XEP-0354: Customizable Message Routing

Florian Schmaus
mailto:flo@geekplace.eu
xmpp:flo@geekplace.eu

2014-10-15
Version 0.1

Status	Type	Short Name
Experimental	Standards Track	NOT_YET_ASSIGNED

This specification specifies customizable behavior of RFC 6121 section 8.5.2.1.1 to allow various message routing algorithms (e.g., for load balancing).

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction 1

2 Requirements 1

3 Glossary 1

4 Use Cases 2
- 4.1 Non-balancing Message Routing Algorithms . . . 2
- 4.2 Balancing Message Routing Algorithms . . . 2
- 4.3 Non-Use Cases . . . 3

5 Protocol 3
- 5.1 Discovering Support . . . 3
- 5.2 Quering the current active and available message routing algorithms . . . 4
- 5.3 Changing the active message routing algorithm . . . 4
- 5.4 Message Routing Hints . . . 5
  - 5.4.1 Determing support . . . 5
  - 5.4.2 Determing available message routing algorithms of a (remote) server . 6
  - 5.4.3 Sending a message with routing hint . . . 6

6 Business Rules 7
- 6.1 General Rules . . . 7
- 6.2 Routing Algorithm Specification . . . 7
  - 6.2.1 Deliver to all . . . 7
  - 6.2.2 Deliver to most active resource . . . 7
  - 6.2.3 Load balance using round robin . . . 7
  - 6.2.4 Weighted load balancing . . . 7

7 Implementation Notes 8

8 Security Considerations 8

9 IANA Considerations 8

10 XMPP Registrar Considerations 8
- 10.1 Protocol Namespaces . . . 8
- 10.2 Protocol Versioning . . . 8

11 XML Schema 9

1 Introduction

The ”Server Rules for Processing XML Stanzas” section in RFC 6121 [1] defines only a ”one receives” or ”all receive” algorithm for message routing if the receiving entity of a message stanza has multiple connected resources and the message is addressed to a bare JID. Also RFC 6121 does not define a method to check or change the currently used message routing algorithm. Furthermore, none of the described routing algorithms is a good match when it comes down to achieve load-balancing between the connected resources, which is sometimes desirable.
This extensions solves those issues by allowing clients to configure their desired message routing algorithm. By exploiting the freedom provided by RFC 6121 which allows servers to implement their own algorithm for determining the ”most available” resource.
PubSub Queueing (XEP-0254) [2] defines a XEP to queue PubSub items, which could be used as alternative approach, instead of the herein defined message routing algorithms (’urn:xmpp:cmr:roundrobin’ and ’urn:xmpp:cmr:weighted’). But this solution requires more and complexer code on the server and client side, while the guarantees provided by XEP-0254 are not always required.

2 Requirements

The authors have designed the customizable message routing protocol with the following requirements in mind:

- The protocol MUST NOT violate the server rules for processing XML Stanzas as defined in RFC 6121 § 8
- The protocol should be easy to use by clients and easy to adopt by server implementations
- The protocol must by extensible by further message routing algorithms

3 Glossary

The following terms are used throughout this document

Customizable Message Routing (CMR) The name of this XEP and the protocol defined by it.

[1] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[2] XEP-0254: PubSub Queueing <https://xmpp.org/extensions/xep-0254.html>.

Message Routing Algorithm An algorithm that defines how a server processes message stanzas send to a server-local bare JID of type 'message' or 'chat', with at least one available or connected resource of the receiver (RFC 6121 § 8.5.2.1.1).

4 Use Cases

4.1 Non-balancing Message Routing Algorithms

Non-balancing message algorithms consists of the traditional algorithms that are widely used by XMPP implementations as defined in RFC 6121 § 8.5.2.1.1 and RFC 3921 [3]. They usually try to send a message either to the all resources or try to determine the best resource based on some heuristic. They where designed with human-to-human interaction in mind.
Examples for non-balancing algorithms include:

- Deliver to all ('urn:xmpp:cmr:all')
- Deliver to most active resource ('urn:xmpp:cmr:mostactive')

4.2 Balancing Message Routing Algorithms

Balanced message routing algorithms try to distribute messages between the available resources of the receiver. They are ideal for use with the Internet of Things (IoT) and especially machine to machine (M2M) technologies.
Consider a large deployment of sensor nodes send their data to a cluster for further processing. Every cluster node establishes a connection using the same bare JID but a different resource for receiving the sensor data. Furthermore, after a cluster connection is authenticated, it queries the XMPP service for the CMR status, and enables a balancing message routing algorithm.
The sensor nodes can now send their data as payload of a message stanza to the cluster. The server will evenly distributed the data between the cluster nodes by using a round-robin scheme.
Since CMR balances message stanzas of type 'normal' or 'chat', PubSub notifications will also be evenly distributed between the connected resources of an connection where CMR is active.
Examples for balancing algorithms include:

- Load balance using round robin ('urn:xmpp:cmr:roundrobin')
- Weighted load balancing ('urn:xmpp:cmr:weighted')

[3] RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc3921>.

