XEP-0281: DMUC1: Distributed Multi-User Chat

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

2010-07-20
Version 0.2

Status	Type	Short Name
Retracted	Standards Track	NOT-YET-ASSIGNED

This document defines methods for distributing Multi-User Chat (MUC) rooms across multiple chat services.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

1.1 Motivation

Multi-User Chat (XEP-0045) [1] defines a full-featured technology for multi-user text conferencing in XMPP. By design, XEP-0045 assumes that a conference room is hosted at a single service, which can be accessed from any point on the network. However, this assumption introduces a single point of failure for the conference room, since if occupants at a using domain lose connectivity to the hosting domain then they also lose connectivity to the room. In some deployment scenarios (and even on the open Internet) this behavior is suboptimal. Therefore, this document attempts to define a technology for distributing MUC rooms across multiple services.

1.2 Requirements

This specification addresses the following requirements:

1. Enable distribution of a MUC room across multiple services.
2. Enable a service to determine which other services it will peer with.
3. Enable the room creator to specify if distribution is allowed.
4. Enable occupants to remain in an instance of the conference if connectivity is lost to other instances.
5. Enable syncing of history, configuration, and room rosters on reconnect.

1.3 Approach

The basic approach to distribution of MUC rooms is as follows:

1. A user creates a room on a service and configures it as ”distributed” (or the service assumes that the room is distributed based on local service policy); this first instance of the room is called a SOURCE and the service on which it is created is called a FIRSTHOST.
2. The firsthost can immediately request that other services (called PEERHOSTS) replicate the room by creating their own local instances (called SHADOWS); alternatively, the firsthost can wait to send the replication request until users from the peerhost have joined the room.
3. If a user from the peerhost attempts to join the source room after replication is established, the firsthost invites the user to join the shadow rather than the source by sending a direct invitation to the user.

[1] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

4. As long as the peerhost and firsthost have connectivity, they share room messages, room rosters, and room configuration changes in real time. If any conflict arises, the firsthost's information rules since it is "first among equals".

5. If the peerhost loses connectivity to the firsthost, it maintains the shadow, including local room history, room roster, and room configuration, and if possible also maintains connectivity with other peerhosts.

6. Upon reconnecting to the firsthost, a peerhost exchanges room history and room rosters with the firsthost and receives room configuration data (if modified).

The room IDs of source rooms SHOULD be opaque to users and unique across all possible peerhosts, for example by generating a UUID in accordance with RFC 4122 [2] or by hashing the human-readable name of the room using the SHA-256 algorithm in accordance with SHA [3].

2 Terminology

2.1 General Terms

This document adds the following terms to those defined in XEP-0045:

- Firsthost -- The MUC service at which a room is created.
- Peerhost -- Any MUC service (other than the firsthost) that hosts an instance of the room.
- Source -- The canonical instance of the room at the firsthost.
- Shadow -- An instance of the room at a peerhost.

2.2 Entities

In this document, the examples use the following entities.

- Firsthost: chat.shakespeare.lit
- Peerhosts: rooms.macbeth.lit and conference.marlowe.lit
- Source: coven@chat.shakespeare.lit
- Shadows: theroom@rooms.macbeth.lit and theroom@conference.marlowe.lit

[2] RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace <http://tools.ietf.org/html/rfc4122>.
[3] Secure Hash Standard: Federal Information Processing Standards Publication 180-2 <http://csrc.nist.gov/publications/fips/fips180-2/fips186-2withchangenotice.pdf>.

3 Use Cases

3.1 Creating a Room

When the original room owner creates the room (or subsequently configures the room), the service MAY simply default all rooms to "distributed".
Alternatively, the server MAY offer the option of making the room a "distributed room". This is done by including the "muc#roomconfig_distributed" feature in the room configuration form:

Listing 1: Service Sends Configuration Form

```
<iq from='coven@chat.shakespeare.lit'
    id='create1'
    to='crone1@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden' var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#roomconfig</value>
      </field>
      ...
      <field
          label='Distribute␣Room␣Across␣Multiple␣Services?'
          type='boolean'
          var='muc#roomconfig_distributed'>
        <value>0</value>
      </field>
      ...
    </x>
  </query>
</iq>
```

The room owner specifies a value of "1" or "true" [4] if room distribution is desired:

Listing 2: Room Owner Submits Configuration Form

```
<iq from='coven@chat.shakespeare.lit'
    id='configure1'
    to='crone1@shakespeare.lit/desktop'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#roomconfig</value>
      </field>
```

[4] In accordance with Section 3.2.2.1 of XML Schema Part 2: Datatypes, the allowable lexical representations for the xs:boolean datatype are the strings "0" and "false" for the concept 'false' and the strings "1" and "true" for the concept 'true'; implementations MUST support both styles of lexical representation.

```
      ...
      <field var='muc#roomconfig_distributed'>
        <value>true</value>
      </field>
      ...
    </x>
  </query>
</iq>
```

Alternatively, the firsthost can choose to perform room distribution in the background, rather than exposing the 'muc#roomconfig_distributed' option to the user.

