Product Development : Connection Manager JEP

This page last changed on Jun 02, 2006 by gaston.

Introduction

A core scalability challenge for XMPP servers is handling very large numbers of persistent TCP/IP connections. This JEP attempts to address that challenge by defining an architecture for delegating connection management to simple Connection Manager processes. Each Connection Manager is responsible for accepting and handling client connections by multiplexing to a small pool of connections (or single connection) with a central XMPP server. Many Connection Managers can be used with a single XMPP server to allow scaling to many (tens of thousands) client connections. The protocol is based on JEP-114: Jabber Component Protocol, TLS as defined in the RFC3920 - XMPP Core and optionally leverages JEP-0138: Stream Compression.

Editorial Note
The JEP may be later split into two smaller JEPs. The Connection Manager to Server Connections section essentially defines a more modern version of JEP-114: Jabber Component Protocol that can replace the old JEP. The Client Connection Handling section could be re-crafted as an extension to the updated external component JEP for multiplexing connection streams.

The new evolution of JEP-114: Jabber Component Protocol would support secure connections (TLS) and stream compression. Authentication is still based on a common shared secret that is used during a handshake. So far SASL authentication is not being used but it may be included in a future version.

Connection Manager to Server Connections

Before accepting client connections, a Connection Manager must connect to an XMPP server. Optionally, the connection can use TLS or compression. After connecting, the Connection Manager must authenticate using a shared secret. Finally, the Connection Manager recieves a list of stream features from the server that should be presented to connecting clients.

Opening Connection (Connection Manager - Server)

Example 1. Connection Manager sends stream header to server

```
<stream:stream
    xmlns='jabber:connectionmanager'
   xmlns:stream='http://etherx.jabber.org/streams'
    to='manager1Name/conn1'>
```

Example 2. Server replies with stream header, including StreamID

```
<stream:stream
   xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:connectionmanager'
```

```
    from='manager1Name/conn1'
    id='3BF96D32'>
```

Example 3. Server sends the STARTTLS extension to Connection Manager offering Stream Compression Feature if available:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
  <compression xmlns='http://jabber.org/features/compress'>
   <method>zlib</method>
  </compression>
</stream:features>
```

TLS Negotiation (Connection Manager - Server)

Example 4: Connection Manager sends the STARTTLS command to server:

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

Example 5: Server informs Connection Manager that it is allowed to proceed:

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Example 5 (alt): Server informs Connection Manager that TLS negotiation has failed and closes both stream and TCP connection:

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

Example 6: Connection Manager and server attempt to complete TLS negotiation over the existing TCP connection.

Example 7: If TLS negotiation is successful, Connection Manager initiates a new stream to server:

```
<stream:stream
    xmlns='jabber:connectionmanager'
   xmlns:stream='http://etherx.jabber.org/streams'
    to='manager1Name/conn1'>
```

Example 7 (alt): If TLS negotiation is unsuccessful, server closes TCP connection.

Example 8: Server responds by sending a stream header to Connection Manager along with any available stream features:

```
<stream:stream
   xmlns:stream='http://etherx.jabber.org/streams'
```

```
   xmlns='jabber:connectionmanager'
    from='manager1Name/conn1'
    id='3BF96D32'>
<stream:features>
  <compression xmlns='http://jabber.org/features/compress'>
   <method>zlib</method>
  </compression>
</stream:features>
```

Compression Negotiation (Connection Manager - Server)

Example 9. Connection Manager Requests Stream Compression

```
<compress xmlns='http://jabber.org/protocol/compress'>
 <method>zlib</method>
</compress>
```

Example 10. Server Entity Acknowledges Stream Compression

```
<compressed xmlns='http://jabber.org/protocol/compress'/>
```

Compression is started at this point

Example 11. Connection Manager Initiates New Stream

```
<stream:stream
   xmlns='jabber:connectionmanager'
   xmlns:stream='http://etherx.jabber.org/streams'
    to='manager1Name/conn1'>
   to='shakespeare.lit'>
```

Example 12: Server responds by sending a stream header to Connection Manager:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
   xmlns='jabber:connectionmanager'
    from='manager1Name/conn1'
    id='3BF96D32'>
```

