XEP-0288: Bidirectional Server-to-Server Connections

Philipp Hancke
mailto:fippo@andyet.com
xmpp:fippo@goodadvice.pages.de

Dave Cridland
mailto:dave.cridland@surevine.com
xmpp:dave.cridland@surevine.com

2016-10-17
Version 1.0.1

Status	Type	Short Name
Draft	Standards Track	bidi

This specification defines a protocol for using server-to-server connections in a bidirectional way such that stanzas are sent and received on the same TCP connection.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

RFC 3920 [1] restricted server-to-server communication in such a way that a server had to use one TCP connection for XML stanzas sent from the server to the peer, and another TCP connection (initiated by the peer) for stanzas from the peer to the server, for a total of two TCP connections. RFC 6120 [2] allows two servers to send stanzas in a bidirectional way, but does not define methods for explicitly signalling the usage thereof. This is accomplished herein.
While this may seem like a mere optimization that decreases the number of sockets used by an implementation or increases the performance of the server-to-server connection[3], it actually removes some of the practical barriers for the implementation of Multiplexing in Server Dialback (XEP-0220) [4].

2 Protocol

2.1 Stream Feature

If a server supports bidirectional server-to-server streams, it should inform the connecting entity when returning stream features during the stream negotiation process (both before and after TLS negotiation). This is done by including a <bidi/> element qualified by the 'urn:xmpp:features:bidi' namespace.

Listing 1: Stream features

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <bidi xmlns='urn:xmpp:features:bidi'/>
</stream:features>
```

If the initiating entity chooses to use TLS, STARTTLS negotiation MUST be completed before enabling bidirectionality.

2.2 Negotiation

To enable bidirectional communication, the connecting server sends a <bidi/> element qualified by the 'urn:xmpp:bidi' namespace. This SHOULD be done before either SASL negotiation or Server Dialback.

[1] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[2] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[3] In constrained environments, bidirectional server-to-server connections exhibit a reduced packet round trip time, see <http://www.isode.com/whitepapers/xmpp-performance-constrained.html>.
[4] XEP-0220: Server Dialback <https://xmpp.org/extensions/xep-0220.html>.

Listing 2: Connecting Server Requests Bidirectionality

```
<!-{}- Client -{}->
<bidi xmlns='urn:xmpp:bidi'/>
```

After enabling bidirectionality, the connecting server continues to authenticate via SASL or requests to send stanzas for a domain pair with Server Dialback. The receiving server MUST NOT send stanzas to the peer before it has authenticated via SASL, or the peer's identity has been verified via Server Dialback. Note that the receiving server MUST NOT attempt to verify a dialback key on the same connection where the corresponding request was issued.
Also note that the receiving server MUST only send stanzas for which it has been authenticated - in the case of TLS/SASL based authentication, this is the value of the stream's 'to' attribute, whereas in the case of Server Dialback this is the inverse of any domain pair that has been used in a dialback request.
Finally, once bidirectionality is enabled, the receiving server MAY initiate Server Dialback authentications for other domains it hosts to any domain authenticated to be hosted by the connecting server. In particular, it may initiate Target Piggybacking for any target domain that has successfully been used as a source domain by the connecting server. Note that this implies that a connecting server that uses bidi and dialback MUST support dialback error conditions as defined in XEP 0220[5].

3 Examples

This section shows two complete examples of bidirectional streams, the first example uses SASL EXTERNAL, the second uses Server Dialback.

Listing 3: Bidirectional Streams with SASL Authentication

```
<!-{}- Client -{}->
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
               xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               to='montague.lit' from='capulet.lit'
               xml:lang='en' version='1.0'>
<!-{}- Server -{}->
<stream:stream xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               xmlns:stream='http://etherx.jabber.org/streams'
                  xml:lang='en'
               id='65b30434afd7646699d077f7affcb2c120c48e18'
               from='montague.lit' to='capulet.lit' version='1.0'>
<!-{}- Server -{}->
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

