Extensible Messaging and Presence Protocol (XMPP)

Protocol Introduction and Overview

Sam Whited

XSF Editor / Council

JID: sam@samwhited.com

2017–11–01

CLOUDFLARE

XMPP

The following requirements keywords as used in this document are to be interpreted as described in RFC 2119: "MUST", "SHALL", "REQUIRED"; "MUST NOT", "SHALL NOT"; "SHOULD", "RECOMMENDED"; "SHOULD NOT", "NOT RECOMMENDED"; "MAY", "OPTIONAL".

XMPP is a network protocol for exchanging data between two entities in near-real-time.

Standards

XMPP standardization managed by the IETF. Responsibility for extensions delegated to the XMPP Standards Foundation.

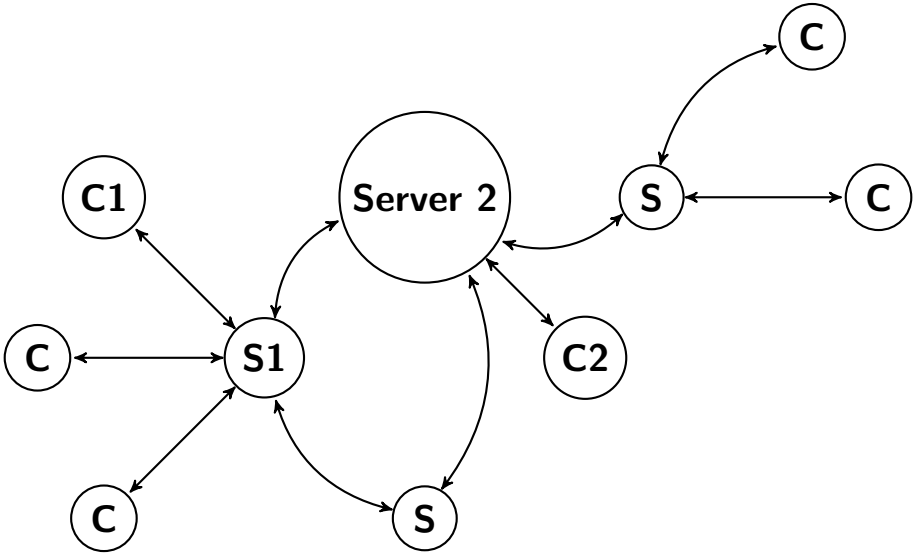
- IETF
  - RFC 6120: XMPP Core
  - RFC 6121: XMPP IM
  - RFC 7590: Use of TLS in XMPP
  - RFC 7622: XMPP Address Format
- XSF
  - XEP-0045: Multi-User Chat
  - XEP-0198: Stream Management
  - XEP-0367: Message Attaching
  - XEP-0387: XMPP Compliance Suites 2017
  - ...

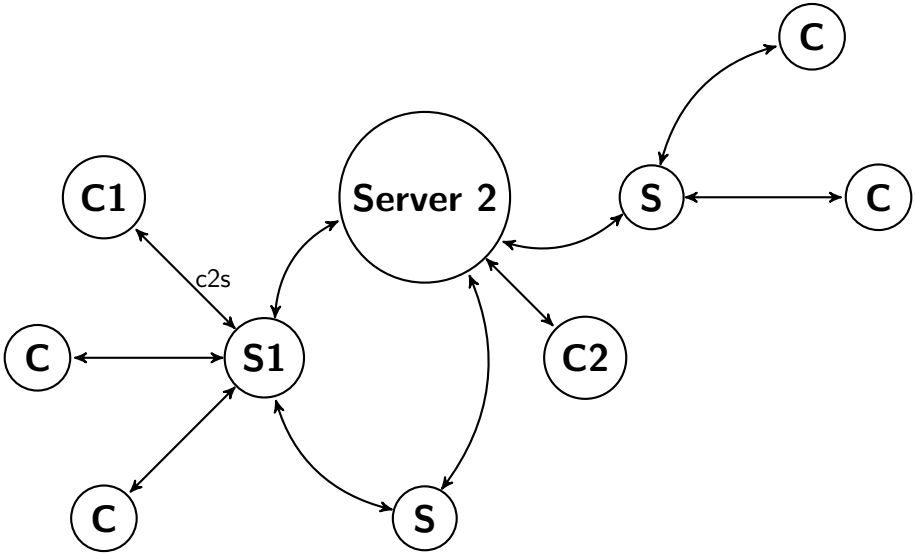
1999	XMPP created by the Jabber open-source community.
2002	IETF forms the XMPP WG.
2008	Cisco aquires Jabber, Inc.
2004	RFC 3920, RFC 3921, RFC 3922, and RFC 3923 approved.
2011	RFCs superseded by RFC 6120, RFC 6121, and RFC 6122.
2014	Websocket subprotocol RFC 7395 created.
2015	Address format superseded by RFC 7622, TLS updates in RFC 7590.