Connection Manager Authentication (Connection Manager - Server)

Example 13. Connection Manager sends handshake element

```
<handshake>aaee83c26aeeafcbabeabfcbcd50df997e0a2a1e</handshake>
```

Example 14. Server sends empty handshake element to acknowledge success

<handshake/>

Client Connection Handling

Stream Feature Discovery (Connection Manager - Server)

Connection Managers MUST handle initial stream headers of newly connected clients. Therefore, Connection Managers need to know which stream features should be offered to new clients. Once a Connection Manager has initially authenticated with the server, it MUST receive from the server the connections options available for clients. Whenever the connections options available for clients were modified in the server, the server MUST publish again the information to all connected Connection Managers. Note that the server MAY not need to push this information when a second or more connection is established from a connection manager to the server.

The connections options available for clients includes:

- if TLS is available, optional or required
- SASL mechanisms available before TLS is negotiated
- if compression is available
- if Non-SASL authentication is available
- if In-Band Registration is available

Example 15. Server pushes configuration to connection manager

```
<iq from='example.com' id='someid' type='set'>
  <configuration xmlns='http://jabber.org/protocol/connectionmanager'>
    <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
      <required/>
    </starttls>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <mechanism>DIGEST-MD5</mechanism>
      <mechanism>PLAIN</mechanism>
    </mechanisms>
    <compression
xmlns="http://jabber.org/features/compress"><method>zlib</method></compression>
    <auth xmlns='http://jabber.org/features/iq-auth'/>
    <register xmlns='http://jabber.org/features/iq-register'/>
  </configuration>
</iq>
```

Features that are not supported by the server will not be included in the IQ stanza. Initial version of Connection Managers will not support the old SSL method.

Example 16. Connection manager confirms reception

```
<iq from='manager1Name/conn1' to='example.com' id='someid' type='result'/>
```

Opening Client Connection (Client - Connection Manager - Server)

As clients connect to the Connection Manager, the Connection Manager must notify the XMPP servers of those new connections. Notifications MUST also be sent to the server when client connections are closed.

After a client sent the **initial opening stream** stanza Connection Managers MUST inform the server that a new client session has been created.

Example 17: Connection Manager informs the server that a client session has been created

```
<iq from='manager1Name/conn1' to='example.com' id='someid' type='set'>
   <session xmlns='http://jabber.org/protocol/connectionmanager' id='streamID'>
      <create/>
   </session>
</iq>
```

Example 18. Server confirms reception of notification

```
<iq from='example.com' id='someid' type='result'/>
```

Closing Client Connection (Client - Connection Manager - Server)

Client sessions may be terminated for several reasons:

- Client decided to terminate connection
- Client unexpectedly went down
- A Connection Manager was stopped
- A Connection Manager unexpectedly went down
- A Client connection was closed from the server
- The server was stopped

Client decided to terminate connection

Example 19. Users terminates connection

```
</stream:stream>
```

Example 20: Connection Manager informs the server that a client session has been closed

```
<iq from='manager1Name/conn1' to='example.com' id='someid' type='set'>
   <session xmlns='http://jabber.org/protocol/connectionmanager' id='streamID'>
      <close/>
   </session>
</iq>
```

Example 21. Server confirms reception of notification

```
<iq from='example.com' id='someid' type='result'/>
```

Client unexpectedly went down

Example 22: Connection Manager informs the server that a client session has been closed

```
<iq from='manager1Name/conn1' to='example.com' id='someid' type='set'>
   <session xmlns='http://jabber.org/protocol/connectionmanager' id='streamID'>
      <close/>
   </session>
</iq>
```

Example 23. Server confirms reception of notification

```
<iq from='example.com' id='someid' type='result'/>
```

A Connection Manager was stopped

The Connection Manager needs to terminate each connected client session and inform the server that the connection is no longer available.

