XEP-0305: XMPP Quickstart

Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

2013-03-01
Version 0.3

Status	Type	Short Name
Deferred	Standards Track	N/A

This document defines methods for speeding the process of connecting or reconnecting to an XMPP server.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Establishing an XMPP session can require a fairly large number of round trips between the initiating entity and the receiving entity. However, in many deployment scenarios it would be helpful to reduce the number of round trips and therefore the time needed to establish a session. This document describes protocol optimizations and best practices to do just that.

2 Preparing to Connect

In accordance with RFC 6120 [1], before attempting to establish a stream over TCP the initiating entity needs to determine the IP address and port at which to connect, usually by means of DNS lookups as described in Section 3.2 of RFC 6120. Implementations SHOULD cache the results of DNS lookups in order to avoid this step whenever possible. Similar considerations apply to connections established over one of the HTTP bindings, i.e., either BOSH (see BOSH (XEP-0124) [2] and XMPP Over BOSH (XEP-0206) [3]) or WebSocket (see RFC 6455 [4] and XMPP Over WebSocket [5]).

XMPP applications SHOULD cache whatever information they can about the peer, especially stream features data and Service Discovery (XEP-0030) [6] information. To facilitate such caching, servers SHOULD include Entity Capabilities (XEP-0115) [7] data in stream features as shown in Section 6.3 of XEP-0115. Note that for maximum benefit the server MUST include all of the stream features it supports in its replies to "disco#info" queries (i.e., not advertise such features only during stream establishment).

XMPP clients SHOULD cache roster information, and servers SHOULD make such caching possible, using Roster Versioning (XEP-0237) [8] as subsequently included in Section 2.1.1 of RFC 6121 [9].

3 Pipelining

The primary method of speeding the connection process is pipelining of requests, along the lines of RFC 2920 [10] and the QUICKSTART extension proposed for SMTP (The QUICKSTART

[1] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[2] XEP-0124: Bidirectional-streams Over Synchronous HTTP <https://xmpp.org/extensions/xep-0124.html>.
[3] XEP-0206: XMPP Over BOSH <https://xmpp.org/extensions/xep-0206.html>.
[4] RFC 6455: The WebSocket Protocol <http://tools.ietf.org/html/rfc6455>.
[5] An XMPP Sub-protocol for WebSocket <https://datatracker.ietf.org/doc/draft-ietf-xmpp-websocket/>. Work in progress.
[6] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[7] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.
[8] XEP-0237: Roster Versioning <https://xmpp.org/extensions/xep-0237.html>.
[9] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[10] RFC 2920: SMTP Service Extension for Command Pipelining <http://tools.ietf.org/html/rfc2920>.

SMTP service extension [11]). The application of similar principles to XMPP was originally suggested by Tony Finch in February 2008 in a message to the standards@xmpp.org discussion list <http://mail.jabber.org/pipermail/standards/2008-February/017966.html>.
In essence, pipelining relies on two assumptions:

1. The parties to a stream can proactively send multiple XMPP-related "commands" in a single TCP packet (as one simple example, the receiving entity can send both the response stream header and stream features in a single packet).

2. The features that the receiving entity supports (e.g., stream features and SASL mechanisms) are stable over time, which means the initiating entity can assume support for certain features and send certain XMPP-related commands without discovering at the time of each connection attempt that the receiving entity supports them.

Together, these assumptions enable the parties to reduce the number of round trips needed to complete the stream negotiation process by "pipelining" XMPP-related commands over the stream.
Note well that pipelining at the XMPP layer is not to be confused with HTTP pipelining, which was added to HTTP in version 1.1 and which is not encouraged when using the HTTP bindings for XMPP.

4 Discovery

If an XMPP server supports pipelining, it MUST advertise a stream feature of <pipelining xmlns='urn:xmpp:features:pipelining'/>.
As noted, a server SHOULD also include its entity capabilities data in stream features as shown in Section 6.3 of XEP-0115.

5 TCP Binding

If both parties support pipelining, they can proceed as follows over the TCP binding (the examples use the XML from Section 9.1 of RFC 6120 for the client-server stream establishment, but the same principles apply to server-to-server streams).
In the client-to-server half of the first exchange, the client assumes that the server supports the XMPP STARTTLS extension so it pipelines its initial stream header, the <starttls/> command, and the TLS ClientHello message.

Listing 1: Client Initiation

```
<stream:stream
```

[11] The QUICKSTART SMTP service extension <http://tools.ietf.org/id/draft-fanf-smtp-quickstart-01.txt>. Work in progress.

```
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
TLS ClientHello
```

In the server-to-client half of the first exchange, the server pipelines its response stream header, stream features advertisement, STARTTLS <proceed/> response, and TLS ServerHello messages (which might include ServerHello, Certificate, ServerKeyExchange, CertificateRequest, and ServerHelloDone -- see RFC 5246 [12] for details).