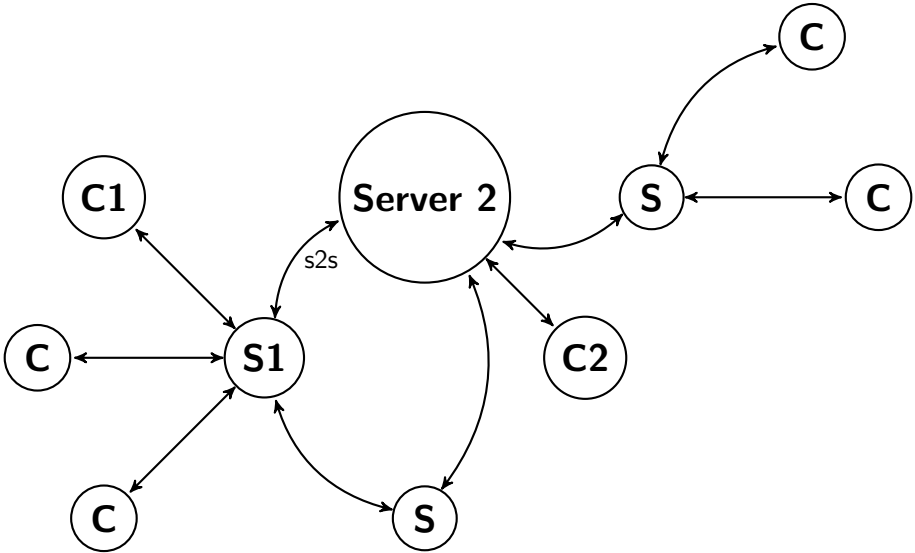
eXtensible Messaging and Presence Protocol

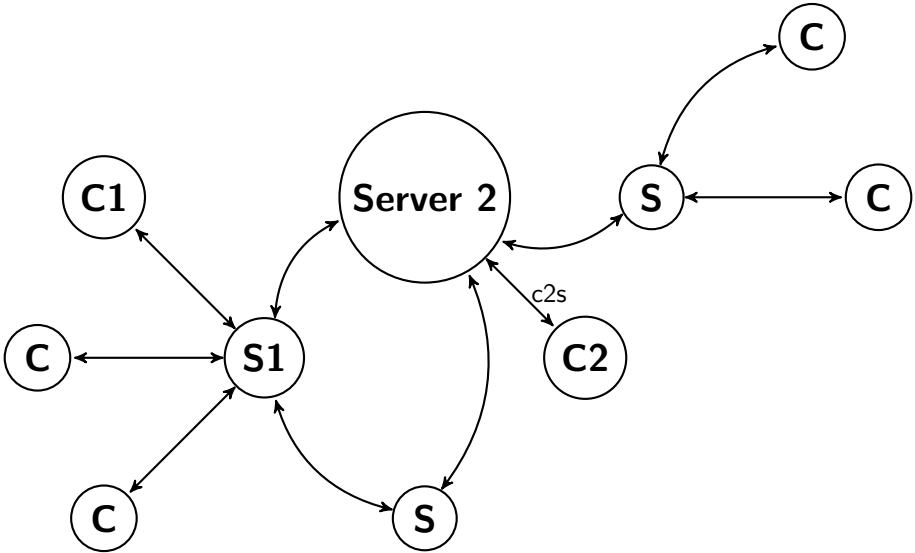
(from 10,000 feet)

- XML streams (not documents)
- Elements with payloads
  - Stanzas: *message*, *presence*, *iq*
  - Other: *auth*, *compress*, ...
- Minimal core spec with extensions defined in XEP's
- Federated network of servers