Example 24. Connection Manager terminates the client session

```
<stream:error>
   <system-shutdown xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

Example 25: Connection Manager informs the server that the Connection Manager is being stopped

```
<stream:error>
   <system-shutdown xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

The server will send unavailable presences and terminate client sessions that were originated from the Connection Manager that was stopped.

A Connection Manager unexpectedly went down

Clients will notice the problem and may want to reconnect again using other Connection Managers or wait until the faulty Connection Manager is up again.

On the other hand, the server MUST detect the problem and terminate client sessions that were originated using the no longer available Connection Manager. As defined in RFC3921 the server MUST send an unavailable presence for those clients that are no longer available.

A Client connection was closed from the server

Example 26: Server informs the Connection Manager that a client session has been closed

```
<iq from='example.com' to='manager1Name' id='someid' type='set'>
   <session xmlns='http://jabber.org/protocol/connectionmanager' id='streamID'>
      <close/>
   </session>
</iq>
```

Example 27. Connection Manager confirms reception of notification

```
<iq from='manager1Name/conn1' id='someid' type='result'/>
```

The server was stopped

The Server needs to close the stream of all connected entity.

Example 28. Server terminates the Connection Manager sessions

```
<stream:error>
   <system-shutdown xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

The Connection Manager detects that the server is shutting down so it needs to terminate each connected client session.

Example 29. Connection Manager terminates the client session

```
<stream:error>
   <system-shutdown xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

At this point Connection Managers MAY be automatically stopped or may keep running and trying to connect to the server whenever it is started up again.

Client Connections to Connection Managers

Use of TLS (Client - Connection Manager)

Clients should use the same protocol as specified in RFC3920. Connection Managers MUST handle the whole interaction without sending any stanzas to the server. The only stanza that Connection Managers MUST send to the server is to indicate that a new client session has been created as defined in Opening Client Connection. Moreover, Connection Managers SHOULD offer the stream features that were published from the server.

Use of SASL (Client - Connection Manager - Server)

SASL authentication is performed by the server. Therefore, Connection Managers should interact with the server to successfully handle authentication requests sent by clients. The protocol used for SASL negotiation is the same one that is defined in RFC3920 with some minor modifications.

Stream headers and stream features are handled by Connection Managers just like we do for TLS negotiation. However <auth> and <response> stanzas sent from clients should be wrapped and forwarded to the server. Moreover, <challenge>, <failure> and <success> stanzas should also be wrapped when sent from the server to Connection Managers.

The reason why we need to wrap all those stanzas is that none of them have information about the client that is trying to authenticate with the server (i.e. stanzas do not have a FROM or TO attribute). And since the connection/s between Connection Managers and the server are multiplexed we need to identify the user that is trying to authenticate.

Example 30. Client selects an authentication mechanism:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
    mechanism='DIGEST-MD5'/>
```

Example 31. Connection Manager forwards selected authentication method to the server:

```
<route from='manager1Name/conn1' to='example.com' streamid='streamID'>
   <auth mechanism="DIGEST-MD5" xmlns="urn:ietf:params:xml:ns:xmpp-sasl"/>
</route>
```

The ID attribute of the manage element represents the stream ID assigned to the client connection. The server will use the stream ID to check if a new Client Session needs to be created for the specified stream ID.

Example 32. Server responds to Connection Manager the challenge that needs to be sent to the Client:

```
<route from='example.com' streamid='streamID'>
   <challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb3A9ImF1dGgi
      LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg==
   </challenge>
</route>
```

Example 33. Connection Manager sends a [BASE64]Josefsson, S., The Base16, Base32, and Base64 Data Encodings, July 2003. encoded challenge to client:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb3A9ImF1dGgi
LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg==
</challenge>
```

Example 32 (alt). Server returns error to Connection Manager:

```
<route from='example.com' streamid='streamID'>
   <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <incorrect-encoding/>
   </failure>
</route>
```

