SIMPLE	J. Rosenberg
Internet-Draft	jdrosen.net
Intended status: Informational	A. Houri
Expires: November 13, 2010	IBM
	C. Smyth
	Avaya
	F. Audet
	Skype
	May 12, 2010

Models for Intra-Domain Presence and Instant Messaging (IM) Bridging
draft-ietf-simple-intradomain-federation-05

Abstract

Presence and Instant Messaging (IM) bridging involves the sharing of presence information and exchange of IM across multiple systems within a single domain. As such, it is a close cousin to presence and IM federation, which involves the sharing of presence and IM across differing domains. Presence and IM bridging can be the result of a multi-vendor network, or a consequence of a large organization that requires partitioning. This document examines different use cases and models for intra-domain presence and IM bridging. It is meant to provide a framework for defining requirements and specifications for presence and IM bridging. The document assumes SIP as the underlying protocl but many of the modles can fit other protocols as well.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction
2. Intra-Domain Bridging vs. Clustering
3. Use Cases for Intra-Domain Bridging
    3.1. Scale
    3.2. Organizational Structures
    3.3. Multi-Vendor Requirements
    3.4. Specialization
4. Considerations for Bridging Models
5. Overview of the Models
6. Partitioned
    6.1. Applicability
    6.2. Routing
        6.2.1. Centralized Database
        6.2.2. Routing Proxy
        6.2.3. Subdomaining
        6.2.4. Peer-to-Peer
        6.2.5. Forking
        6.2.6. Provisioned Routing
    6.3. Policy
    6.4. Presence Data
    6.5. Conversation Consistency
7. Exclusive

7.1. Routing
7.1.1. Centralized Database
7.1.2. Routing Proxy
7.1.3. Subdomaining
7.1.4. Peer-to-Peer
7.1.5. Forking
7.2. Policy
7.3. Presence Data
7.4. Conversation Consistency
8. Unioned
8.1. Hierarchical Model
8.1.1. Routing
8.1.2. Policy and Identity
8.1.2.1. Root Only
8.1.2.2. Distributed Provisioning
8.1.2.3. Central Provisioning
8.1.2.4. Centralized PDP
8.1.3. Presence Data
8.1.4. Conversation Consistency
8.2. Peer Model
8.2.1. Routing
8.2.2. Policy
8.2.3. Presence Data
8.2.4. Conversation Consistency
9. More about Policy
10. Acknowledgements
11. Security Considerations
12. IANA Considerations
13. Informative References
§ Authors' Addresses

1. Introduction

TOC

Presence refers to the ability, willingness and desire to communicate across differing devices, mediums and services [RFC2778] (Day, M., Rosenberg, J., and H. Sugano, “A Model for Presence and Instant Messaging,” February 2000.). Presence is described using presence documents [RFC3863] (Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, “Presence Information Data Format (PIDF),” August 2004.) [RFC4479] (Rosenberg, J., “A Data Model for Presence,” July 2006.), exchanged using a Session Initiation Protocol (SIP) [RFC3261] (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” June 2002.) based event package [RFC3856] (Rosenberg, J., “A Presence Event Package for the Session Initiation

Protocol (SIP)," August 2004.). Similarly, instant messaging refers to the exchange of real-time text-oriented messaging between users. SIP defines two mechanisms for IM - pager mode [RFC3428] (Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging," December 2002.) and session mode [RFC4975] (Campbell, B., Mahy, R., and C. Jennings, "The Message Session Relay Protocol (MSRP)," September 2007.).

Presence and Instant Messaging (IM) bridging involves the sharing of presence information and exchange of IM across multiple systems within a single domain. As such, it is a close cousin to presence and IM federation [RFC5344] (Houri, A., Aoki, E., and S. Parameswar, "Presence and Instant Messaging Peering Use Cases," October 2008.), which involves the sharing of presence and IM across differing domains.

For example, consider the network of Figure 1 (Inter-Domain Presence Model), which shows one model for inter-domain presence federation. In this network, Alice belongs to the example.org domain, and Bob belongs to the example.com domain. Alice subscribes to her buddy list on her presence server (which is also acting as her Resource List Server (RLS) [RFC4662] (Roach, A., Campbell, B., and J. Rosenberg, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists," August 2006.)), and that list includes bob@example.com. Alice's presence server generates a back-end subscription on the federated link between example.org and example.com. The example.com presence server authorizes the subscription, and if permitted, generates notifications back to Alice's presence server, which are in turn passed to Alice.

```
       .............................     ..............................
       .                           .     .                            .
       .                           .     .                            .
       .     alice@example.org     .     .      bob@example.com       .
       .      +------------+   SUB .     .      +------------+        .
       .      |            |   Bob .     .      |            |        .
       .      |  Presence  |------------------->|  Presence  |        .
       .      |   Server   |       .     .      |   Server   |        .
       .      |            |       .     .      |            |        .
       .      |            |<-------------------|            |        .
       .      |            |     NOTIFY  .      |            |        .
       .      +------------+       .     .      +------------+        .
       .          ^    |           .     .             ^              .
       .      SUB |    |           .     .             |PUB           .
       .    Buddy |    |NOTIFY     .     .             |              .
       .     List |    |           .     .             |              .
       .          |    |           .     .             |              .
       .          |    V           .     .             |              .
       .         +-------+         .     .        +-------+           .
       .         |       |         .     .        |       |           .
       .         |       |         .     .        |       |           .
       .         |       |         .     .        |       |           .
       .         +-------+         .     .        +-------+           .
       .                           .     .                            .
       .           Alice's         .     .           Bob's            .
       .             PC            .     .            PC              .
       .                           .     .                            .
       .............................     ..............................

                example.org                       example.com
```

Figure 1: Inter-Domain Presence Model

Similarly, inter-domain IM federation would look like the model shown in Figure 2 (Inter-Domain SIP based IM Model):

```
.............................     ..............................
.                           .     .                            .
.                           .     .                            .
.     alice@example.org     .     .      bob@example.com       .
.      +------------+   INV .     .      +------------+        .
.      |            |   Bob .     .      |            |        .
.      |            |------------------->|            |        .
.      |     IM     |       .     .      |      IM    |        .
.      |   Server   |       .     .      |    Server  |        .
.      |            |<------------------>|            |        .
.      |            |       IM           |            |        .
.      +------------+       Content      +------------+        .
.          ^    ^           .     .          ^    |            .
.   INVITE |    |           .     .   IM     |    |INV         .
.   Bob    |    | IM        .     .   Content|    |Bob         .
.          |    | Content   .     .          |    |            .
.          |    |           .     .          |    |            .
.          |    V           .     .          V    V            .
.         +-------+         .     .        +-------+           .
.         |       |         .     .        |       |           .
.         |       |         .     .        |       |           .
.         |       |         .     .        |       |           .
.         +-------+         .     .        +-------+           .
.                           .     .                            .
.           Alice's         .     .           Bob's            .
.             PC            .     .            PC              .
.                           .     .                            .
.............................     ..............................

         example.org                       example.com
```

Figure 2: Inter-Domain SIP based IM Model

In this model, example.org and example.com both have an "IM server". This would typically be a SIP proxy or B2BUA responsible for handling both the signaling and the IM content (as these are separate in the case of session mode). The IM server would handle routing of the IM along with application of IM policy.

Though both of these pictures show federation between domains, a similar interconnection - presence and IM bridging - can happen within a domain as well. We define intra-domain bridging as the interconnection of presence and IM servers within a single administrative domain. Typically, a single administrative domain often means the same DNS domain within the right hand side of the @-sign in the SIP URI.

This document considers the architectural models and different problems that arise when performing intra-domain presence and IM bridging. Though presence and IM are quite distinct functions, this document considers both since the architectural models and issues are common between the two. The document first clarifies the distinction between intra-domain bridging and clustering. It defines the primary issues that arise in intra-domain presence and IM bridging, and then goes on to define the three primary models for it - partitioned, unioned and exclusive.

This document doesn't make any recommendation as to which model is best. Each model has different areas of applicability and are appropriate in a particular deployment. The intent is to provide informative material and ideas on how this can be done.

---

2. Intra-Domain Bridging vs. Clustering

Intra-domain bridging is the interconnection of servers within a single domain. This is very similar to clustering, which is the tight coupling of a multiplicity of physical servers to realize scale and/or high availability. Consequently, it is important to clarify the differences. Note that clustering is out of scope for this document.

Firstly, clustering implies a tight coupling of components. Clustering usually involves proprietary information sharing, such as state sharing, which in turn are tightly bound with the internal implementation of the product. Intra-domain bridging, on the other hand, is a loose coupling. Although database replication may be used across bridged systems, it is not in the same level of stats replication that usually occurs in clusters.

