# XEP-0451: Stanza Multiplexing

Sam Whited
mailto:sam@samwhited.com
xmpp:sam@samwhited.com
https://blog.samwhited.com/

2021-01-19
Version 0.1.0

| Status | Type | Short Name |
|---|---|---|
| Experimental | Standards Track | mux |

This spec provides a mechanism for multiplexing multiple virtual hosts over a single XMPP session.

# Legal

## Copyright

## Permissions

## Warranty

## Liability

## Conformance

## Contents

# 1 Introduction

The ability to multiplex multiple virtual hosts over a single XMPP session (historically known as ”piggybacking”) was originally defined in RFC 3920 [1] and later pulled out into Server Dialback (XEP-0220) [2] for use with RFC 6120 [3]. With the advent of cheap or free TLS certificates the use of dialback began falling off on the public XMPP network as more secure authentication mechanisms such as SASL EXTERNAL began to become more common. However, multiplexing is still a useful technique in constrained environments regardless of the authentication mechanism being used.

Multiplexing is also useful for reusing connections for additional services associated with a domain but hosted at a subdomain. For example, both the ”montague.example” and the ”capulet.example” may be hosted by the same XMPP server which may also host Mediated Information eXchange (MIX) (XEP-0369) [4] services at ”chat.montague.example” and ”rooms.capulet.example” respectively. Without multiplexing this would require eight TCP connections for a bidirectional exchange of stanzas between two sending domains and two target domains. However, with multiplexing this can be reduced to two connections, or, at the operator’s discretion, more than two for operational reasons. If multiplexing is not used, the number of server-to-server connections needed to exchange stanzas between virtual hosting providers or multi-service XMPP servers can increase signficantly. This can lead to the number of connections exceeding the maximum number of connections allowed from a single address as explained in Best Practices to Discourage Denial of Service Attacks (XEP-0205) [5].

This specification defines new mechanisms for advertising and negotiating multiple hosts over a single session. Furthermore it advances the state of the art over the multiplexing solution defined in Server Dialback (XEP-0220) [6] by working on both client-to-server (c2s) and server-to-server (s2s) sessions.

# 2 Advertising Support

If a server supports receiving multiplexed streams it SHOULD inform the connecting entity when returning stream features during the negotiation process. Two mechanisms exist for authenticating domains that can be multiplexed over a connection: domains may be authenticated using the TLS certificate (and client certificate if applicable), and domains may be authorized using the connection authorization mechanism described later in this document.

To advertise support for multiplexing all domains present in a TLS certificate the server includes a <mux/> element qualified by the ’urn:xmpp:mux:0’ namespace in the stream

[1] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.

[2] XEP-0220: Server Dialback <https://xmpp.org/extensions/xep-0220.html>.

[3] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

[4] XEP-0369: Mediated Information eXchange (MIX) <https://xmpp.org/extensions/xep-0369.html>.

[5] XEP-0205: Best Practices to Discourage Denial of Service Attacks <https://xmpp.org/extensions/xep-0205.html>.

[6] XEP-0220: Server Dialback <https://xmpp.org/extensions/xep-0220.html>.

features list. This feature MUST be advertised only after TLS has been negotiated (either by opportunistic TLS using the STARTTLS feature or by implicit TLS when establishing the TCP socket) an before authentication using SASL EXTERNAL has been performed. This feature is not mandatory to negotiate.

Listing 1: Server advertises support for mux using SASL EXTERNAL

```
<stream:features>
  <mux xmlns='urn:xmpp:mux:0'/>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>EXTERNAL</mechanism>
  </mechanisms>
</stream:features>
```

The mux feature may also be advertised after authentication with SASL EXTERNAL. If advertised after authentication the feature MUST include a list of supported hosts wrapped in <host/> elements.