Listing 2: Server Response to Initiation

```
<stream:stream
  from='im.example.com'
  id='t7AMCin9zjMNwQKDnplntZPIDEI='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
  <pipelining xmlns='urn:xmpp:features:pipelining'/>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://prosody.im/'
     ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
</stream:features>
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
TLS ServerHello
```

Without pipelining, the foregoing exchange would require 3 round trips; with pipelining it requires 1 round trip.

Now the parties complete the TLS negotiation (i.e., some combination of the TLS messages specified in RFC 5246); for our purposes we don't count these round trips because they are the same no matter whether we use pipelining or not.

At the end of the TLS negotiation, the server knows that the client will need to restart the stream so it proactively attaches its response stream header and stream features in the same TCP packet at the TLS Finished message, thus starting the next exchange.

[12] RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 <http://tools.ietf.org/html/rfc5246>.

Listing 3: Server Proactively Restarts Stream and Sends Stream Features

```
TLS Finished
<stream:stream
  from='im.example.com'
  id='vgKi/bkYME8OAj4rlXMkpucAqe4='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
  <pipelining xmlns='urn:xmpp:features:pipelining'/>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://prosody.im/'
     ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
</stream:features>
```

In response, the client pipelines its initial stream header with the command for initiating the SASL authentication process (including, if appropriate for the SASL mechanism used, the "initial response" data as explained in Section 6.3.10 of RFC 6120).

Listing 4: Client Initiates SASL Authentication

```
<stream:stream
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl"
      mechanism="SCRAM-SHA-1">
  biwsbj1qdWxpZXQscj1vTXNUQUF3QUFBQU1BQUFBTlAwVEFBQUFBQUJQVTBBQQ==
</auth>
```

Without pipelining, the second exchange would require another 2 round trips; with pipelining it requires only 1.

At this point the client and server might exchange multiple SASL-related messages, depending on the SASL mechanism in use. Because this specification does not attempt to reduce the number of round trips involved in the challenge-response sequence, we do not describe these exchanges here.

When the client suspects that it is sending its final SASL response, with pipelining it appends

an initial stream header and resource binding request.

Listing 5: Client Sends Final SASL Response with Stream Header and Bind Request

```
<response xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
  Yz1iaXdzLHI9b01zVEFBd0FBQUFNQUFBQU5QMFRBQUFBQUFCUFUwQUFlMTI0N
  jk1Yi02OWE5LTRkZTYtOWMzMC1iNTFiMzgwOGM1OWUscD1VQTU3dE0vU3ZwQV
  RCa0gyRlhzMFdEWHZKWXc9
</response>
<stream:stream
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<iq id='yhc13a95' type='set'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>balcony</resource>
  </bind>
</iq>
```

The server then informs the client of SASL success (including "additional data with success" as explained in Section 6.3.10 of RFC 6120), sends a response stream header and stream features, and informs the client of successful resource binding.

Listing 6: Server Accepts Bind Request

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  dj1wTk5ERlZFUXh1WHhDb1NFaVc4R0VaKzFSU289
</success>
<stream:stream
  from='im.example.com'
  id='gPybzaOzBmaADgxKXu9UClbprp0='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
  <sm xmlns='urn:xmpp:sm:3'/>
  <pipelining xmlns='urn:xmpp:features:pipelining'/>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://prosody.im/'
     ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
</stream:features>
<iq id='yhc13a95' type='result'>
```

```
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <jid>
      juliet@im.example.com/balcony
    </jid>
  </bind>
</iq>
```

Without pipelining, this exchange would require another 3 round trips; with pipelining it requires only 1.
Therefore, without pipelining the XMPP exchanges for stream establishment require at least 6 round trips (and perhaps more depending on the SASL mechanism used); with pipelining the minimum number of round trips is 3.
Naturally, for typical client-to-server sessions, additional round trips are needed so that the client can gather service discovery information, retrieve the roster, etc. As noted, these steps can be reduced or eliminated by using entity capabilities and roster versioning.

6 HTTP Bindings

In the HTTP bindings (BOSH and WebSocket) channel encryption occurs at the HTTP layer and therefore the first exchange shown above for the TCP binding is not used.
For now, this section focuses on BOSH. A future version of this document will discuss WebSocket (once draft-moffitt-xmpp-over-websocket has been updated to include examples).
When pipelining is used, a BOSH client can include its XMPP authentication (SASL) request in the BOSH session creation request, as shown in the following example.

Listing 7: BOSH Session Request with Pipelined SASL Authentication Request

```
POST /webclient HTTP/1.1
Host: httpcm.jabber.org
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 104