Example 33 (alt). Connection Manager returns error to client:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <incorrect-encoding/>
</failure>
</stream:stream>
```

Example 34. Client sends a [BASE64]Josefsson, S., The Base16, Base32, and Base64 Data Encodings, July 2003. encoded response to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29tZXJlYWxtIixub25jZT0i
T0E2TUc5dEVRR20yaGgiLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAw
MDAwMDAxLHFvcD1hdXRoLGRpZ2VzdC11cmk9InhtcHAvZXhhbXBsZS5jb20i
LHJlc3BvbnNlPWQzODhkYWQ5MGQ0YmJkNzYwYTE1MjMyMWYyMTQzYWY3LGNo
YXJzZXQ9dXRmLTgK
</response>
```

Example 35. Connection Manager forwards response to server:

```
<route from='manager1Name/conn1' to='example.com' streamid='streamID'>
   <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29tZXJlYWxtIixub25jZT0i
      T0E2TUc5dEVRR20yaGgiLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAw
      MDAwMDAxLHFvcD1hdXRoLGRpZ2VzdC11cmk9InhtcHAvZXhhbXBsZS5jb20i
      LHJlc3BvbnNlPWQzODhkYWQ5MGQ0YmJkNzYwYTE1MjMyMWYyMTQzYWY3LGNo
      YXJzZXQ9dXRmLTgK
   </response>
</route>
```

Example 36. Server informs Connection Manager of successful client authentication:

```
<route from='example.com' streamid='streamID'>
   <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
</route>
```

Example 37. Connection Manager informs client of successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Example 36 (alt). Server informs Connection Manager of failed client authentication:

```
<route from='example.com' streamid='streamID'>
```

```
   <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <incorrect-encoding/>
   </failure>
</route>
```

Example 37 (alt). Connection Manager returns error to client:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
 <temporary-auth-failure/>
</failure>
</stream:stream>
```

Resource Binding (Client - Connection Manager - Server)

Clients that need to bind a specific resource to their streams will send a <bind> stanza as defined in RFC3920. The <bind> stanza does not contain any information indicating that it belongs to a certain stream. Therefore, Connection Managers MUST wrap the <bind> stanza before sending it to the server.

Example 38. Connection Manager forwards resource binding stanza to server:

```
<route from='manager1Name/conn1' to='example.com' streamid='streamID'>
   <iq xmlns='jabber:client' type='set' id='bind_1'>
      <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
         <resource>someresource</resource>
      </bind>
   </iq>
</route>
```

Example 39. Server informs Connection Manager of result:

```
<route from='example.com' streamid='streamID'>
   <iq xmlns='jabber:client' type='result' id='bind_1'>
      <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
         <jid>somenode@example.com/someresource</jid>
      </bind>
   </iq>
</route>
```

Session Establishment (Client - Connection Manager - Server)

Clients that need to establish a session on a server in order to engage in the expected instant messaging and presence activities will send a <session> stanza as defined in RFC3921. The <session> stanza does not contain any information indicating that it belongs to a certain stream. Therefore, Connection Managers MUST wrap the <session> stanza before sending it to the server.

Example 40. Connection Manager forwards session establishment stanza to server:

```
<route from='manager1Name/conn1' to='example.com' streamid='streamID'>
   <iq xmlns='jabber:client' type='set' id='sess_1'>
```

```
      <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
   </iq>
</route>
```

Example 41. Server informs Connection Manager of result:

```
<route from='example.com' streamid='streamID'>
   <iq xmlns='jabber:client' type='result' id='sess_1'/>
</route>
```

Regular traffic flow (Client - Connection Manager - Server)

After a client session has been authenticated by the server any packet sent by the client MUST be wrapped and forwarded to the server. The same applies for traffic originated from the server.

Connection Manager fails to deliver a packet

A user sends a packet to another user. The server may see that the target user is available and decide to forward the packet to a connection manager. The connection manager may fail to deliver the packet because the client just disconnected or because the connection was down and it went unnoticed.

In this case Connection Managers MUST inform the server that the packet failed to be delivered. This case applies to:

- IQ packets of type SET or GET
- Message packets (not of type error)

For IQ packets of type SET or GET the Connection Manager should return an IQ of type ERROR containing a stanza error of type <unexpected-request/>. If the packet failed to be delivered is a Message then the Connection Manager SHOULD return the original packet wrapped by an IQ packet whose child element is <session> qualified by the namespace http://jabber.org/protocol/connectionmanager.

Example 42. Connection Manager informs the server that a message failed to be delivered