Secondly, clustering most usually occurs amongst components from the same vendor. This is due to the tight coupling described above. Intra-domain bridging, on the other hand, can occur between servers from different vendors. As described below, this is one of the chief use cases for intra-domain bridging.

Thirdly, clustering is almost always invisible to users. Communications between users within the same cluster almost always have identical functionality to communications between users on the same server within the cluster. The cluster boundaries are invisible; indeed the purpose of a cluster is to build a system which behaves as if it were a single monolithic entity, even though it is not. Bridging, on the other hand, is often visible to users. There will frequently be loss of functionality when crossing a cluster. Though this is not a hard and fast rule, it is a common differentiator.

Fourthly, connections between federated and bridged systems almost always involve standards, whereas communications within a cluster often involves proprietary mechanisms. Standards are needed for bridging

because the systems can be from different vendors, and thus agreement is needed to enable interoperation.
Finally, a cluster will often have an upper bound on its size and capacity, due to some kind of constraint on the coupling between nodes in the cluster. However, there is typically no limit, or a much larger limit, on the number of bridged systems that can be put into a domain. This is a consequence of their loose coupling.
Though these rules are not hard and fast, they give general guidelines on the differences between clustering and intra-domain bridging.

---

3. Use Cases for Intra-Domain Bridging

TOC

There are several use cases that drive intra-domain bridging.

3.1. Scale

TOC

One common use case for bridging is an organization that is just very large, and their size exceeds the capacity that a single server or cluster can provide. So, instead, the domain breaks its users into partitions (perhaps arbitrarily) and then uses intra-domain bridging to allow the overall system to scale up to arbitrary sizes. This is common practice today for service providers and large enterprises.

3.2. Organizational Structures

TOC

Another use case for intra-domain bridging is a multi-national organization with regional IT departments, each of which supports a particular set of nationalities. It is very common for each regional IT department to deploy and run its own servers for its own population. In that case, the domain would end up being composed of the presence servers deployed by each regional IT department. Indeed, in many organizations, each regional IT department might end up using different vendors. This can be a consequence of differing regional requirements for features (such as compliance or localization support), differing sales channels and markets in which vendors sell, and so on.

TOC

3.3. Multi-Vendor Requirements

Another use case for intra-domain bridging is an organization that requires multiple vendors for each service, in order to avoid vendor lock in and drive competition between its vendors. Since the servers will come from different vendors, a natural way to deploy them is to partition the users across them. Such multi-vendor networks are extremely common in large service provider networks, many of which have hard requirements for multiple vendors.
Typically, the vendors are split along geographies, often run by different local IT departments. As such, this case is similar to the organizational division above.

3.4. Specialization

TOC

Another use case is where certain vendors might specialize in specific types of clients. For example, one vendor might provide a mobile client (but no desktop client), while another provides a desktop client but no mobile client. It is often the case that specific client applications and devices are designed to only work with their corresponding servers. In an ideal world, clients would all implement to standards and this would not happen, but in current practice, the vast majority of presence and IM endpoints work only (or only work well) with the server from the same vendor. A domain might want each user to have both a mobile client and a desktop client, which will require servers from each vendor, leading to intra-domain bridging.
Similarly, presence can contain rich information, including activities of the user (such as whether they are in a meeting or on the phone), their geographic location, and their mood. This presence state can be determined manually (where the user enters and updates the information), or automatically. Automatic determination of these states is far preferable, since it puts less burden on the user. Determination of these presence states is done by taking "raw" data about the user, and using it to generate corresponding presence states. This raw data can come from any source that has information about the user, including their calendaring server, their VoIP infrastructure, their VPN server, their laptop operating system, and so on. Each of these components is typically made by different vendors, each of which is likely to integrate that data with their presence servers. Consequently, presence servers from different vendors are likely to specialize in particular pieces of presence data, based on the other infrastructure they provide. The overall network will need to contain servers from those vendors, composing together the various sources of information, in order to combine their benefits. This use case is specific to presence, and results in intra-domain bridging.

---

4. Considerations for Bridging Models

TOC

When considering architectures for intra-domain presence and IM bridging, several issues need to be considered. The first two of these apply to both IM and presence (and indeed to any intra-domain communications, including voice). The latter two are specific to presence and IM respectively:

Routing: How are subscriptions and IMs routed to the right presence and IM server(s)? This issue is more complex in intra-domain models, since the right hand side of the @-sign cannot be used to perform this routing.

Policy and Identity: Where do user policies reside, and what presence and IM server(s) are responsible for executing that policy? What identities does the user have in each system and how do they relate?

Presence Data Ownership: Which presence servers are responsible for which pieces of presence information, and how are those pieces composed to form a coherent and consistent view of user presence?

Conversation Consistency: When considering instant messaging, if IM can be delivered to multiple servers, how do we make sure that the overall conversation is coherent to the user?

The sections below describe several different models for intra-domain bridging. Each model is driven by a set of use cases, which are described in an applicability subsection for each model. Each model description also discusses how routing, policy, presence data ownership and conversation consistency work.

5. Overview of the Models

TOC

There are three models for intra-domain bridging. These are partitioned, exclusive, and unioned. They can be explained relative to each other via a decision tree.

```
          +--------------+
          |              |
          |  Is a user   | one
          |  managed by  |-----> PARTITIONED
          |  one system  |
          |  or more than|
          |  one?        |
          +--------------+
                  |
                  | more
                  V

         +-----------------+
         |  Can the user   |
         |  be managed by  | no
         |  more than one  |---> EXCLUSIVE
         |  system at the  |
         |  same time?     |
         +-----------------+
                  |
                  |yes
                  |
                  V

               UNIONED
```

Figure 3: Decision Tree

The first question is whether any particular user is 'managed' by just one system, or more than one? Here, 'managed' means that the user is provisioned on the system, and can use it for some kind of presence and IM services. In the partitioned model, the answer is no - a user is on only one system. In that way, partitioned bridging is analagous to an inter-domain model where a user is handled by a single domain.
If a user is 'managed' by more than one system, is it more than one at the same time, or only one at a time? In the exclusive model, it is one at a time. The user can log into one system, log out, and then log into the other. For example, a user might have a PC client connected to system one, and a different PC client connected to system two. They can use one or the other, but not both. In unioned bridging, they can be connected to more than one at the same time. For example, a user might have a mobile client connected to one system, and a PC client connected to another.

6. Partitioned

In the partitioned model, a single domain has a multiplicity of servers, each of which manages a non-overlapping set of users. That is, for each user in the domain, their presence data, policy and IM handling reside on a single server. Each "single server" may in fact be a cluster.
Another important facet of the partitioned model is that, even though users are partitioned across different servers, they each share the same domain name in the right hand side of their URI, and this URI is what those users use when communicating with other users both inside and outside of the domain. There are many reasons why a domain would want all of its users to share the same right-hand side of the @-sign even though it is partitioned internally:

*The partitioning may reflect organizational or geographical structures that a domain admistrator does not want to reflect externally.

*If each partition had a separate domain name (i.e., engineering.example.com and sales.example.com), if a user changed organizations, this would necessitate a change in their URI.

*For reasons of vanity, users often like to have their URI (which appear on business cards, email, and so on), to be brief and short.

*If a watcher wants to add a presentity based on username and does not want to know, or does not know, which subdomain or internal department the presentity belongs to, a single domain is needed.

This model is illustrated in Figure 4 (Partitioned Model). As the model shows, the domain example.com has six users across three servers, each of which is handling two of the users.

```
.....................................................................
.                                                                   .
.                                                                   .
.                                                                   .
.   joe@example.com      alice@example.com     padma@example.com    .
.   bob@example.com      zeke@example.com      hannes@example.com   .
.    +-----------+        +-----------+       +-----------+         .
.    |           |        |           |       |           |         .
.    |   Server  |        |   Server  |       |   Server  |         .
.    |     1     |        |     2     |       |     3     |         .
.    |           |        |           |       |           |         .
.    +-----------+        +-----------+       +-----------+         .
.                                                                   .
.                                                                   .
.                                                                   .
.                           example.com                             .
.....................................................................
```

Figure 4: Partitioned Model

6.1. Applicability

TOC

The partitioned model arises naturally in larger domains, such as an enterprise or service provider, where issues of scale, organizational structure, or multi-vendor requirements cause the domain to be managed by a multiplicity of independent servers.
In cases where each user has an AoR (Address of Record) that directly points to its partition (for example, us.example.com), that model becomes identical to the inter-domain federated model and is not treated here further.

6.2. Routing

TOC

The partitioned intra-domain model works almost identically to an inter-domain federated model, with the primary difference being routing. In inter-domain federation, the domain part of the URI can be used to route presence subscriptions and IM messages from one domain to the other. This is no longer the case in an intra-domain model. Consider the case where Joe subscribes to his buddy list, which is

served by his presence server (server 1 in Figure 4 (Partitioned Model)). Alice is a member of Joe's buddy list. How does server 1 know that the back-end subscription to Alice needs to get routed to server 2?
There are several techniques that can be used to solve this problem, which are outlined in the subsections below.