[5] Ideally, support for dialback errors would be signalled by a proper extension mechanism such as <stream:features/>. However, these are currently only sent from the receiving server to the connecting server and can therefore not be used for signalling support for dialback errors in the other direction.

```
  <bidi xmlns='urn:xmpp:features:bidi'/>
</stream:features>
<!-{}- Client -{}->
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
<!-{}- Server -{}->
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
<!-{}- Client -{}->
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
               xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               to='montague.lit' from='capulet.lit'
               xml:lang='en' version='1.0'>
<!-{}- Server -{}->
<stream:stream xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               xmlns:stream='http://etherx.jabber.org/streams'
                  xml:lang='en'
               id='b5cd769b1dc292c6f6557fe76cabc4d112333f9a'
               from='montague.lit' to='capulet.lit' version='1.0'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>EXTERNAL<mechanism>
  </mechanisms>
  <bidi xmlns='urn:xmpp:features:bidi'/>
</stream:features>
<!-{}- Client -{}->
<bidi xmlns='urn:xmpp:bidi'/>
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='EXTERNAL'>
  Y2FwdWxldC5saXQ=
</auth>
<!-{}- Server -{}->
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
<!-{}- Client -{}->
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
               xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               to='montague.lit' from='capulet.lit'
               xml:lang='en' version='1.0'>
<!-{}- Server -{}->
<stream:stream xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               xmlns:stream='http://etherx.jabber.org/streams'
                  xml:lang='en'
               id='b5cd769b1dc292c6f6557fe76cabc4d112333f9a'
               from='montague.lit' to='capulet.lit' version='1.0'>
<stream:features/>
<!-{}- At this point, S is allowed to send C stanzas from montague.lit
    to capulet.lit
     since that is the value of 'from' in the stream open sent by C
        above.
  -{}->
<!-{}- Client -{}->
<iq from='juliet@capulet.lit/balcony' to='montague.lit' type='get'
```

```
  id='8dfc70af'><query xmlns='urn:xmpp:ping'/></iq>
<!-{}- Server -{}->
<iq from='montague.lit' to='juliet@capulet.lit/balcony' type='result'
  id='8dfc70af'><query xmlns='urn:xmpp:ping'/></iq>
```

Listing 4: Bidirectional Streams with Server Dialback

```
<!-{}- Client -{}->
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
               xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               to='montague.lit' from='capulet.lit'
               xml:lang='en' version='1.0'>
<!-{}- Server -{}->
<stream:stream xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               xmlns:stream='http://etherx.jabber.org/streams'
                  xml:lang='en'
               id='65b30434afd7646699d077f7affcb2c120c48e18'
               from='montague.lit' to='capulet.lit' version='1.0'>
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <bidi xmlns='urn:xmpp:features:bidi'/>
</stream:features>
<!-{}- Client -{}->
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
<!-{}- Server -{}->
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
<!-{}- Client -{}->
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
               xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               to='montague.lit' from='capulet.lit'
               xml:lang='en' version='1.0'>
<!-{}- Server -{}->
<stream:stream xmlns='jabber:server' xmlns:db='jabber:server:dialback'
               xmlns:stream='http://etherx.jabber.org/streams'
                  xml:lang='en'
               id='b5cd769b1dc292c6f6557fe76cabc4d112333f9a'
               from='montague.lit' to='capulet.lit' version='1.0'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
  <bidi xmlns='urn:xmpp:features:bidi'/>
</stream:features>
```

Listing 5: Stream Setup before TLS

```
<!-{}- Client -{}->
<bidi xmlns='urn:xmpp:bidi'/>
<db:result from='capulet.lit' to='montague.lit'>
  e3f5cf21f12749ef2cf59269bc0118f35bc46b26</db:result>
<!-{}- Server -{}->
<db:result from='montague.lit' to='capulet.lit' type='valid'/>
```

```
<!-{}- At this point S may send from montague.lit to capulet.lit.-{}->
<!-{}- Client -{}->
<iq from='juliet@capulet.lit/balcony' to='montague.lit' type='get'
  id='8dfc70af'><query xmlns='urn:xmpp:ping'/></iq>
<!-{}- Server -{}->
<iq from='montague.lit' to='juliet@capulet.lit/balcony' type='result'
  id='8dfc70af'><query xmlns='urn:xmpp:ping'/></iq>
<db:result from='conference.montague.lit' to='capulet.lit'>
     1bac3ef56fed987cfe098c9785c654a5476ed765</db:result>
<!-{}- The above is also legal - S attempts to authenticate as
     a different domain as well, presumably a MUC domain.
     note that S can do this form of multiplexing regardless
     of the support for dialback errors since that was required by RFC
          3920
  -{}->
<!-{}- Client -{}->
<db:result from='capulet.lit' to='conference.montague.lit' type='valid
    '/>
<!-{}- Now S can send as conference.m.l as well as C sending to that
    domain.
-{}->
```

4 Security Considerations

This specification introduces no security considerations above and beyond those discussed in RFC 6120 or XEP-0220. Note that the impact of the ”unsolicited server dialback” attack described in XEP-0220 is considerably larger for bidirectional streams, e.g. a vulnerability which allows spoofing might also route messages to the wrong targets. Additionally, dialback elements with a ”type” attribute also need to be handled in incoming connections.

5 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [6].

[6] 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/>.

6 XMPP Registrar Considerations

6.1 Protocol Namespaces

The XMPP Registrar [7] includes 'urn:xmpp:bidi' in its registry of protocol namespaces (see <https://xmpp.org/registrar/namespaces.html>).

6.2 Stream Features

The XMPP Registrar includes 'urn:xmpp:features:bidi' in its registry of stream features (see <https://xmpp.org/registrar/stream-features.html>).

7 XML Schema

7.1 Bidi

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0288: http://www.xmpp.org/extensions/xep-0288.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='bidi' type='empty'/>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

[7] 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/>.

7.2 Stream Feature

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

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

  <xs:element name='bidi' type='empty'/>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

8 Acknowledgements

Thanks to Justin Karneges and Torje Henriksen.