3.2 Replicating a Room

When a firsthost would like a peerhost to provide a shadow, it sends a replication request to the peerhost.

Listing 3: Firsthost Requests Replication of Room

```
<iq from='chat.shakespeare.lit'
    id='replicate1'
    to='rooms.macbeth.lit'
    type='set'>
  <replicate xmlns='urn:xmpp:dmuc:0'>
    <room id='theroom'/>
  </replicate>
</iq>
```

If the peerhost is willing and able to replicate the room, it returns an IQ-result:

Listing 4: Peerhost Acknowledges Replication Request

```
<iq to='rooms.macbeth.lit'
    id='replicate1'
    from='chat.shakespeare.lit'
    type='result'/>
```

Several error cases are possible (the peerhost is resource constrained, the firsthost is forbidden to peer with the peerhost, etc.); these will be specified more fully in a future version of this specification.
Once the peerhost acknowledges that it is willing and able to replicate the room, four things happen:

1. The source sends the room configuration to the shadow.
2. The source sends the room roster to the shadow.

3. The shadow optinoally requests the room history from the source.

4. The firsthost informs other peerhosts about the new peerhost.

Listing 5: Source Sends Room Configuration to Shadow

```
<iq from='coven@chat.shakespeare.lit'
    id='config1'
    to='theroom@rooms.macbeth.lit'
    type='set'>
  <config xmlns='urn:xmpp:dmuc:0'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden' var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#roomconfig</value>
      </field>
      ...
    </x>
  </config>
</iq>
```

Listing 6: Shadow Acknowledges Receipt of Room Configuration

```
<iq to='theroom@rooms.macbeth.lit'
    id='config1'
    from='coven@chat.shakespeare.lit'
    type='result'/>
```

Listing 7: Source Sends Room Roster to Shadow

```
<iq from='coven@chat.shakespeare.lit'
    id='roster1'
    to='theroom@rooms.macbeth.lit'
    type='set'>
  <roster xmlns='urn:xmpp:dmuc:0'>
    <item affiliation='owner' jid='crone1@shakespeare.lit/desktop'
       name='foo'/>
    <item affiliation='admin' jid='wiccarocks@shakespeare.lit/laptop'
       name='bar'/>
    <item affiliation='none' jid='hag66@shakespeare.lit/pda' name='baz
       '/>
  </roster>
</iq>
```

Listing 8: Shadow Acknowledges Receipt of Room Roster

```
<iq to='theroom@rooms.macbeth.lit'
    id='roster1'
    from='coven@chat.shakespeare.lit'
    type='result'/>
```

The new shadow SHOULD request the room history. This is done by sending an IQ-get from the shadow to the source, containing a <history/> element qualified by the 'http://jabber.org/protocol/muc' namespace (the syntax and semantics of this element are described in XEP-0045).

Listing 9: Peerhost Requests Room History

```
<iq to='theroom@rooms.macbeth.lit'
    id='history1'
    from='coven@chat.shakespeare.lit'
    type='get'>
  <room xmlns='urn:xmpp:dmuc:0'
        id='theroom'>
    <history xmlns='http://jabber.org/protocol/muc'/>
  </room>
</iq>
```

The history request MAY include any of the attributes specified in XEP-0045:: 'maxchars', 'maxstanzas', 'seconds', and 'since'.

Listing 10: Firsthost Sends Room History to Peerhost

```
<iq from='coven@chat.shakespeare.lit'
    id='history1'
    to='theroom@rooms.macbeth.lit'
    type='result'/>
  <history xmlns='http://jabber.org/protocol/muc'>
    <message
        xmlns='jabber:client'
        from='coven@chat.shakespeare.lit/foo'
        type='groupchat'>
      <body>Message 1.</body>
      <delay xmlns='urn:xmpp:delay'
         from='crone1@shakespeare.lit/desktop'
         stamp='2002-10-13T23:58:37Z'/>
    </message>

    <message
        xmlns='jabber:client'
        from='coven@chat.shakespeare.lit/bar'
        type='groupchat'>
      <body>Message 2.</body>
      <delay xmlns='urn:xmpp:delay'
         from='wiccarocks@shakespeare.lit/laptop'
         stamp='2002-10-13T23:58:43Z'/>
    </message>

    <message
        xmlns='jabber:client'
```

```
        from='coven@chat.shakespeare.lit/baz'
        type='groupchat'>
      <body>Message 3.</body>
      <delay xmlns='urn:xmpp:delay'
         from='hag66@shakespeare.lit/pda'
         stamp='2002-10-13T23:58:49Z'/>
    </message>
  </history>
</iq>
```