6.2.1. Centralized Database

TOC

```
.....................................................................
.                         +-----------+                             .
.           alice?        |           |                             .
.       +---------------> |  Database |                             .
.       |   server 2      |           |                             .
.       |   +-------------|           |                             .
.       |   |             +-----------+                             .
.       |   |                                                       .
.       |   |                                                       .
.       |   |                                                       .
.       |   |                                                       .
.       |   |                                                       .
.       |   |                                                       .
.       |   V                                                       .
.   joe@example.com      alice@example.com     padma@example.com      .
.   bob@example.com      zeke@example.com      hannes@example.com   .
.    +-----------+        +-----------+       +-----------+          .
.    |           |        |           |       |           |          .
.    |   Server  |        |   Server  |       |   Server  |          .
.    |     1     |        |     2     |       |     3     |          .
.    |           |        |           |       |           |          .
.    +-----------+        +-----------+       +-----------+          .
.                                                                   .
.                                                                   .
.                                                                   .
.                           example.com                             .
.....................................................................
```

Figure 5: Centralized DB

One solution is to rely on a common, centralized database that maintains mappings of users to specific servers, shown in Figure 5 (Centralized DB). When Joe subscribes to his buddy list that contains Alice, server 1 would query this database, asking it which server is responsible for alice@example.com. The database would indicate server 2, and then server 1 would generate the backend SUBSCRIBE request towards server 2. Similarly, when Joe sends an INVITE to establish an IM session with Padma, he would send the IM to his IM server, an it would query the database to find out that Padma is supported on server 3. This is a common technique in large email systems. It is often implemented using internal sub-domains; so that the database would return alice@central.example.com to the query, and server 1 would modify the Request-URI in the request to reflect this.
Routing database solutions have the problem that they require standardization on a common schema and database protocol in order to work in multi-vendor environments. For example, LDAP and SQL are both possibilities. There is variety in LDAP schema; one possibility is H.350.4, which could be adapted for usage here [RFC3944] (Johnson, T., Okubo, S., and S. Campos, “H.350 Directory Services,” December 2004.).

6.2.2. Routing Proxy

```
.....................................................................
.                         +-----------+                             .
.        SUB/INV alice    |           |                             .
.       +---------------> |  Routing  |                             .
.       |                 |   Proxy   |                             .
.       |                 |           |                             .
.       |                 +-----------+                             .
.       |                       |                                   .
.       |                       |                                   .
.       |                       |                                   .
.       |                       |SUB/INV alice                      .
.       |                       |                                   .
.       |                       |                                   .
.       |                       V                                   .
.   joe@example.com      alice@example.com     padma@example.com    .
.   bob@example.com      zeke@example.com      hannes@example.com   .
.    +-----------+        +-----------+       +-----------+         .
.    |           |        |           |       |           |         .
.    |   Server  |        |   Server  |       |   Server  |         .
.    |     1     |        |     2     |       |     3     |         .
.    |           |        |           |       |           |         .
.    +-----------+        +-----------+       +-----------+         .
.                                                                   .
.                                                                   .
.                                                                   .
.                           example.com                             .
.....................................................................
```

Figure 6: Routing Proxy

A similar solution is to rely on a routing proxy or B2BUA. Instead of a centralized database, there would be a centralized SIP proxy farm. Server 1 would send requests (SUBSCRIBE, INVITE, etc.) for users it doesn't serve to this server farm, and the servers would lookup the user in a database (which is now accessed only by the routing proxy), and the resulting requests are sent to the correct server. A redirect server can be used as well, in which case the flow is very much like that of a centralized database, but uses SIP.

Routing proxies have the benefit that they do not require a common database schema and protocol, but they do require a centralized server function that sees all subscriptions and IM requests, which can be a scale challenge. For IM, a centralized proxy is very challenging when using pager mode, since each and every IM is processed by the central proxy. For session mode, the scale is better, since the proxy handles only the initial INVITE.

6.2.3. Subdomaining

TOC

In this solution, each user is associated with a subdomain, and is provisioned as part of their respective server using that subdomain. Consequently, each server thinks it is its own, separate domain. However, when a user adds a presentity to their buddy list without the subdomain, they first consult a shared database which returns the subdomained URI to subscribe or IM to. This sub-domained URI can be returned because the user provided a search criteria, such as "Find Alice Chang", or provided the non-subdomained URI (alice@example.com). This is shown in Figure 7 (Subdomaining)

```
.....................................................................
.                         +-----------+                             .
.    who is Alice?        |           |                             .
. +---------------------->|  Database |                             .
. |  alice@b.example.com  |           |                             .
. | +---------------------|           |                             .
. | |                     +-----------+                             .
. | |                                                               .
. | |                                                               .
. | |                                                               .
. | |                                                               .
. | |                                                               .
. | |                                                               .
. | |                                                               .
. | | joe@example.com      alice@example.com     padma@example.com  .
. | | bob@example.com      zeke@example.com      hannes@example.com .
. | |  +-----------+        +-----------+       +-----------+       .
. | |  |           |        |           |       |           |       .
. | |  |   Server  |        |   Server  |       |   Server  |       .
. | |  |     1     |        |     2     |       |     3     |       .
. | |  |           |        |           |       |           |       .
. | |  +-----------+        +-----------+       +-----------+       .
. | |                            ^                                  .
. | |                            |                                  .
. | |                            |                                  .
. | |                            |                                  .
. | |                            |                                  .
. | |                            |                                  .
. | |                     +-----------+                             .
. | +-------------------->|           |                             .
. |                       |  Client   |                             .
. |                       |           |                             .
. +-----------------------|           |                             .
.                         +-----------+                             .
.                                                                   .
.                                                                   .
.                                                                   .
.                           example.com                             .
.....................................................................
```

Figure 7: Subdomaining

Subdomaining puts the burden of routing within the client. The servers can be completely unaware that they are actually part of the same domain, and integrate with each other exactly as they would in an inter-domain model. However, the client is given the burden of determining the subdomained URI from the original URI or buddy name, and then subscribing or IMing directly to that server, or including the subdomained URI in their buddylist. The client is also responsible for hiding the subdomain structure from the user and storing the mapping information locally for extended periods of time. In cases where users have buddy list subscriptions, the client will need to resolve the buddy name into the sub-domained version before adding to their buddy list.
Subdmaining can be done via different databases. In order to provide consistent interface to clients, a front-end of a SIP redirect proxies can be implemented. A client would send the SIP request to one of the redirect proxies and the redirect proxy will reply with the right domain after consulting the database in whatever protocol the databases exposes.

6.2.4. Peer-to-Peer

TOC

Another model is to utilize a peer-to-peer network amongst all of the servers, and store URI to server mappings in the distributed hash table it creates. This has some nice properties but does require a standardized and common peer-to-peer protocol across vendors.

6.2.5. Forking

TOC

Yet another solution is to utilize forking. Each server is provisioned with the domain names or IP addresses of the other servers, but not with the mapping of users to each of those servers. When a server needs to handle a request for a user it doesn't have, it forks the request to all of the other servers. This request will be rejected with a 404 on the servers which do not handle that user, and accepted on the one that does. The approach assumes that servers can differentiate inbound requests from end users (which need to get passed on to other servers - for example via a back-end subscription) and from other servers (which do not get passed on). This approach works very well in organizations with a relatively small number of servers (say, two or three), and becomes increasingly ineffective with more and more servers. Indeed, if multiple servers exist for the purposes of achieving scale, this approach can defeat the very reason those additional servers were deployed.

6.2.6. Provisioned Routing

Yet another solution is to provision each server with each user, but for servers that don't actually serve the user, the provisioning merely tells the server where to proxy the request. This solution has extremely poor operational properties, requiring multiple points of provisioning across disparate systems.

6.3. Policy

A fundamental characteristic of the partitioned model is that there is a single point of policy enforcement (authorization rules and composition policy) for each user.
For more discussion regarding policy see Section 9 (More about Policy).

6.4. Presence Data

Another fundamental characteristic of the partitioned model is that the presence data for a user is managed authoritatively on a single server. In the example of Figure 4 (Partitioned Model), the presence data for Alice lives on server 2 alone (recall that server two may be physically implemented as a multiplicity of boxes from a single vendor, each of which might have a portion of the presence data, but externally it appears to behave as if it were a single server). A subscription from Bob to Alice may cause a transfer of presence information from server 2 to server 1, but server 2 remains authoritative and is the single root source of all data for Alice.