C
C1
Server 2
S
C
C
S1
C2
C
S


C
C1
Server 2
S
C
c2s
C
S1
C2
C
S

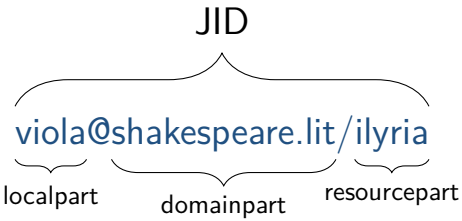

C
C1
Server 2
S
C
s2s
C
S1
C2
C
S


C
C1
Server 2
S
C
c2s
C
S1
C2
C
S

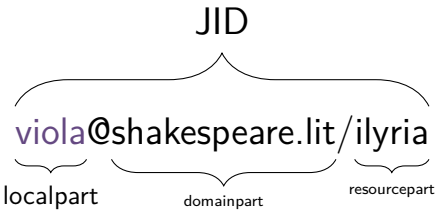
XMPP Address Format

RFC 7622

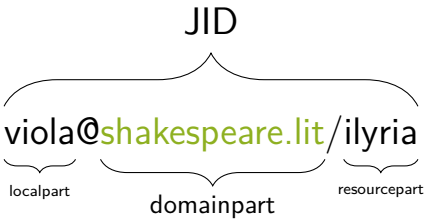
Anatomy of a JID



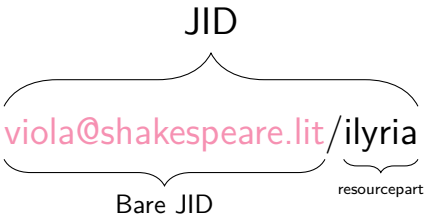
Anatomy of a JID



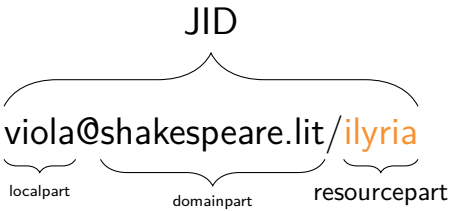
Anatomy of a JID



Anatomy of a JID



Anatomy of a JID



XMPP Core

RFC 6120

Stream's

- Client first
- Two streams, input and output, over one TCP socket
- As a security measure, streams are restarted when their state changes (eg. TLS or stream compression)
- Event based and pipelined (async communication)

Stream Initialization and Feature Negotiation

Client

Server

Client Server

```
<?xml version='1.0'?>
<stream:stream …>
```

Client

```
<?xml version='1.0'?>
<stream:stream …>
```

Server

```
<?xml version='1.0'?>
<stream:stream …>
```

Client

```
<?xml version='1.0'?>
<stream:stream …>
```

Server

```
<?xml version='1.0'?>
<stream:stream …>
<stream:features>
<starttls …>
<required />
</starttls>
</stream:features>
```

Client	Server
`<?xml version='1.0'?>`	
`<stream:stream …>`	
	`<?xml version='1.0'?>`
	`<stream:stream …>`
	`<stream:features>`
	`<starttls …>`
	`<required />`
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	

Client	Server
`<?xml version='1.0'?>`	
`<stream:stream …>`	
	`<?xml version='1.0'?>`
	`<stream:stream …>`
	`<stream:features>`
	`<starttls …>`
	`<required />`
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	
	`<proceed …/>`

Client	Server
`<?xml version='1.0'?>`	
`<stream:stream …>`	
	`<?xml version='1.0'?>`
	`<stream:stream …>`
	`<stream:features>`
	`<starttls …>`
	`<required />`
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	
	`<proceed …/>`

0100011101100101011101000010000001100010011000010110001101101011001000000111010001101111001000000111011101101111011100100110101100101110

Client	Server
`<?xml version='1.0'?>`	
`<stream:stream …>`	
	`<?xml version='1.0'?>`
	`<stream:stream …>`
	`<stream:features>`
	`<starttls …>`
	`<required />`
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	
	`<proceed …/>`
0100011101100101011101000010000001100010011000010110001101101011001	
0000011101000110111100100000011101110110111011100100110101100101110	
`<stream:stream …>`	

Client	Server
`<?xml version='1.0'?>`	
`<stream:stream …>`	
	`<?xml version='1.0'?>`
	`<stream:stream …>`
	`<stream:features>`
	`<starttls …>`
	`<required />`
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	
	`<proceed …/>`
0100011101100101011101000010000001100010011000010110001101101011001	
0000011101000110111100100000011101110110111011100100110101100101110	
`<stream:stream …>`	
	`<stream:stream …>`

Client	Server
	`<starttls …>`
	`<required />`
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	
	`<proceed …/>`
0100011101100101011101000010000001100010011000010110001101101011001	
0000011101000110111100100000011101110110111011100100110101100101110	
`<stream:stream …>`	
	`<stream:stream …>`
	`<stream:features …>`
	`<mechanisms …>`
	`<mechanism>SCRAM-SHA-1</mechanism>`
	`</mechanisms>`