Listing 11: Peerhost Acknowledges Receipt of Room History

```
<iq to='theroom@rooms.macbeth.lit'
    id='history1'
    from='coven@chat.shakespeare.lit'
    type='result'/>
```

The firsthost also informs other connected peerhosts about the new peerhost.

Listing 12: Firsthost Reports New Peerhost

```
<iq from='chat.shakespeare.lit'
    id='ph1'
    to='conference.marlowe.lit'
    type='set'>
  <peerhost xmlns='urn:xmpp:dmuc:0'>
    <room id='theroom'/>
  </peerhost>
</iq>
```

Listing 13: Peerhost Acknowledges Notification of New Peerhost

```
<iq from='conference.marlowe.lit'
    id='ph1'
    to='chat.shakespeare.lit'
    type='result'/>
```

3.3 Finding a Room

XEP-0045 specifies that rooms can be found on a chat service using Service Discovery (XEP-0030) [5].
A user at a peerhost could send a service discovery items (”disco#items”) request to the firsthost in order to find the source room. However, this introduces a dependency on communication with the firsthost, and we have stipulated that such communication might not be available.
Alternatively, the peerhost could keep a list of the shadow rooms that it hosts. From the

[5] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

perspective of the user at the peerhost, these rooms are local in the sense that the user can join the shadow and thereby interact with the occupants of the shadow (and, if server-to-server communication is working, with the occupants of the source). This approach is RECOMMENDED.
Note: If the peerhost frequently loses communications with the firsthost and the list of rooms located at the firsthost is large, the peerhost will want to use a more efficient method of synchronizing its room list than sending disco#items requests and receiving large disco#items results. However, methods for optimizing this synchronization process are out of scope for this specification.
To find rooms on the peerhost, the local user sends a "disco#items" request to the peerhost.

Listing 14: User Queries Peerhost for Rooms

```
<iq from='somewitch@macbeth.lit/mobile'
    id='nb2ga51g'
    to='rooms.macbeth.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The service returns a list of the public rooms it hosts, which includes any shadow rooms.

Listing 15: Peerhost Returns Disco Items

```
<iq from='rooms.macbeth.lit'
    id='nb2ga51g'
    to='somewitch@macbeth.lit/mobile'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='heath@rooms.macbeth.lit' name='A Lonely Heath'/>
    <item jid='coven@rooms.macbeth.lit' name='A Dark Cave'/>
    <item jid='forres@rooms.macbeth.lit' name='The Palace'/>
    <item jid='inverness@rooms.macbeth.lit' name='Macbeth&apos;s
        Castle'/>
  </query>
</iq>
```

The user can then send a disco#info request to each room. If the room is a shadow, the service MUST include extended information about the room using the Service Discovery Extensions (XEP-0128) [6] format, specifically with a "http://jabber.org/protocol/muc#roominfo" FORM_TYPE and a (newly-defined) "dmuc-source" field.

Listing 16: Room Returns Extended Disco Info Results

```
<iq from='coven@rooms.macbeth.lit'
```

[6] XEP-0128: Service Discovery Extensions <https://xmpp.org/extensions/xep-0128.html>.

```
    id='hf71vs8m'
    to='somewitch@macbeth.lit/mobile'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='conference' name='A␣Dark␣Cave' type='text'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/muc#roominfo</value>
      </field>
      <field var='dmuc-source' label='Source␣Room' type='jid-single'>
        <value>coven@chat.shakespeare.lit</value>
      </field>
    </x>
  </query>
</iq>
```

This informs the user that the source room is "coven@chat.shakespeare.lit"; as a result, the user's client can trap any outbound requests destined for the source room (service discovery, room join, etc.) and redirect them to the shadow.

3.4 Joining a Shadow

The process of joining a shadow is exactly as described in XEP-0045.

Listing 17: Local User Seeks to Join Shadow Room

```
<presence
    from='somewitch@macbeth.lit/mobile'
    to='coven@rooms.macbeth.lit/anotherwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

The shadow then informs the source (and any other shadows) of the user's presence; it does so by sending presence from the roomjid of the user at the shadow to a roomjid with the same roomnick at the source and shadow(s).