6.5. Conversation Consistency

Since the IM for a particular user are always delivered through a particular server that handles the user, it is relatively easy to achieve conversation consistency. That server receives all of the messages and readily pass them onto the user for rendering. Furthermore, a coherent view of message history can be assembled by the server, since it sees all messages. If a user has multiple devices, there are challenges in constructing a consistent view of the conversation with page mode IM. However, those issues exist in general with page mode and are not worsened by intra-domain bridging.

7. Exclusive

TOC

In the former (static) partitioned model, the mapping of a user to a specific server is done by some off-line configuration means. The configuration assigns a user to a specific server and in order to use a different server, the user needs to change (or request the administrator to do so) the configuration.
In some environments, this restriction of a user to use a particular server may be a limitation. Instead, it is desirable to allow users to freely move back and forth between systems, though using only a single one at a time. This is called Exclusive Bridging.
Some use cases where this can happen are:

- *The organization is using multiple systems where each system has its own characteristics. For example one server is tailored to work with some CAD (Computer Aided Design) system and provide presence and IM functionality along with the CAD system. The other server is the default presence and IM server of the organization. Users wish to be able to work with either system when they wish to, they also wish to be able to see the presence and IM with their buddies no matter which system their buddies are currently using.

- *An enterprise wishes to test presence servers from two different vendors. In order to do so they wish to install a server from each vendor and see which of the servers is better. In the static partitioned model, a user will have to be statically assigned to a particular server and could not compare the features of the two servers. In the dynamic partitioned model, a user may choose on whim which of the servers that are being tested to use. They can move back and forth in case of problems.

- *An enterprise is currently using servers from one vendor, but has decided to add a second. They would like to gradually migrate users from one to the other. In order to make a smooth transition, users can move back and forth over a period of a few weeks until they are finally required to stop going back, and get deleted from their old system.

- *A domain is using multiple clusters from the same vendor. To simplify administration, users can connect to any of the clusters, perhaps one local to their site. To accomplish this, the clusters are connected using exclusive bridging.

TOC

7.1. Routing

Due to its nature, routing in the Exclusive bridging model is more complex than the routing in the partitioned model.
Association of a user to a server can not be known until the user publishes a presence document to a specific server or registers to that server. Therefore, when Alice subscribes to Bob's presence information, or sends him an IM, Alice's server will not easily know the server that has Bob's presence and is handling his IM.
In addition, a server may get a subscription to a user, or an IM targeted at a user, but the user may not be connected to any server yet. In the case of presence, once the user appears in one of the servers, the subscription should be sent to that server.
A user may use two servers at the same time and have hers/his presence information on two servers. This should be regarded as a conflict and one of the presence clients should be terminated or redirected to the other server.
Fortunately, most of the routing approaches described for partitioned bridging, excepting provisioned routing, can be adapted for exclusive bridging.

7.1.1. Centralized Database

TOC

A centralized database can be used, but will need to support a test-and-set functionality. With it, servers can check if a user is already in a specific server and set the user to the server if the user is not on another server. If the user is already on another server, a redirect (or some other error message) will be sent to that user.
When a client sends a subscription request for some target user, and the target user is not associated with a server yet, the subscription must be 'held' on the server of the watcher. Once the target user connects and becomes bound to a server, the database needs to send a change notification to the watching server, so that the 'held' subscription can be extended to the server which is now handling presence for the user.
Note that this approach actually moves the scaling problem of the routing mechanism to the database, especially when the percentage of the community that is offline is large.

7.1.2. Routing Proxy

TOC

The routing proxy mechanism can be used for exclusive bridging as well. However, it requires signaling from each server to the routing proxy to indicate that the user is now located on that server. This can be done

by having each server send a REGISTER request to the routing proxy, for that user, and setting the contact to itself. The routing proxy would have a rule which allows only a single registered contact per user. Using the registration event package [RFC3680] (Rosenberg, J., “A Session Initiation Protocol (SIP) Event Package for Registrations,” March 2004.), each server subscribes to the registration state at the routing proxy for each user it is managing. If the routing proxy sees a duplicate registration, it allows it, and then uses a reg-event notification to the other server to de-register the user. Once the user is de-registered from that server, it would terminate any subscriptions in place for that user, causing the watching server to reconnect the subscription to the new server. Something similar can be done for in-progress IM sessions; however this may have the effect of causing a disruption in ongoing sessions.
Note that this approach actually moves the scaling problem of the routing mechanism to the registrar, especially when the percentage of the community that is offline is large.

7.1.3. Subdomaining

TOC

Subdomaining is just a variation on the centralized database. Assuming the database supports a test-and-set mechanism, it can be used for exclusive bridging.
However, the principle challenge in applying subdomaining to exclusive bridging is database change notifications. When a user moves from one server to another, that change needs to be propagated to all clients which have ongoing sessions (presence and IM) with that user. This requires a large-scale change notification mechanism - to each client in the network.

7.1.4. Peer-to-Peer

TOC

Peer-to-peer routing can be used for routing in exclusive bridging. Essentially, it provides a distributed registrar function that maps each AoR to the particular server that they are currently registered against. When a UA registers to a particular server, that registration is written into the P2P (Peer to Peer) network, such that queries for that user are directed to that presence server.
However, change notifications can be troublesome. When a user registered on server 1 now registers on server 2, server 2 needs to query the P2P network, discover that server 1 is handling the user, and then tell server 1 that the user has moved. Server 1 then needs to terminate its ongoing subscriptions and send the to server 2.

Furthermore, P2P networks do not inherently provide a test-and-set primitive, and consequently, it is possible for race conditions to occur where there is an inconsistent view on where the user is currently registered.

7.1.5. Forking

The forking model can be applied to exclusive bridging. When a user registers with a server or publishes a presence document to a server, and that server is not serving the user yet, that server begins serving the user. Furthermore, it needs to propagate a change notification to all of the other servers. This can be done using a registration event package; basically each server would subscribe to every other server for reg-event notifications for users they serve.
When subscription or IM request is received at a server, and that server doesn't serve the target user, it forks the subscription or IM to all other servers. If the user is currently registered somewhere, one will accept, and the others will reject with a 404. If the user is registered nowhere, all others generate a 404. If the request is a subscription, the server that received it would 'hold' the subscription, and then subscribe for the reg-event package on every other server for the target user. Once the target user registers somewhere, the server holding the subscription gets a notification and can propagate it to the new target server.
Like the P2P solution, the forking solution lacks an effective test-and-set mechanism, and it is therefore possible that there could be inconsistent views on which server is handling a user. One possible scenario where multiple servers will think that thy are serving the user would be when a subscription request is forked and reaches to multiple servers, each of them thinks that it serves the user.

7.2. Policy

Unless policy is somehow managed in the same database and is accessed by the servers in the exclusive bridging model, policy becomes more complicated in the exclusive bridging model. In the partitioned model, a user had their presence and IM managed by the same server all of the time. Thus, their policy can be provisioned and excecuted there. With exclusive bridging, a user can freely move back and forth between servers. Consequently, the policy for a particular user may need to execute on multiple different servers over time.
The simplest solution is just to require the user to separately provision and manage policies on each server. In many of the use cases above, exclusive bridging is a transient situation that eventually

settles into partitioned bridging. Thus, it may not be unreasonable to require the user to manage both policies during the transition. It is also possible that each server provides different capabilities, and thus a user will receive different service depending on which server they are connected to. Again, this may be an acceptable limitation for the use cases it supports.
For more discussion regarding policy see Section 8.1.2 (Policy and Identity) and Section 9 (More about Policy).

7.3. Presence Data

TOC

As with the partitioned model, in the exclusive model, the presence data for a user resides on a single server at any given time. This server owns all composition policies and procedures for collecting and distributing presence data.

7.4. Conversation Consistency

TOC

Because a user receives all of their IM on a single server at a time, there aren't issues with seeing a coherent conversation for the duration that a user is associated with that server.
However, if a user has sessions in progress while they move from one server to another, it is possible that IM's can be misrouted or dropped, or delivered out of order. Fortunately, this is a transient event, and given that its unlikely that a user would actually have in-progress IM sessions when they change servers, this may be an acceptable limitation.
However, conversation history may be more troubling. IM message history is often stored both in clients (for context of past conversations, search, etc.) and in servers (for the same reasons, in addition to legal requirements for data retention). If a user changes servers, some of their past conversations will be stored on one server, and some on another. Any kind of search or query facility provided amongst the server-stored messages would need to search amongst all of the servers to find the data.

