



XMPP

XEP-0305: XMPP Quickstart

Peter Saint-Andre
<mailto:stpeter@jabber.org>
<xmpp:stpeter@jabber.org>
<https://stpeter.im/>

2011-08-25
Version 0.1

Status	Type	Short Name
Experimental	Standards Track	N/A

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

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 - 2011 by the [XMPP Standards Foundation](#) (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <http://xmpp.org/about-xmpp/xsf/xsf-ipr-policy/> or obtained by writing to XMPP Standards Foundation, 1899 Wynkoop Street, Suite 600, Denver, CO 80202 USA).

Contents

1	Introduction	1
2	Preparing to Connect	1
3	Pipelining	1
4	Reconnection	6
5	Security Considerations	6
6	IANA Considerations	6
7	XMPP Registrar Considerations	6
8	Acknowledgements	7

1 Introduction

Establishing an XMPP session requires a fairly large number of round trips between the initiating entity and the receiving entity. In many deployment scenarios, it would be helpful to reduce the number of round trips and, in general, the time needed to establish a session. This document defines protocols and best practices to do just that.

Note: Various parts of this document might be moved to separate documents at some point.

2 Preparing to Connect

In accordance with [RFC 6120](#)¹, before attempting to establish a stream 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.

XMPP applications SHOULD cache whatever information they can about the peer, especially stream features data and [Service Discovery](#)² information. To facilitate such caching, servers SHOULD include [Entity Capabilities](#)³ 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](#)⁴ as subsequently included in Section 2.1.1 of [RFC 6121](#)⁵.

3 Pipelining

One method of speeding the connection process is pipelining of requests, as in [RFC 2920](#)⁶ and the QUICKSTART extension proposed for SMTP ([The QUICKSTART SMTP service extension](#)⁷). The application of similar principles to XMPP was originally suggested by Tony Finch in February 2008 <<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

¹RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <<http://tools.ietf.org/html/rfc6120>>.

²XEP-0030: Service Discovery <<http://xmpp.org/extensions/xep-0030.html>>.

³XEP-0115: Entity Capabilities <<http://xmpp.org/extensions/xep-0115.html>>.

⁴XEP-0237: Roster Versioning <<http://xmpp.org/extensions/xep-0237.html>>.

⁵RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <<http://tools.ietf.org/html/rfc6121>>.

⁶RFC 2920: SMTP Service Extension for Command Pipelining <<http://tools.ietf.org/html/rfc2920>>.

⁷The QUICKSTART SMTP service extension <<http://tools.ietf.org/id/draft-fanf-smtp-quickstart-01.txt>>. Work in progress.

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 again 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.

If an XMPP server supports pipelining, it MUST advertise a stream feature of `<pipelining xmlns='urn:xmpp:features:pipelining'/>`.

If both parties support pipelining, they can proceed as follows (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 Step 1, 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: Step 1: Client Initiation

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

In Step 2, 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 5246⁸ for details).

Listing 2: Step 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>
```

⁸RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 <<http://tools.ietf.org/html/rfc5246>>.

```

<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

```

In Step 3, the parties complete the TLS negotiation.

Listing 3: Step 3: TLS Handshake

Client might send some combination of Certificate, ClientKeyExchange, CertificateVerify, ChangeCipherSpec, and Finished (see RFC 5246)

In Step 4, the server knows that the client will need to restart the stream so it proactively attaches its response stream header and stream features after the TLS Finished message.

Listing 4: Step 4: Server Proactively Restarts Stream

```

TLS Finished
<stream:stream
  from='im.example.com'
  id='vgKi/bkYME80Aj4rlXMkpucAqe4='
  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 Step 5, the client pipelines its initial stream header with the command for initiating the SASL authentication process (including SASL "initial response" data as explained in Section

6.3.10 of RFC 6120 to reduce the number of round trips).

Listing 5: Step 5: 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>
```

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

Listing 6: Step 6: Server Sends SASL Challenge

```
<challenge xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
  cj1vTXNUQUF3QUFBQU1BQUFBTlAwVEFBQUFBQUJQVTBBQWUxMjQ2OTViLTU5Y
  TktNGRlNi05YzMwLWI1MWIzODA4YzU5ZSxzPU5qaGtZVE0wTURndE5HWTBaaT
  AwTmPkUxUa3hNbVV0TKRsbU5UTm10RE5rTURNeixpPTQwOTY=
</challenge>
```

When the client suspects that it is sending its final SASL response, it SHOULD append an initial stream header and resource binding request.

Listing 7: Step 7: Client Sends Final SASL Response with Stream Header and Bind Request

```
<response xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
  Yz1iaXdzLHI9b01zVEFBd0FBQUFNQUFBQU5QMFRBQUFBQUFCUFUwQUF1MTI0N
  jk1Yi020WE5LTRkZTYtOWMzMC1iNTFiMzgwOGM1OWUscD1VQTU3dE0vU3ZwQV
  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>

```

In Step 8, the server informs the client of SASL success (including "additional data with success" as explained in Section 6.3.10 of RFC 6120 to reduce the number of round trips), sends a response stream header and stream features, and informs the client of successful resource binding.

Listing 8: Step 8: Server Accepts Bind Request

```

<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  dj1wTk5ER1ZFUXh1WHhDb1NFaVc4R0VaKzFSU289
</success>
<stream:stream
  from='im.example.com'
  id='gPybza0zBmaADgxKXu9UC1bprp0='
  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>

```

The XMPP stream negotiation process in RFC 6120 required at least 19 round trips (including 4 for TLS negotiation). With pipelining, the number of round trips is reduced to 8. 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.

4 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 [BOSH](#)⁹ 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](#)¹⁰) in addition to the technologies already mentioned.

Reconnection can be further enhanced by using the stream resumption feature described in [Stream Management](#)¹¹. 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.

5 Security Considerations

To follow.

6 IANA Considerations

This document requires no interaction with the [Internet Assigned Numbers Authority \(IANA\)](#)¹².

7 XMPP Registrar Considerations

This specification defines the following XML namespace:

- `urn:xmpp:features:pipelining`

⁹XEP-0124: Bidirectional-streams Over Synchronous HTTP <<http://xmpp.org/extensions/xep-0124.html>>.

¹⁰RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State <<http://tools.ietf.org/html/rfc5077>>.

¹¹XEP-0198: Stream Management <<http://xmpp.org/extensions/xep-0198.html>>.

¹²The 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/>>.

Upon advancement of this specification from a status of Experimental to a status of Draft, the [XMPP Registrar](#)¹³ shall add the foregoing namespace to the registry located at [<http://xmpp.org/registrar/stream-features.html>](http://xmpp.org/registrar/stream-features.html), as described in Section 4 of [XMPP Registrar Function](#)¹⁴.

8 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.

¹³The 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 <http://xmpp.org/registrar/>.

¹⁴XEP-0053: XMPP Registrar Function [<http://xmpp.org/extensions/xep-0053.html>](http://xmpp.org/extensions/xep-0053.html).