4.3 Non-Use Cases

CMR is not suitable for scenarios where the routing behavior should also consider resources with negative priority. This would violate RFC 6121 § 8.5.2.1.1. A suitable specification for such use cases may already exists. For example a proper solution that involves forking of messages even to resources with negatives priorities is explained in Section 6 of Message Carbons (XEP-0280) [4].

5 Protocol

5.1 Discovering Support

An entity advertises support for this protocol by including the 'urn:xmpp:cmr:0' feature in its service discovery information features as specified in Service Discovery (XEP-0030) [5] or section 6.3 of Entity Capabilities (XEP-0115) [6].

Listing 1: Service discovery information request

```
<iq xmlns='jabber:client'
    from='romeo@montague.example/garden'
    id='info1'
    to='montague.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 2: Service discovery information response

```
<iq xmlns='jabber:client'
    from='montague.example'
    id='info1'
    to='romeo@montague.example/garden'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:cmr:0'/>
    ...
  </query>
</iq>
```

[4] XEP-0280: Message Carbons <https://xmpp.org/extensions/xep-0280.html>.
[5] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[6] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

5.2 Quering the current active and available message routing algorithms

Clients are able to query the server for the currently active and available message routing algorithms.

Listing 3: Client requests the current CMR state

```
<iq xmlns='jabber:client'
    from='romeo@montague.example/garden'
    id='state1'
    type='get'>
  <query xmlns='urn:xmpp:cmr:0'/>
</iq>
```

The result IQ, must include exactly one <active/> element and at least one <available/> element. Both elements must posses a 'algorithm' attribute, which contains the name of the available or active algorithm.

Listing 4: Server returns information about the CMR state

```
<iq xmlns='jabber:client'
  from='romeo@montague.example'
  id='state1'
  to='romeo@montague.example/garden'
  type='result'>
<query xmlns='xmlns='urn:xmpp:cmr:0'>
␣␣<active␣algorithm='urn:xmpp:cmr:all'␣/>
␣␣<available␣algorithm='urn:xmpp:cmr:all'/>
␣␣<available␣algorithm='urn:xmpp:cmr:mostactive'/>
␣␣<available␣algorithm='urn:xmpp:cmr:roundrobin'/>
␣␣<available␣algorithm='urn:xmpp:cmr:weighted'/>
</iq>
```

5.3 Changing the active message routing algorithm

To change the active message routing algorithm, the client sends an <iq/> of type 'set' containing a 'cmr' child element which contains the desired algorithm as value of the 'algorithm' attribute.

Listing 5: Client requests to change the active message routing algorithm

```
<iq xmlns='jabber:client'
    from='romeo@montague.example/garden'
    id='change1'
    type='set'>
  <cmr xmlns='urn:xmpp:cmr:0'
       algorithm='urn:xmpp:cmr:roundrobin' />
</iq>
```

The server will respond with an result <iq/> if the routing algorithm was successfully changed.

Listing 6: Server acknowledges the change of the active message routing algorithm

```
<iq xmlns='jabber:client'
    from='romeo@montague.example'
    id='change1'
    to='romeo@montague.example/garden'
    type='result'/>
```

If the server is unable to change the message routing algorithm, then an error <iq/> is returned to the client.

Listing 7: Server informs client that the routing algorithm was not changed

```
<iq xmlns='jabber:client'
    from='romeo@montague.example'
    id='change1'
    to='romeo@montague.example/garden'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

5.4 Message Routing Hints

If allowed and supported by the server, clients are able to annotate message stanza with a routing hint, that SHOULD affect the used message routing algorithm for the annotated stanza.