TOC

8. Unioned

In the unioned model, each user is actually served by more than one presence server at a time. In this case, "served" implies two properties:

* A user is served by a server when that user is provisioned on that server, and

* That server is authoritative for some piece of presence state associated with that user or responsible for some piece of registration state associated with that user, for the purposes of IM delivery

In essence, in the unioned model, a user's presence and registration data is distributed across many presence servers, while in the partitioned and exclusive models, its centralized in a single server. Furthermore, it is possible that the user is provisioned with different identifiers on each server.
This definition speaks specifically to ownership of dynamic data - presence and registration state - as the key property. This rules out several cases which involve a mix of servers within the enterprise, but do not constitute intra-domain unioned bridging:

* A user utilizes an outbound SIP proxy from one vendor, which connects to a presence server from another vendor. Even though this will result in presence subscriptions, notifications, and IM requests flowing between servers, and the user is potentially provisioned on both, there is no authoritative presence or registration state in the outbound proxy, and so this is not intra-domain bridging.

* A user utilizes a Resource List Server (RLS) from one vendor, which holds their buddy list, and accesses presence data from a presence server from another vendor. This case is actually the partitioned case, not the unioned case. Effectively, the buddy list itself is another "user", and it exists entirely on one server (the RLS), while the actual users on the buddy list exist entirely within another. Consequently, this case does not have the property that a single presence resource exists on multiple servers at the same time.

* A user subscribes to the presence of a presentity. This subscription is first passed to their presence server, which acts as a proxy, and instead sends the subscription to the UA of the user, which acts as a presence edge server. In this model, it may appear as if there are two presence servers for the user (the actual server and their UA). However, the server is acting as a proxy in this case - there is only one source of presence

      information. For IM, there is only one source of registration
      state - the server. Thus, this model is partitioned, but with
      different servers owning IM and presence.

The unioned models arise naturally when a user is using devices from different vendors, each of which has their own respective servers, or when a user is using different servers for different parts of their presence state. For example, Figure 8 (Unioned Case 1) shows the case where a single user has a mobile client connected to server one and a desktop client connected to server two.

```
               alice@example.com           alice@example.com
                +------------+              +------------+
                |            |              |            |
                |            |              |            |
                |   Server   |--------------|   Server   |
                |     1      |              |     2      |
                |            |              |            |
                |            |              |            |
                +------------+              +------------+
                   \                                /
                    \                              /
                     \                            /
                      \                           /
                       \                         /
                        \                       /
                         \...................../.......
                          \                    /      .
                          .\                  /       .
                          . \  |         +--------+   .
                          .    |         |+-------+|  .
                          .   +---+      ||      ||   .
                          .   |+-+|      ||      ||   .
                          .   |+-+|      |+-------+|  .
                          .   |   |      +--------+   .
                          .   |   |      /------ /    .
                          .   +---+     /------ /     .
                          .            --------/      .
                          .                           .
                          .............................

                                    Alice
```

Figure 8: Unioned Case 1

As another example, a user may have two devices from the same vendor, both of which are asociated with a single presence server, but that presence server has incomplete presence state about the user. Another presence server in the enterprise, due to its access to state for that user, has additional data which needs to be accessed by the first presence server in order to provide a comprehensive view of presence data. This is shown in Figure 9 (Unioned Case 2). This use case tends to be specific to presence.

```
  alice@example.com           alice@example.com
   +------------+              +------------+
   |            |              |            |
   |  Presence  |              |  Presence  |
   |   Server   |--------------|   Server   |
   |     1      |              |     2      |
   |            |              |            |
   |            |              |            |
   +------------+              +------------+
         ^                      |         |
         |                      |         |
         |                      |         |
   ///-------\\\                |         |
||| specialized |||             |         |
 || state       ||              |         |
   \\\-------///                |         |
                        .............................
                        .       |         |         .
                        .     | |       +---------+ .
                        .     |         |+-------+| .
                        .    +---+      ||       || .
                        .    |+-+|      ||       || .
                        .    |+-+|      |+-------+| .
                        .    |   |      +---------+ .
                        .    |   |      /------ /   .
                        .    +---+     /------ /    .
                        .             --------/     .
                        .                           .
                        .                           .
                        .............................
                                   Alice
```

Figure 9: Unioned Case 2

Another use case for unioned bridging are subscriber moves. Consider a domain which uses multiple servers, typically running in a partitioned configuration. The servers are organized regionally so that each user is served by a server handling their region. A user is moving from one region to a new job in another, while retaining their SIP URI. In order to provide a smooth transition, ideally the system would provide a "make before break" functionality, allowing the user to be added onto the new server prior to being removed from the old. During the transition period, especially if the user had multiple clients to be moved, they can end up with state existing on both servers at the same time.

8.1. Hierarchical Model

TOC

The unioned intra-bridging model can be realized in one of two ways - using a hierarchical structure or a peer structure.
In the hierarchical model, presence subscriptions and IM requests for the target are always routed first to one of the servers - the root. In the case of presence, the root has the final say on the structure of the presence document delivered to watchers. It collects presence data from its child presence servers (through notifications or publishes received from them) and composes them into the final presence document. In the case of IM, the root applies IM policy and then passes the IM onto the children for delivery. There can be multiple layers in the hierarchical model. This is shown in Figure 10 (Hierarchical Model) for presence.

```
                                +-----------+
              *-----------*     |           |
              |Auth and   |---->|  Presence | <--- root
              |Composition|     |   Server  |
              *-----------*     |           |
                                |           |
                                +-----------+
                                   /       ---
                                  /           ----
                                 /                ----
                                /                     ----
                               V                          -V
                     +-----------+                      +-----------+
                     |           |                      |           |
     *-----------*   |  Presence |      *-----------*   |  Presence |
     |Auth and   |-->|   Server  |      |Auth and   |-->|   Server  |
     |Composition|   |           |      |Composition|   |           |
     *-----------*   |           |      *-----------*   |           |
                     +-----------+                      +-----------+
                        |    ---
                        |       -----
                        |            -----
                        |                 -----
                        |                      -----
                        |                           -----
                        V                                --V
                     +-----------+                      +-----------+
                     |           |                      |           |
     *-----------*   |  Presence |      *-----------*   |  Presence |
     |Auth and   |-->|   Server  |      |Auth and   |-->|   Server  |
     |Composition|   |           |      |Composition|   |           |
     *-----------*   |           |      *-----------*   |           |
                     +-----------+                      +-----------+
```

Figure 10: Hierarchical Model

It is important to note that this hierarchy defines the sequence of presence composition and policy application, and does not imply a literal message flow. As an example, consider once more the use case of Figure 8 (Unioned Case 1). Assume that presence server 1 is the root, and presence server 2 is its child. When Bob's PC subscribes to Bob's buddy list (on presence server 2), that subscription will first go to presence server 2. However, that presence server knows that it is not the root in the hierarchy, and despite the fact that it has presence state for Alice (who is on Bob's buddy list), it creates a back-end subscription to presence server 1. Presence server 1, as the root, subscribes to Alice's state at presence server 2. Now, since this subscription came from presence server 1 and not Bob directly, presence server 2 provides the presence state. This is received at presence server 1, which composes the data with its own state for Alice, and then provides the results back to presence server 2, which, having acted as an RLS, forwards the results back to Bob. Consequently, this flow, as a message sequence diagram, involves notifications passing from presence server 2, to server 1, back to server 2. However, in terms of composition and policy, it was done first at the child node (presence server 2), and then those results used at the parent node (presence server 1).

Note that we are assuming that presence servers will subscribe to each other. It is also possible to assume that given the hierarchy configuration knowledge, a presence server can send PUBLISH messages to other presence servers based on the configured hierarchy. However, sending PUBLISH messages from one presence server to another presence server will actually make the presence server that sends the PUBLISH messages a presence source and not a presence server from the point of view of the receiving presence server. Therefore, we will assume that presence servers will subscribe to each other but PUBLISH messages instead of subscriptions could be used if they are preferred.

8.1.1. Routing

TOC

In the hierarchical model, the servers need to collectively be provisioned with the topology of the network. This topology defines the root and the parent/child relationships. These relationships could in fact be different on a user-by-user basis; however, this is complex to manage. In all likelihood, the parent and child relationships are identical for each user. The overall routing algorithm can be described thusly:

- If a SUBCRIBE is received from the parent node for this presentity, perform subscriptions to each child node for this presentity, and then take the results, apply composition and authorization policies, and propagate to the parent. If a node is the root, the logic here applies regardless of where the request came from.

- If an IM request is received from the parent node for a user, perform IM processing and then proxy the request to each child IM server for this user. If a node is the root, the logic here applies regardless of where the request came from.

- If a request is received from a node that is not the parent node for this presentity, proxy the request to the parent node. This includes cases where the node that sent the request is a child node. Note that if the node that receives the request can send the request directly to the root, it should do so thus reducing the traffic in the system.