Listing 2: Server advertises support for connection authorization

```
<stream:features>
  <mux xmlns='urn:xmpp:mux:0'>
    <host>capulet.example</host>
    <host>montague.example</host>
    <host>chat.montague.example</host>
    <host>rooms.capulet.example</host>
  </mux>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
</stream:features>
```

## 3 Authentication with SASL EXTERNAL

If the initiating entity wishes to indicate that it intends to use multiplexing with SASL EXTERNAL it MUST respond to the empty <mux/> element by sending another empty <mux/> element qualified by the 'urn:xmpp:mux:0' namespace in reply. No stream restart is necessary. After indicating support for multiplexing by negotiating the mux stream feature, authentication can proceed. When using SASL EXTERNAL this is done by validating the certificate as detailed in Best Practices for Use of SASL EXTERNAL (XEP-0178) [7] except that every domain that is present in the certificate is now eligible for multiplexing without further negotiation. Further stream features (such as resource binding) still use the JID from the original connection (or from the authorization identity).
If a bidirectional s2s connection has been negotiated for this session using Bidirectional Server-to-Server Connections (XEP-0288) [8], negotiation of the mux stream feature also

[7] XEP-0178: Best Practices for Use of SASL EXTERNAL <https://xmpp.org/extensions/xep-0178.html>.
[8] XEP-0288: Bidirectional Server-to-Server Connections <https://xmpp.org/extensions/xep-0288.html>.

implies that the receiving entity SHOULD multiplex stanzas sent back to the initiating entity for all domains in the verified client certificate. If bidi is not negotiated then mux will need to be negotiated again when the original receiving entity establishes a connection with the original initiating entity.

## 4 Connection Authorization

Often it is not desirable to have one certificate containing every XMPP address or host managed by the server, or the use of SASL EXTERNAL may be impossible. In these cases the initiating entity may request authorization to send stanzas over an existing connection.
If the initiating entity has an authenticated connection to a server and wishes to send stanzas to another server that was listed in the original servers post-auth <mux/> stream feature it MAY establish an XMPP connection with the new server and verify that new server also lists the original server in its post-auth mux stream feature. If it does the initiating entity replies with a <mux/> element qualified by the 'urn:xmpp:mux:0' namespace with a shared secret as the payload and the host being selected included in the 'host' attribute. The old server then sends an IQ over its existing connection with the initiating entity containing the same mux element and secret, thereby confirming its relationship to the new server. If the client verifies that the secrets match it sends an empty IQ of type "result" in response to indicate success, otherwise the IQ response should be a "not-acceptable" stanza error (see RFC 6120 [9] §8.3.3.9).
For example, if the server montague.example wishes to establish a multiplexed connection with capulet.example and rooms.capulet.example the flow would look like this:

```
montague.example        capulet.example
----------------        ---------------
       |                          |
       |  [if necessary,          |
       |   perform DNS lookup     |
       |   on Target Domain,      |
       |   open TCP connection,   |
       |   and establish stream]  |
       | -----------------------> |             rooms.capulet.example
       |                          |             ---------------------
       |  [if necessary,          |                       |
       |   perform DNS lookup,    |                       |
       |   on Sender Domain,      |                       |
       |   open TCP connection,   |                       |
       |   and establish stream]  |                       |
       | -----------------------------------------------> |
       |                          |                       |
       |  send mux secret         |                       |
```

[9] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

```
|  -----------------------------------------------> |
|                          |                       |
|       send mux secret IQ |                       |
| <----------------------- |                       |
|                          |                       |
| send IQ response         |                       |
| -----------------------> |                       |
|                          |                       |
|                          |      close connection |
| <----------------------------------------------- |
```

The XML for this exchange would look like the following:

Listing 3: Initial connection between montague.example and capulet.example

```
<!-{}-
  Elided: a stream is negotiated between montague.example and capulet.
      example.
  After authentication is complete capulet.example advertises support
      for mux:
-{}->
<stream:features>
  <mux xmlns='urn:xmpp:mux:0'>
    <host>rooms.capulet.example</host>
  </mux>
</stream:features>

<!-{}-
  Negotiation proceeds and the mux stream feature is not selected.
      After
  negotiation is complete montague.example tries to establish a
      connection with
  rooms.capulet example and sends it a secret. The server responds
      with the same
  secret:
-{}->
<iq to="montague.example" from="capulet.example" type="set" id="
    1285152">
  <mux xmlns='urn:xmpp:mux:0'>secret</mux>
</iq>

<-{}- The server at montague.example indicates that the secret was
    verified. -{}->
<iq to="capulet.example" from="montague.example" type="result" id="
    1285152"/>
```