<body content='text/xml; charset=utf-8'
      from='user@example.com'
      hold='1'
      rid='1573741820'
      to='example.com'
      route='xmpp:example.com:9999'
      secure='true'
      wait='60'
      xml:lang='en'
      xmpp:version='1.0'
      xmlns='http://jabber.org/protocol/httpbind'
      xmlns:xmpp='urn:xmpp:xbosh'>
  <auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl"
```

```
        mechanism="SCRAM-SHA-1">
    biwsbj1qdWxpZXQscj1vTXNUQUF3QUFBQU1BQUFBTlAwVEFBQUFBQUJQVTBBQQ==
  </auth>
</body>
```

Note: If the client does not expect to receive a SASL challenge from the server but simply a success or failure notification (e.g., when using a simpler SASL mechanism such as PLAIN RFC 4616 [13]), then it can also pipeline its XMPP resource binding request with the BOSH session creation request.
The BOSH connection manager then returns a session creation response with a pipelined SASL authentication response.

Listing 8: Session Creation Response with Pipelined SASL Authentication Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 674

<body wait='60'
      inactivity='30'
      polling='5'
      requests='2'
      hold='1'
      from='example.com'
      accept='deflate,gzip'
      sid='SomeSID'
      secure='true'
      charsets='ISO_8859-1 ISO-2022-JP'
      xmpp:restartlogic='true'
      xmpp:version='1.0'
      authid='ServerStreamID'
      xmlns='http://jabber.org/protocol/httpbind'
      xmlns:xmpp='urn:xmpp:xbosh'
      xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:features>
    <pipelining xmlns='urn:xmpp:features:pipelining'/>
    <c xmlns='http://jabber.org/protocol/caps'
       hash='sha-1'
       node='http://prosody.im/'
       ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <mechanism>SCRAM-SHA-1</mechanism>
      <mechanism>PLAIN</mechanism>
    </mechanisms>
  </stream:features>
  <challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
```

[13] RFC 4616: The PLAIN Simple Authentication and Security Layer (SASL) Mechanism <http://tools.ietf.org/html/rfc4616>.

```
    cj1vTXNUQUF3QUFBQU1BQUFBTlAwVEFBQUFBQUJQVTBBQWUxMjQ2OTViLTY5Y
    TktNGRlNi05YzMwLWI1MWIzODA4YzU5ZSxzPU5qaGtZVE0wTURndE5HWTBaaT
    AwTmpkbUxUa3hNbVV0TkRsbU5UTm1ORE5rTURNeixpPTQwOTY=
  </challenge>
</body>
```

Without pipelining, this exchange would require 2 round trips; with pipelining, it requires only 1.
If the SASL exchange involved a challenge, in its final SASL response the client includes an XMPP resource binding request (note that the BOSH <body/> wrapper includes a restart='true' attribute, instead of sending this in a new empty <body/> as shown in XEP-0206).

Listing 9: SASL Response with Pipelined XMPP Resource Binding Request

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 295

<body rid='1573741822'
      sid='SomeSID'
      xmpp:restart='true'
      xmlns='http://jabber.org/protocol/httpbind'
      xmlns:xmpp='urn:xmpp:xbosh'>
  <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    Yz1iaXdzLHI9b01zVEFBd0FBQUFNQUFBQU5QMFRBQUFBQUFCUFUwQUFlMTI0N
    jk1Yi02OWE5LTRkZTYtOWMzMC1iNTFiMzgwOGM1OWUscD1VQTU3dE0vU3ZwQV
    RCa0gyRlhzMFdEWHZKWXc9
  </response>
  <iq id='bind_1'
      type='set'
      xmlns='jabber:client'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
      <resource>httpclient</resource>
    </bind>
  </iq>
</body>
```

An XMPP stream restart is required by RFC 6120 at this point; however, the server can include the stream features element (if any) along with the SASL authentication success and XMPP resource binding success notifications, as shown in the following example.

Listing 10: SASL Success with Pipelined Stream Features and XMPP Resource Binding Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 149
```

```
<body xmlns='http://jabber.org/protocol/httpbind'>
  <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    dj1wTk5ERlZFUXh1WHhDb1NFaVc4R0VaKzFSU289
  </success>
  <stream:features>
    <pipelining xmlns='urn:xmpp:features:pipelining'/>
    <c xmlns='http://jabber.org/protocol/caps'
       hash='sha-1'
       node='http://prosody.im/'
       ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
  </stream:features>
  <iq id='bind_1'
      type='result'
      xmlns='jabber:client'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
      <jid>user@example.com/httpclient</jid>
    </bind>
  </iq>
</body>
```

Without pipelining, this second exchange would require 3 round trips; with pipelining, it requires only 1.