This routing rule is relatively simple, and in a two-server system is almost trivial to provision. Interestingly, it works in cases where some users are partitioned and some are unioned. When the users are partitioned, this routing algorithm devolves into the forking algorithm of Section 6.2.5 (Forking). This points to the forking algorithm as a good choice since it can be used for both partitioned and unioned.
An important property of the routing in the hierarchical model is that the sequence of composition and policy operations for any IM or presence session is identical, regardless of the watcher or sender of the IM. The result is that the overall presence state provided to a watcher, and overall IM behavior, is always consistent and independent of the server the client is connected to. We call this property the *consistency property*, and it is an important metric in assessing the correctness of a bridged presence and IM system.

8.1.2. Policy and Identity

TOC

Policy and identity are a clear challenge in the unioned model.
Firstly, since a user is provisioned on many servers, it is possible that the identifier they utilize could be different on each server. For example, on server 1, they could be joe@example.com, whereas on server 2, they are joe.smith@example.com. In cases where the identifiers are not equivalent, a mapping function needs to be provisioned. This ideally happens on root server.
Secondly, the unioned model will result in back-end subscriptions extending from one presence server to another presence server. These subscriptions, though made by the presence server, need to be made on-behalf-of the user that originally requested the presence state of the presentity. Since the presence server extending the back-end subscription will not often have credentials to claim identity of the watcher, asserted identity using techniques like P-Asserted-ID [RFC3325] (Jennings, C., Peterson, J., and M. Watson, “Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks,” November 2002.) or authenticated identity [RFC4474] (Peterson, J. and C. Jennings, “Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP),” August 2006.) are required, along with the associated trust relationships between servers. Optimizations, such as view sharing [I-D.ietf-simple-view-sharing] (Rosenberg, J., Donovan, S., and K.

McMurry, “Optimizing Federated Presence with View Sharing,” November 2008.) can help improve performance. The same considerations apply for IM.

The principle challenge in a unioned model is policy, including both authorization and composition policies. There are three potential solutions to the administration of policy in the hierarchical model (only two of which apply in the peer model, as we'll discuss below). These are root-only, distributed provisioned, and central provisioned.

8.1.2.1. Root Only

TOC

In the root-only policy model, authorization policy, IM policy, and composition policy are applied only at the root of the tree. This is shown in Figure 11 (Root Only).

```
                                +-----------+
              *-----------*     |           |
              |           |---->|           | <--- root
              |  Policy   |     |   Server  |
              *-----------*     |           |
                                |           |
                                +-----------+
                                   /       ---
                                  /           ----
                                 /                ----
                                /                     ----
                               V                          -V
                     +-----------+                      +-----------+
                     |           |                      |           |
                     |           |                      |           |
                     |   Server  |                      |   Server  |
                     |           |                      |           |
                     |           |                      |           |
                     +-----------+                      +-----------+
                        |    ---
                        |       -----
                        |            -----
                        |                 -----
                        |                      -----
                        |                           -----
                        V                                --V
                     +-----------+                      +-----------+
                     |           |                      |           |
                     |           |                      |           |
                     |   Server  |                      |   Server  |
                     |           |                      |           |
                     |           |                      |           |
                     +-----------+                      +-----------+
```

Figure 11: Root Only

As long as a subscription request came from its parent, every child presence server would automatically accept the subscription, and provide notifications containing the full presence state it is aware of. Similarly, any IM received from a parent would be simply propagated onwards towards children. Any composition performed by a child presence server would need to be lossless, in that it fully combines the source data without loss of information, and also be done without any per-user

provisioning or configuration, operating in a default or administrator-provisioned mode of operation.
The root-only model has the benefit that it requires the user to provision policy in a single place (the root). However, it has the drawback that the composition and policy processing may be performed very poorly. Presumably, there are multiple presence servers in the first place because each of them has a particular speciality. That speciality may be lost in the root-only model. For example, if a child server provides geolocation information, the root presence server may not have sufficient authorization policy capabilities to allow the user to manage how that geolocation information is provided to watchers.

8.1.2.2. Distributed Provisioning

TOC

The distributed provisioned model looks exactly like the diagram of Figure 10 (Hierarchical Model). Each server is separately provisioned with its own policies, including what users are allowed to watch, what presence data they will get, how it will be composed, what IMs get blocked, and so on.
One immediate concern is whether the overall policy processing, when performed independently at each server, is consistent, sane, and provides reasonable degrees of privacy. It turns out that it can, if some guidelines are followed.
For presence, consider basic "yes/no" authorization policies. Lets say a presentity, Alice, provides an authorization policy in server 1 where Bob can see her presence, but on server 2, provides a policy where Bob cannot. If presence server 1 is the root, the subscription is accepted there, but the back-end subscription to presence server 2 would be rejected. As long as presence server 1 then rejects the subscription, the system provides the correct behavior. This can be turned into a more general rule:

- To guarantee privacy safety, if the back-end subscription generated by a presence server is denied, that server must deny the triggering subscription in turn, regardless of its own authorization policies. This means that a presence server cannot send notifications on its own until it has confirmed subscriptions from downstream servers.

For IM, basic yes/no authorization policies work in a similar way. If any one of the servers has a policy that says to block an IM, the IM is not propagated further down the chain. Whether the overall system blocks IMs from a sender depends on the topology. If there is no forking in the hierarchy, the system has the property that, if a sender is blocked at any server, the user is blocked overall. However, in tree

structures where there are multiple children, it is possible that an IM could be delivered to some downstream clients, and not others.
Things get more complicated when one considers presence authorization policies whose job is to block access to specific pieces of information, as opposed to blocking a user completely. For example, lets say Alice wants to allow Bob to see her presence, but not her geolocation information. She provisions a rule on server 1 that blocks geolocation information, but grants it on server 2. The correct mode of operation in this case is that the overall system will block geolocation from Bob. But will it? In fact, it will, if a few additional guidelines are followed:

- If a presence server adds any information to a presence document beyond the information received from its children, it must provide authorization policies that govern the access to that information.

- If a presence server does not understand a piece of presence data provided by its child, it should not attempt to apply its own authorization policies to access of that information.

- A presence server should not add information to a presence document that overlaps with information that can be added by its parent. Of course, it is very hard for a presence server to know whether this information overlaps. Consequently, provisioned composition rules will be required to realize this.

If these rules are followed, the overall system provides privacy safety and the overall policy applied is reasonable. This is because these rules effectively segment the application of policy based on specific data, to the servers that own the corresponding data. For example, consider once more the geolocation use case described above, and assume server 2 is the root. If server 1 has access to, and provides geolocation information in presence documents it produces, then server 1 would be the only one to provide authorization policies governing geolocation. Server 2 would receive presence documents from server 1 containing (or not) geolocation, but since it doesn't provide or control geolocation, it lets that information pass through. Thus, the overall presence document provided to the watcher will contain gelocation if Alice wanted it to, and not otherwise, and the controls for access to geolocation would exist only on server 1.
For more discussion regarding policy see Section 9 (More about Policy).

8.1.2.3. Central Provisioning

TOC

The central provisioning model is a hybrid between root-only and distributed provisioning. Each server does in fact execute its own authorization and composition policies. However, rather than the user provisioning them independently in each place, there is some kind of central portal where the user provisions the rules, and that portal generates policies for each specific server based on the data that the corresponding server provides. This is shown in Figure 12 (Central Provisioning).

```
                            +---------------------+
                            |provisioning portal  |...........
                            +---------------------+          .
                              .  .    .     .   .            .
                              .  .    .     .   .            .
                              .  .    .     .   .......................
    ...........................  .    .     .                .        .
    .                            .    .     .                .        .
    .                            .    .     .                .        .
    .  ...........................    .     .............    .        .
    .  .                              .                 .    .        .
    .  .         ......................                 .    .        .
    .  .         V            +-----------+             .    .        .
    .  .    *-----------*     |           |             .    .        .
    .  .    |Auth and   |---->|  Presence | <--- root   .    .        .
    .  .    |Composition|     |   Server  |             .    .        .
    .  .    *-----------*     |           |             .    .        .
    .  .                      |           |             .    .        .
    .  .                      +-----------+             .    .        .
    .  .                        |      ----             .    .        .
    .  .                        |          -------      .    .        .
    .  .                        |                 -------    .        .
    .  .                        |                       .-------      .
    .  .                        V                       .    .  ---V  V
    .  .                    +-----------+               .    . +---------+
    .  .                    |           |               V    . |         |
    .  .    *-----------*   |  Presence |      *-----------* . |Presence |
    .  ....>|Auth and   |-->|   Server  |      |Auth and   |-->| Server  |
    .       |Composition|   |           |      |Composition| . |         |
    .       *-----------*   |           |      *-----------* . +---------+
    .                       +-----------+                    .
    .                         /        \                     .
    .                        /          \                    .
    .                       /            \                   .
    .                      /              \                  .
    .                     /                \                 .
    .                    /                  \                .
    .                   V                   V                ...
    .              +-----------+        +-----------+          .
    V              |           |        |           |          V
   *-----------*   |  Presence |        |  Presence |     *-----------*
   |Auth and   |-->|   Server  |        |   Server  |<----|Auth and   |
   |Composition|   |           |        |           |     |Composition|
   *-----------*   |           |        |           |     *-----------*
                   +-----------+        +-----------+
```

Figure 12: Central Provisioning

Centralized provisioning brings the benefits of root-only (single point of user provisioning) with those of distributed provisioning (utilize full capabilities of all servers). Its principle drawback is that it requires another component - the portal - which can represent the union of the authorization policies supported by each server, and then delegate those policies to each corresponding server.
The other drawback of centralized provisioning is that it assumes completely consistent policy decision making on each server. There is a rich set of possible policy decisions that can be taken by servers, and this is often an area of differentiation.