Listing 18: Shadow Informs Source of New Occupant

```
<presence
    from='theroom@rooms.macbeth.lit/anotherwitch'
    to='coven@chat.shakespeare.lit/anotherwitch'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='participant'/>
  </x>
</presence>
```

The source then delivers that presence stanza to its local users. (Note: The shadow needs to send only one presence stanza to the source, thus reducing the number of stanzas sent over the server-to-server link between the peerhost and the firsthost.)

3.5 Joining a Room

If there is no local shadow available at the peerhost, or if the peerhost does not support extended service discovery information as described above, then the local user at the peerhost will end up sending a join request to the source room instead of the shadow room.
When this happens, the firsthost determines if it will invite the user to join a shadow at a peerhost instead. The process for determining when to send invitations is implementation specific and might be subject to configuration at the firsthost (e.g., the firsthost might send invitations only to users of a domain associated with the peerhost and only after a certain number of such users have joined the room at the firsthost).
To begin, a user at the peerhost attempts to join the source room at the firsthost:

Listing 19: User Seeks to Join Source Room

```
<presence
    from='somewitch@macbeth.lit/mobile'
    to='coven@chat.shakespeare.lit/firstwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

The source room returns a <redirect/> presence error to the user and invites the user to join a shadow room instead.

Listing 20: Source Informs User of a Shadow

```
<presence
    from='coven@chat.shakespeare.lit/baz'
    to='somewitch@macbeth.lit/mobile'
    type='error'>
  <error type='cancel'>
    <redirect xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>rooms.
        macbeth.lit</redirect>
  </error>
</presence>
```

Then the source invites the user to the shadow using the protocol defined in Direct MUC Invitations (XEP-0249) [7].

Listing 21: Source Invites User to a Shadow

[7] XEP-0249: Direct MUC Invitations <https://xmpp.org/extensions/xep-0249.html>.

```
<message
    from='coven@chat.shakespeare.lit'
    to='somewitch@macbeth.lit/mobile'
  <x xmlns='jabber:x:conference'
     jid='theroom@rooms.macbeth.lit'/>
</message>
```

The user then joins the shadow.

Listing 22: User Joins Shadow

```
<presence
    from='somewitch@macbeth.lit/mobile'
    to='theroom@rooms.macbeth.lit/firstwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

The shadow then informs the source (and any other shadows) of the user's presence; it does so by sending presence from the roomjid of the user at the shadow to a roomjid with the same roomnick at the source and shadow(s).

Listing 23: Shadow Informs Source of New Occupant

```
<presence
    from='theroom@rooms.macbeth.lit/firstwitch'
    to='coven@chat.shakespeare.lit/firstwitch'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='participant'/>
  </x>
</presence>
```

The source then delivers that presence stanza to its local users. (Note: The shadow needs to send only one presence stanza to the source, thus reducing the number of stanzas sent over the server-to-server link between the peerhost and the firsthost.)

3.6 Sending a Message

When a user sends a message to an instance, the instance sends the message to its local occupants and to other instances.

Listing 24: User Sends Message

```
<message
    from='somewitch@macbeth.lit/mobile'
    to='theroom@rooms.macbeth.lit'
    type='groupchat'>
  <body>Message 4.</body>
```

```
</message>
```

Listing 25: Instance Distributes Message

```
<message
    from='theroom@rooms.macbeth.lit/firstwitch'
    to='theroom@conference.marlowe.lit/firstwitch'
    type='groupchat'>
  <body>Message 4.</body>
</message>

<message
    from='theroom@rooms.macbeth.lit/firstwitch'
    to='coven@chat.shakespeare.lit/firstwitch'
    type='groupchat'>
  <body>Message 4.</body>
</message>
```

The source then delivers that message stanza to its local users. (Note: The shadow needs to send only one message stanza to the source, thus reducing the number of stanzas sent over the server-to-server link between the peerhost and the firsthost.)

3.7 sync

To follow.

4 Determining Support

If a MUC service supports distributed rooms, it MUST return a feature of "urn:xmpp:dmuc:0" in response to service discovery information requests.

5 Security Considerations

To follow.

6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [8].

[8] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see

7 XMPP Registrar Considerations

This document requires no interaction with the XMPP Registrar [9].

8 Acknowledgements

Thanks to Jay Carlson, Boyd Fletcher, and Michael Krutsch for their early conversations regarding distributed chatrooms.

9 XML Schema

To follow.

<http://www.iana.org/>.

[9] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.