Therefore, without pipelining the XMPP exchanges for stream establishment over BOSH require at least 5 round trips (if the SASL mechanism is not multi-stage, and perhaps more depending on the SASL mechanism used); with pipelining the minimum number of round trips is 1.

Note: It might seem that with pipelining the minimum number of round trips is 2. However, consider the case of a session creation request that includes (a) a SASL authentication request for a SASL mechanism that is not multi-stage, such as PLAIN, (b) an XMPP resource binding request, and (c) a stream restart request; in the "happy path" the session creation response would include (a) stream features, (b) a SASL authentication success notification, and (c) an XMPP resource binding response. This flow is shown in the following example.

Listing 11: BOSH Session Request with Pipelined SASL Authentication Request and XMPP Resource Binding Request, Including Stream Restart Request

```
POST /webclient HTTP/1.1
Host: httpcm.jabber.org
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 104

<body content='text/xml;␣charset=utf-8'
      from='user@example.com'
      hold='1'
      rid='1573741820'
```

```
      to='example.com'
      route='xmpp:example.com:9999'
      secure='true'
      wait='60'
      xml:lang='en'
      xmpp:restart='true'
      xmpp:version='1.0'
      xmlns='http://jabber.org/protocol/httpbind'
      xmlns:xmpp='urn:xmpp:xbosh'>
  <auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="PLAIN">
    [plain credentials here]
  </auth>
  <iq id='bind_2' type='set' xmlns='jabber:client'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
  </iq>
</body>
```

Listing 12: BOSH Session Response with Pipelined Stream Features, SASL Authentication Request, and XMPP Resource Binding Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 674

<body wait='60'
      inactivity='30'
      polling='5'
      requests='2'
      hold='1'
      from='example.com'
      accept='deflate,gzip'
      sid='SomeSID'
      secure='true'
      charsets='ISO_8859-1 ISO-2022-JP'
      xmpp:restartlogic='true'
      xmpp:version='1.0'
      authid='ServerStreamID'
      xmlns='http://jabber.org/protocol/httpbind'
      xmlns:xmpp='urn:xmpp:xbosh'
      xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:features>
    <pipelining xmlns='urn:xmpp:features:pipelining'/>
    <c xmlns='http://jabber.org/protocol/caps'
       hash='sha-1'
       node='http://prosody.im/'
       ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <mechanism>SCRAM-SHA-1</mechanism>
      <mechanism>PLAIN</mechanism>
    </mechanisms>
```

```
  </stream:features>
  <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
  <iq id='bind_2' type='result' xmlns='jabber:client'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
      <jid>user@example.com/F580F69C-6E1E-41CF-94A5-692D96D0EC51</jid>
    </bind>
  </iq>
</body>
```

7 Reconnection

The pain of multiple round trips is magnified if the initiating entity needs to reconnect frequently (e.g., because of intermittent network outages). Although XEP-0124 can be used to mitigate the pain, BOSH is not appropriate for all scenarios and is not currently used in others (e.g., server-to-server streams).

To minimize the speed of reconnection, implementations are strongly encouraged to support TLS Session Resumption (RFC 5077 [14]) in addition to the technologies already mentioned.

Reconnection can be further enhanced by using the stream resumption feature defined in Stream Management (XEP-0198) [15]. XEP-0198 does not legislate exactly when it is safe for the server to allow the client to send the <resume/> request. Clearly, sending it before the stream is encrypted would increase the possibility of replay attacks. However, sending it after TLS negotiation (Step 4 above) but before SASL authentication and resource binding (Steps 5 through 8) would enable the client to begin sending stanzas more quickly. It is a matter of server policy whether to advertise the SM feature after TLS negotiation or only after SASL negotiation.

8 Security Considerations

Because pipelining does not skip any channel encryption or authentication steps, but merely packs them into a smaller number of TCP packets or HTTP request/response pairs, it is unlikely that the foregoing quickstart methods introduce security vulnerabilities. However, the server needs to be careful not to send stream features that it would not otherwise send before a security context is established.

[14] RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State <http://tools.ietf.org/html/rfc5077>.

[15] XEP-0198: Stream Management <https://xmpp.org/extensions/xep-0198.html>.

9 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [16].

10 XMPP Registrar Considerations

This specification defines the following XML namespace:

- urn:xmpp:features:pipelining

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [17] shall add the foregoing namespace to the registry located at <https://xmpp.org/registrar/stream-features.html>, as described in Section 4 of XMPP Registrar Function (XEP-0053) [18].

11 Acknowledgements

Special thanks to Tony Finch for suggesting this work and for providing the initial outline of how pipelining would work. Thanks also to Waqas Hussain, Jehan Pagès, and Kevin Smith for their feedback.

[16] 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/>.

[17] 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/>.

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