8.1.2.4. Centralized PDP

TOC

The centralized provisioning model assumes that there is a single point of policy administration, but that there is independent decision making at each presence and IM server. This only works in cases where the decision function - the policy decision point - is identical in each server.
An alternative model is to utilize a single point of policy administration and a single point of policy decision making. Each presence server acts solely as an enforcement point, asking the policy server (through a policy protocol of some sort) how to handle the presence or IM. The policy server then comes back with a policy decision - whether to proceed with the subscription or IM, and how to filter and process it. This is shown in Figure 13 (Central PDP).

```
 +------------+      +---------------+
 |Provisioning|=====>|Policy Decision|
 |   Portal   |      |   Point (PDP) |
 +------------+      +---------------+
                      # #   #  #   #
    ################### #   #  #   ###########################
    #                   #   #  #                             #
    #            ########   #  ####################          #
    #            #    +-----------+               #          #
    #            #    |           |               #          #
    #            #    |           | .... root     #          #
    #            #    |   Server  |               #          #
    #            #    |           |               #          #
    #            #    |           |               #          #
    #            #    +-----------+               #          #
    #            #       /       ---              #          #
    #            #      /          ----           #          #
    #            #     /               ----       #          #
    #            #    /                    ----  #          #
    #            #   V                         -V#          #
    #      +-----------+                      +-----------+  #
    #      |           |                      |           |  #
    #      |           |                      |           |  #
    #      |   Server  |                      |   Server  |  #
    #      |           |                      |           |  #
    #      |           |                      |           |  #
    #      +-----------+                      +-----------+  #
    #         |    ---                                       #
    #         |       -----                                  #
    #         |            -----                             #
    #         |                 -----                        #
    #         |                      -----                   #
    #         |                           -----              #
    #         V                                --V           #
    #      +-----------+                      +-----------+  #
    #      |           |                      |           |  #
    #######|           |                      |           |  #
           |   Server  |                      |   Server  |###
           |           |                      |           |
           |           |                      |           |
           +-----------+                      +-----------+

===== Provisioning Protocol
##### Policy Protocol
----- SIP
```

Figure 13: Central PDP

The centralized PDP has the benefits of central provisioning, and consistent policy operation, and decouples policy decision making from presence and IM processing. This decoupling allows for multiple presence and IM servers, but still allows for a single policy function overall. The individual presence and IM servers don't need to know about the policies themselves, or even know when they change. Of course, if a server is caching the results of a policy decision, change notifications are required from the PDP to the server, informing it of the change (alternatively, traditional TTL-based expirations can be used if delay in updates are acceptable).
It is also possible to move the decision making process into each server. In that case, there is still a centralized policy portal and centralized repository of the policy data. The interface between the servers and the repository then becomes some kind of standardized database interface.
For the centralized and distributed provisioning approaches, and the centralized decision approach, the hierarchical model suffers overall from the fact that the root of the policy processing may not be tuned to the specific policy needs of the device that has subscribed. For example, in the use case of Figure 8 (Unioned Case 1), presence server 1 may be providing composition policies tuned to the fact that the device is wireless with limited display. Consequently, when Bob subscribes from his mobile device, when presence server 2 is the root, presence server 2 may add additional data and provide an overall presence document to the client which is not optimized for that device. This problem is one of the principal motivations for the peer model, described below.
For more discussion regarding policy see Section 9 (More about Policy).

8.1.3. Presence Data

The hierarchical model is based on the idea that each presence server in the chain contributes some unique piece of presence information, composing it with what it receives from its child, and passing it on. For the overall presence document to be reasonable, several guidelines need to be followed:

- A presence server must be prepared to receive documents from its peer containing information that it does not understand, and to apply unioned composition policies that retain this information, adding to it the unique information it wishes to contribute.

- A user interface rendering some presence document provided by its presence server must be prepared for any kind of presence document compliant to the presence data model, and must not assume a specific structure based on the limitations and implementation choices of the server to which it is paired.

If these basic rules are followed, the overall system provides functionality equivalent to the combination of the presence capabilities of the servers contained within it, which is highly desirable.

8.1.4. Conversation Consistency

TOC

Unioned bridging introduces a particular challenge for conversation consistency. A user with multiple devices attached to multiple servers could potentially try to participate in the conversation on multiple devices at once. This would clearly pose a challenge. There are really two approaches that produce a sensible user experience.
The first approach simulates the "phone experience" with IM. When a user (say Alice) sends an IM to Bob, and Bob is a unioned user with two devices on two servers, Bob receives that IM on both devices. However, when he "answers" by typing a reply from one of those devices, the conversation continues only on that device. The other device on the other server receives no further IMs for this session - either from Alice or from Bob. Indeed, the IM window on Bob's unanswered device may even disappear to emphasize this fact.
This mode of operation, which we'll call uni-device IM, is only feasible with session mode IM, and its realization using traditional SIP signaling is described in [RFC4975] (Campbell, B., Mahy, R., and C. Jennings, “The Message Session Relay Protocol (MSRP),” September 2007.).
The second mode of operation, called multi-device IM, is more of a conferencing experience. The initial IM from Alice is delivered to both Bob's devices. When Bob answers on one, that response is shown to Alice but is also rendered on Bob's other device. Effectively, we have set up an IM conference where each of Bob's devices is an independent participant in the conference. This model is feasible with both session and pager mode IM; however conferencing works much better overall with session mode.
A related challenge is conversation history. In the uni-device IM mode, this past history for a user's conversation may be distributed amongst the different servers, depending on which clients and servers were involved in the conversation. As with the exclusive model, IM search and retrieval services may need to access all of the servers on which a user might be located. This is easier for the unioned case than the exclusive one, since in the unioned case, the user's location is on a fixed number of servers based on provisioning. This problem is even

more complicated in IM page mode when multiple devices are present, due to the limitation of page mode in these configurations.

8.2. Peer Model

TOC

In the peer model, there is no one root. When a watcher subscribes to a presentity, that subscription is processed first by the server to which the watcher is connected (effectively acting as the root), and then the subscription is passed to other child presence servers. The same goes for IM; when a client sends an IM, the IM is processed first by the server associated with the sender (effectively acting as the root), and then the IM is passed to the child IM servers. In essence, in the peer model, there is a per-client hierarchy, with the root being a function of the client. Consider the use case in Figure 8 (Unioned Case 1) If Bob has his buddy list on presence server 1, and it contains Alice, presence server 1 acts as the root, and then performs a back-end subscription to presence server 2. However, if Joe has his buddy list on presence server 2, and his buddy list contains Alice, presence server 2 acts as the root, and performs a back-end subscription to presence server 1. Similarly, if Bob sends an IM to Alice, it is processed first by server 1 and then server 2. If Joe sends an IM to Alice, it is first processed by server 2 and then server 1. This is shown in Figure 14 (Peer Model).

```
        alice@example.com           alice@example.com
         +------------+              +------------+
         |            |<-------------|            |<--------+
         |            |              |            |         |
 Connect |   Server   |              |   Server   |         |
  Alice  |     1      |              |     2      | Connect |
   +---->|            |------------->|            |  Alice  |
   |     |            |              |            |         |
   |     +------------+              +------------+         |
   |        \                                /              |
   |         \                              /               |
   |          \                            /                |
   |           \                           /                |
   |            \                         /                 |
   |             \                       /                  |
...|........      \...................../.......   .........|........
.          .       \                    /      .   .                .
.          .       .\                  /       .   .    +--------+  .
.   |      .       . \  |         +--------+   .   .    |+------+|  .
.   |      .       .    |         |+------+|   .   .    ||      ||  .
.  +---+   .       .   +---+      ||      ||   .   .    ||      ||  .
.  |+-+|   .       .   |+-+|      ||      ||   .   .    |+------+|  .
.  |+-+|   .       .   |+-+|      |+------+|   .   .    +--------+  .
.  |   |   .       .   |   |      +--------+   .   .    /------ /   .
.  |   |   .       .   |   |      /------ /    .   .   /------ /    .
.  +---+   .       .   +---+     /------ /     .   .  --------/     .
.          .       .            --------/      .   .                .
.          .       .                           .   .                .
............       .............................   ..................

   Bob                     Alice                       Joe
```

Figure 14: Peer Model

Whereas the hierarchical model clearly provides the consistency property, it is not obvious whether a particular deployment of the peer model provides the consistency property. When policy decision making is distributed amongst the servers, it ends up being a function of the composition policies of the individual servers. If Pi() represents the composition and authorization policies of server i, and takes as input one or more presence documents provided by its children, and outputs a presence document, the overall system provides consistency when:

```
                  Pi(Pj()) = Pj(Pi())
```

which is effectively the commutativity property.