5.4.1 Determing support

Listing 8: Client sends service discovery information request

```
<iq xmlns='jabber:client'
    from='romeo@montague.example/garden'
    id='info2'
    to='bar.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 9: Server replies with service discovery information response

```
<iq xmlns='jabber:client'
    from='bar.example'
```

```
    id='info2'
    to='romeo@montague.example/garden'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:cmr:hints:0'/>
    ...
  </query>
</iq>
```

5.4.2 Determing available message routing algorithms of a (remote) server

Listing 10: Client requests available message routing algorithms

```
<iq xmlns='jabber:client'
    from='romeo@montague.example/garden'
    to='bar.example'
    id='state1'
    type='get'>
  <query xmlns='urn:xmpp:cmr:0'/>
</iq>
```

Listing 11: Server replies with available message routing algorithms

```
<iq xmlns='jabber:client'
  from='bar.example'
  id='state1'
  to='romeo@montague.example/garden'
  type='result'>
<query xmlns='xmlns='urn:xmpp:cmr:0'>
␣␣<available␣algorithm='urn:xmpp:cmr:all'/>
␣␣<available␣algorithm='urn:xmpp:cmr:mostactive'/>
␣␣<available␣algorithm='urn:xmpp:cmr:roundrobin'/>
␣␣<available␣algorithm='urn:xmpp:cmr:weighted'/>
␣␣<available␣algorithm='urn:xmpp:cmr:forkalways'/>
</iq>
```

5.4.3 Sending a message with routing hint

Listing 12: Client sends a message with a routing hint

```
<message xmlns='jabber:client'
    from='romeo@montague.example/garden'
    to='foo@bar.example'>
  <body>Hello everyone</body>
  <cmr xmlns='urn:xmpp:cmr:0'
       algorithm='run:xmpp:cmr:forkalways'/>
</message>
```

6 Business Rules

6.1 General Rules

Clients are allowed to change the CMR state at any time. CMR MUST only affect the routing of message stanzas of type 'normal' or 'chat', which are addressed to a bare JID and where the receiving entity has more then one available resource with a non-negative presence priority. That is, it affects the routing decision as described in RFC 6121 § 8.5.2.1.1 for messages of type 'normal' and 'chat'.
The CMR state, ie. the used routing algorithm, is identical for every session of an XMPP client. Therefore implementations MUST NOT allow different CMR states between the resources of the same bare JID.
If an entity advertises support for CMR as described in 3. it MUST support at least one message routing algorithm.

6.2 Routing Algorithm Specification

6.2.1 Deliver to all

Algorithm Namespace: 'urn:xmpp:cmr:all'
Deliver to all non-negative resources with share the same maximum priority. And if message type is 'chat', only to those that have opted in to receive chat messages.

6.2.2 Deliver to most active resource

Algorithm Namespace: 'urn:xmpp:cmr:mostactive'
Deliver the message to the "most available" resource or resources, depending on the server's implementation.

6.2.3 Load balance using round robin

Algorithm Namespace: 'urn:xmpp:cmr:roundrobin'
Deliver the message to the next resource selected by a round-robin algorithm.

6.2.4 Weighted load balancing

Algorithm Namespace: 'urn:xmpp:cmr:weighted'
Deliver the message to a resource selected by a weighted round-robin algorithm. The weight of a resource is determined by its priority.

7 Implementation Notes

Servers implementing CMR MUST at least implement one message routing algorithm, and offer at least one of of 'urn:xmpp:cmr:all' and 'urn:xmpp:cmr:mostactive'. Technically this is a constraint derived from Section 8.5.2.1.1. of RFC 6121.

8 Security Considerations

This specification introduces no known security considerations.

9 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [7].

10 XMPP Registrar Considerations

10.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:cmr:0

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [8] shall add the foregoing namespace to the registry located at <https://xmpp.org/registrar/namespaces.html>, as described in Section 4 of XMPP Registrar Function (XEP-0053) [9].

10.2 Protocol Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of

[7] 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 <http://www.iana.org/>.

[8] 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/>.

[9] XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>.

XEP-0053.

11 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:cmr:0'
    xmlns='urn:xmpp:cmr:0'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-XXXX: http://www.xmpp.org/extensions/xep-xxxx.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='query'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='active'
                    minOccurs='1'
                    maxOccurs='1'/>
        <xs:element ref='available'
                    minOccurs='1'
                    maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='active' type='cmrtype'/>

  <xs:element name='available' type='cmrtype'/>

  <xs:element name='cmr' type='cmrtype'/>

  <xs:complexType name='cmrtype'>
    <xs:complexType>
      <xs:attribute name='algorithm' type='xs:string' use='required'/>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```