Listing 4: Secondary connection between montague.example and rooms.capulet.example

```
<!-{}-
```

```
  Elided: a stream is negotiated between montague.example and
  rooms.capulet.example. After authentication is complete rooms.
     capulet.example
  advertises support for mux:
-{}->
<stream:features>
  <mux xmlns='urn:xmpp:mux:0'>
    <host>capulet.example</host>
  </mux>
</stream:features>

<!-{}-
  The server at montague.example indicates that it wishes to authorize
     its
  existing connection with capulet.example:
-{}->
<mux xmlns='urn:xmpp:mux:0' host='capulet.example'>
  secret
</mux>

<!-{}-
  The server at rooms.capulet.example closes the connection gracefully
  if mux was established and begins using the connection between
  montague.example and capulet.example.
-{}->
</stream:stream>
```

The format of the secret is not specified however, see the Security Considerations section of this document for some suggestions.

## 5 Security Considerations

Some clients may send stanzas with no ”from” attribute specified and rely on the server to add the attribute before routing the stanza to its final destination. If multiplexing is used the lack of a from attribute indicates that the client is acting on behalf of the origin JID for the connection, just like normal, so clients MUST set the from attribute on any stanzas sent on behalf of any multiplexed host.

The format of mux secrets is undefined in this document, however, they MUST be unpredictable. Only the initiating entity should attribute any meaning (if indeed there is any) to the format of mux secrets. In particular the receiving entity MUST NOT rely on a specific format for the secret.

One suggestion for generating mux secrets is to generate a key that signs information about the stream. The format defined in Dialback Key Generation and Validation (XEP-0185) [10] is appropriate for this. If the mux secret is a signature it must protect against reuse by at least

[10] XEP-0185: Dialback Key Generation and Validation <https://xmpp.org/extensions/xep-0185.html>.

include a random secret generated with a cryptographically secure random number source, the origin JID, the JID of the server initially receiving the mux secret, and the stream ID for the stream the key will be authenticating (this is not the same stream as the receiving entity's JID). It is also RECOMMENDED that an expiration time be included in the key after which it is no longer valid.

# 6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [11].

# 7 XMPP Registrar Considerations

## 7.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:mux:0

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [12] shall add the foregoing namespace to the registries located at <https://xmpp.org/registrar/stream-features.html>, as described in Section 4 of XMPP Registrar Function (XEP-0053) [13].

```
<feature>
  <ns>urn:xmpp:mux:0</ns>
  <name>mux</name>
  <element>mux</element>
  <desc>Indicate support for connection multiplexing and transmit
      secret keys to a peer.</desc>
  <doc>Editor to add document reference if accepted</doc>
  <status>provisional</status>
</feature>
```

[11] 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/>.

[12] 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/>.

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

The XMPP Registrar [14] shall also add the foregoing namespace to the Jabber/XMPP Protocol Namespaces Registry located at <https://xmpp.org/registrar/namespaces.html>. Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [15] shall remove the provisional status from this registry entry.

```
<ns>
  <name>urn:xmpp:mux:0</name>
  <doc>Editor to add document reference if accepted</doc>
  <status>provisional</status>
</ns>
```

### 7.2 Namespace 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 XEP-0053.

## 8 Acknowledgements

Thanks to Jeremie Miller, Peter Saint-Andre, and Philipp Hancke for writing Server Dialback (XEP-0220) [16] from which some of the text and techniques used in this document were borrowed.

[14] 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/>.

[15] 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/>.

[16] XEP-0220: Server Dialback <https://xmpp.org/extensions/xep-0220.html>.