg J., A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists, RFC 4662, August 2006.
15. Camarillo G., Roach A., Levin O., Subscriptions to Request-Contained Resource Lists in the Session Initiation Protocol (SIP) , draft-ietf-sip-uri-list-subscribe-00 (work in progress), Sept 2006.
16. Singh V., Schulzrinne H., A Survey of Security Issues and Solutions in Presence, Columbia University Technical Report, cucs-019-06, Feb 2006.
17. Lonnfors M.,Leppanen E., Khartabil H., Urpalainen J., Presence Information Data format (PIDF) Extension for Partial Presence, draft-ietf-simple-partial-pidf-format-07 (work in progress), July 2006.