8.2.1. Routing

TOC

Routing in the peer model works similarly to the hierarchical model. Each server would be configured with the children it has when it acts as the root. The overall presence routing algorithm then works as follows:

* If a presence server receives a subscription for a presentity from a particular watcher, and it already has a different subscription (as identified by dialog identifiers) for that presentity from that watcher, it rejects the second subscription with an indication of a loop. This algorithm does rule out the possibility of two instances of the same watcher subscribing to the same presentity.

* If a presence server receives a subscription for a presentity from a watcher and it doesn't have one yet for that pair, it processes it and generates back end subscriptions to each configured child. If a back-end subscription generates an error due to loop, it proceeds without that back-end input.

The algorithm for IM routing works almost identically.
For example, consider Bob subscribing to Alice. Bob's client is supported by server 1. Server 1 has not seen this subscription before, so it acts as the root and passes it to server 2. Server 2 hasn't seen it before, so it accepts it (now acting as the child), and sends the subscription to its child, which is server 1. Server 1 has already seen the subscription, so it rejects it. Now server 2 basically knows it is the child, and so it generates documents with just its own data.
As in the hierarchical case, it is possible to intermix partitioned and peer models for different users. In the partitioned case, the routing for hierarchical devolves into the forking routing described in Section 6.2.5 (Forking). However, intermixing peer and exclusive bridging for different users is challenging and is out of scope.

TOC

8.2.2. Policy

The policy considerations for the peer model are very similar to those of the hierarchical model. However, the root-only policy approach is non-sensical in the peer model, and cannot be utilized. The distributed and centralized provisioning approaches apply, and the rules described above for generating correct results provide correct results in the peer model as well.
However, the centralized PDP model works particularly well in concert with the peer model. It allows for consistent policy processing regardless of the type of rules, and has the benefit of having a single point of provisioning. At the same time, it avoids the need for defining and having a single root; indeed there is little benefit for utilizing the hierarchical model when a centralized PDP is used.
However, the distributed processing model in the peer model eliminates the problem described in Section 8.1.2.3 (Central Provisioning). The problem is that composition and authorization policies may be tuned to the needs of the specific device that is connected. In the hierarchical model, the wrong server for a particular device may be at the root, and the resulting presence document poorly suited to the consuming device. This problem is alleviated in the peer model. The server that is paired or tuned for that particular user or device is always at the root of the tree, and its composition policies have the final say in how presence data is presented to the watcher on that device.
For more discussion regarding policy see Section 9 (More about Policy).

8.2.3. Presence Data

TOC

The considerations for presence data and composition in the hierarchical model apply in the peer model as well. The principle issue is consistency, and whether the overall presence document for a watcher is the same regardless of which server the watcher connects from. As mentioned above, consistency is a property of commutativity of composition, which may or may not be true depending on the implementation.
Interestingly, in the use case of Figure 9 (Unioned Case 2), a particular user only ever has devices on a single server, and thus the peer and hierarchical models end up being the same, and consistency is provided.

TOC

8.2.4. Conversation Consistency

The hierarchical and peer models have no impact on the issue of conversation consistency; the problem exists identically for both approaches.

---

9. More about Policy

TOC

There are several models that are described in this document that may create some ambiguity regarding what the subscribing user will see when the policy is not managed in a centralized way (Section 8.1.2.3 (Central Provisioning), Section 8.1.2.4 (Centralized PDP)).

- Exclusive model - In this case one server may have a certain policy and the other server may have a different policy. Bob subscribing one day to server 1 will not be able to see Alice's presence while he will be able to see her presence when he subscribes to server 2.

- Hierarchical unioned model - Since only the root will provide the presence information all the users will see the same presence information. However, if there are some contradicting rules in the servers, what the subscriber will see will in most cases be the strictest and most minimal view and will be dependent on the hierarchy of the servers in the hierarchical model.

- Peer unioned model - In this case one peer server may have a certain policy and the other server may have a different policy. Bob subscribing one day to peer server 1 will not be able to see Alice's presence while he will be able to see her presence when he subscribes to peer server 2.

There are several reasons why distributed policy model may needed:

- Adapting policy to the presence server type - The particular presence server may be adapted to specific type of presence information and devices. It will be hard or not feasible to provide a centralized policy for all the types of presence servers/devices

- A presence server that is part of the intradomain bridging may not be able to use the centralized policy provisioning since it does not support this feature.

It is probable that although confusing for users, distributed provisioning will be used at least in the initial deployments of the intradomain bridging until standards for central policy provisioning will be developed and implemented by various presence servers.

10. Acknowledgements

TOC

The authors would like to thank Paul Fullarton, David Williams, Sanjay Sinha, and Paul Kyzivat for their comments. Thanks to Adam Roach, Ben Campbell Michael Froman, Alan Johanston and Christer Holmberg for their dedicated review.

11. Security Considerations

TOC

While the normal presence and IM protocol (as SIP) mechanisms for securing presecne and IM should be used, The principle issue in intra-domain bridging is that of privacy. It is important that the system meets user expectations, and even in cases of user provisioning errors or inconsistencies, it provides appropriate levels of privacy. This is an issue in the unioned models, where user privacy policies can exist on multiple servers at the same time. The guidelines described here for authorization policies help ensure that privacy properties are maintained.

12. IANA Considerations

TOC

There are no IANA considerations associated with this specification.

13. Informative References

TOC

[RFC2778]	Day, M., Rosenberg, J., and H. Sugano, “A Model for Presence and Instant Messaging,” RFC 2778, February 2000 (TXT).
[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 (TXT).
[RFC3265]	Roach, A., “Session Initiation Protocol (SIP)-Specific Event Notification,” RFC 3265, June 2002 (TXT).
[RFC3325]	

	Jennings, C., Peterson, J., and M. Watson, “Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks,” RFC 3325, November 2002 (TXT).
[RFC3428]	Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, “Session Initiation Protocol (SIP) Extension for Instant Messaging,” RFC 3428, December 2002 (TXT).
[RFC3680]	Rosenberg, J., “A Session Initiation Protocol (SIP) Event Package for Registrations,” RFC 3680, March 2004 (TXT).
[RFC3856]	Rosenberg, J., “A Presence Event Package for the Session Initiation Protocol (SIP),” RFC 3856, August 2004 (TXT).
[RFC3863]	Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, “Presence Information Data Format (PIDF),” RFC 3863, August 2004 (TXT).
[RFC3903]	Niemi, A., “Session Initiation Protocol (SIP) Extension for Event State Publication,” RFC 3903, October 2004 (TXT).
[RFC3944]	Johnson, T., Okubo, S., and S. Campos, “H.350 Directory Services,” RFC 3944, December 2004 (TXT).
[RFC4474]	Peterson, J. and C. Jennings, “Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP),” RFC 4474, August 2006 (TXT).
[RFC4479]	Rosenberg, J., “A Data Model for Presence,” RFC 4479, July 2006 (TXT).
[RFC4662]	Roach, A., Campbell, B., and J. Rosenberg, “A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists,” RFC 4662, August 2006 (TXT).
[RFC4975]	Campbell, B., Mahy, R., and C. Jennings, “The Message Session Relay Protocol (MSRP),” RFC 4975, September 2007 (TXT).
[RFC5344]	Houri, A., Aoki, E., and S. Parameswar, “Presence and Instant Messaging Peering Use Cases,” RFC 5344, October 2008 (TXT).
[I-D.ietf-simple-view-sharing]	Rosenberg, J., Donovan, S., and K. McMurry, “Optimizing Federated Presence with View Sharing,” draft-ietf-simple-view-sharing-02 (work in progress), November 2008 (TXT).

Authors' Addresses

TOC

	Jonathan Rosenberg

	jdrosen.net
	Monmouth, NJ
	US
Email:	jdrosen@jdrosen.net
URI:	http://www.jdrosen.net
	Avshalom Houri
	IBM
	Science Park, Rehovot
	Israel
Email:	avshalom@il.ibm.com
	Colm Smyth
	Avaya
	Dublin 18, Sandyford Business Park
	Ireland
Email:	smythc@avaya.com
	Francois Audet
	Skype
Email:	francois.audet@skype.net