```
<iq from='manager1Name/conn1' to='example.com' id='someid' type='set'>
   <session xmlns='http://jabber.org/protocol/connectionmanager' id='streamID'>
      <failed>
         <message>...</message>
      </failed>
   </session>
</iq>
```

Example 43. Server confirms Connection Manager of notification

```
<iq from='example.com' id='someid' type='result'/>
```

The server SHOULD apply the same logic that is used when a user sends a message to another user that is offline. That means, that the message failed to be delivered may be stored offline for later delivery or

may be bounced to the sender.

Error conditions

The following section specifies the error code to return under different possible error conditions. In general the conditions apply to problems that might occur when wrapping packets.

Child element of wrapper packet does not have an id attribute

If the child element of the IQ packet of type SET does not have an "id" attribute, that specifies the stream ID, then the server MUST return an <bad-request/> error, which SHOULD also contain a connection manager specific error condition of <id-required/>.

Example 44. No stream ID was specified

```
<iq from='example.com' to=manager1Name' id='someid' type='error'>
   <original child element goes here/>
   <error type='modify'>
      <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      <id-required xmlns='http://jabber.org/protocol/connectionmanager#errors'/>
   </error>
</iq>
```

No session was found with the specified stream ID

If the provided "id" attribute of the child element refers to a non-existent session then the server MUST return an <item-not-found/> error.

Example 45. Invalid stream ID was specified

```
<iq from='example.com' to=manager1Name' id='someid' type='error'>
   <original child element goes here/>
   <error type='cancel'>
      <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
   </error>
</iq>
```

Zero or many elements were included in the <send> or <failed> elements

IQ packets that are wrapping another packet MUST only wrap one stanza. If no stanza is being wrapped or many of them are being wrapped then the server MUST return an <bad-request/> error, which

SHOULD also contain a connection manager specific error condition of <invalid-payload/>.

Example 46. Invalid wrapper packet was sent

```
<iq from='example.com' to=manager1Name' id='someid' type='error'>
   <original child element goes here/>
   <error type='cancel'>
      <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      <invalid-payload xmlns='http://jabber.org/protocol/connectionmanager#errors'/>
   </error>
</iq>
```

Other than a message was included in the <failed> element

When Connection Managers fail to deliver a Message stanza they MUST inform the server of the failure. Moreover, if the wrapped stanza inside the <failed> element is not of type Message then the server MUST return an <bad-request/> error, which SHOULD also contain a connection manager specific error condition of <unknown-stanza/>.

Example 47. Invalid wrapper packet was sent

```
<iq from='example.com' to=manager1Name' id='someid' type='error'>
   <original child element goes here/>
   <error type='cancel'>
      <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      <unknown-stanza xmlns='http://jabber.org/protocol/connectionmanager#errors'/>
   </error>
</iq>
```

Connection Manager sent an IQ packet of type GET

Wrapper packets MUST be IQ packets of type SET. If an IQ packet of type GET is used then a <bad-request/> error MUST be returned.

Future direction

The JEP can be modified to support not only client connections but also server-to-server communications. The sections that we would need to modify are the ones where Connection Managers indicate the server that a session needs to be created or closed. The <session> "verb" may be renamed to <client-session> and the new "verb" <server-session> may be created.

Summary

Packets that are handled by Connection Managers without any server involvement except for indicating that a new client session has been created:

- Stream headers
- starttls
- compress

Connection Managers use IQ packets to inform the server that a session has been created or closed or to indicate that a packet failed to be delivered. The server uses IQ packets to inform Connection Managers that a given client session needs to be closed.

Once a session has been created, clients traffic MUST be wrapped using the <route> element before being forwarded to the server. Server traffic whose target are clients MUST also be wrapped using the <route> element. The ID attribute included in the <route> element represents the stream ID assigned to the client connection.