Client	Server
	<required />
	</starttls>
	</stream:features>
<starttls …/>	
	<proceed …/>
0100011101100101011101000010000001100010011000010110001101101011001	
0000011101000110111100100000011101110110111011100100110101100101110	
<stream:stream …>	
	<stream:stream …>
	<stream:features …>
	<mechanisms …>
	<mechanism>SCRAM-SHA-1</mechanism>
	</mechanisms>
<auth>	

Client	Server
	`</starttls>`
	`</stream:features>`
`<starttls …/>`	
	`<proceed …/>`
0100011101100101011101000010000001100010011000010110001101101011001 0	
0000011101000110111100100000011101110110111011100100110101100101110	
`<stream:stream …>`	
	`<stream:stream …>`
	`<stream:features …>`
	`<mechanisms …>`
	`<mechanism>SCRAM-SHA-1</mechanism>`
	`</mechanisms>`
`<auth>`	
…	

Client	Server
	`</stream:features>`
`<starttls …/>`	`<proceed …/>`
0100011101100101011101000010000001100010011000010110001101101011001 0	
0000011101000110111100100000011101110110111011100100110101100101110	
`<stream:stream …>`	
	`<stream:stream …>`
	`<stream:features …>`
	`<mechanisms …>`
	`<mechanism>SCRAM-SHA-1</mechanism>`
	`</mechanisms>`
`<auth>`	
…	
…	

Client	Server
`<starttls …/>`	`<proceed …/>`
0100011101100101011101000010000001100010011000010110001101101011001 0	
0000011101000110111100100000011101110110111011100100110101100101110	
`<stream:stream …>`	
	`<stream:stream …>`
	`<stream:features …>`
	`<mechanisms …>`
	`<mechanism>SCRAM-SHA-1</mechanism>`
	`</mechanisms>`
`<auth>`	
…	
…	
…	

Inside the stream

Stanza /ˈstænzə/ (*plural* stanzas) n.

❶ A unit of a poem, written or printed as a paragraph; equivalent to a verse.

❷ (computing) An XML element which acts as basic unit of meaning in XMPP.

Stanza's

The basic primitives of XMPP.

- `<message/>`
- `<iq/>`
- `<presence/>`

These are the only routable elements in an XMPP stream.

<message/>

- One-to-one
- Fire and forget
- No ack
- Useful for anything that does not require a response (chats, alerts, logging, etc.)

```
<message id='262' type='chat'
         to='feste@shakespeare.lit/house'>
  <body>
    What&apos;s a drunken man like, fool?
  </body>
  <request xmlns='urn:xmpp:receipts'/>
  <thread>pNltztLMBQhqakHwcFd</thread>
</message>
```

<iq/> (“Information query”)

- One-to-one
- Acked
- Optional at-least-once delivery

```
<iq from='capulet.lit'
    to='juliet@capulet.lit/balcony'
    id='s2c1' type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>

<iq to='capulet.lit'
    from='juliet@capulet.lit/balcony'
    id='s2c1' type='result'/>
```

<presence/>

- Directed (one-to-one) or broadcast (one-to-many)
- Advertises entity availability to the network
- Payload's for broadcast can ride along (entity capabilities, status messages, etc.)

```
<presence id='aeg8y7pd'
          from='prospero@shakespeare.lit/cell'>
  <status>
    Now my charms are all o&apos;erthrown
  </status>
  <priority>40</priority>
  <show>away</show>
</presence>
```

Namespacing

Stanza payloads are handled based on their XML namespace. By recent convention, namespaces are versioned URN's.

```
<message from='juliet@capulet.lit'
         to='romeo@montague.lit/orchard'
         type='headline' id='tfasd'>
  <result xmlns='urn:xmpp:mam:1' queryid='f27'
          id='5d398-28273-f7382'>
  ...
  </result>
</message>
```

"XMPP is Sacred"

—XEP-0134: XMPP Design Guidelines

When designing a new extension, think very hard about your life before you invent new stream level elements, and never modify core protocol.

Useful Extensions

"Extensions are XMPP's greatest strength, and its greatest weakness."

— Pretty much everyone

XEP-0280: Message Carbons

- Copies incoming messages to resources that would otherwise not have received the message.
- Copies outgoing messages to your other connected resources
- Current behavior not well defined for special messages (Typing notifications, read state markers, etc.)
- It's simple and gets the job done
- One day might be replaced by...

XEP-0313: Message Archive Management (MAM)

- Stores incoming and outgoing messages on server
- New clients can access history
- Clients that have been offline can catch up
- Currently relies on synchronized clocks between client and server (facepalm)
- Work is being done to make it possible to query the archive for messages after a given ID.

Myth: XMPP is bad on mobile

Turns out that XMPP is actually *very* good on mobile devices, both on battery and bandwidth.[1] Historically, mobile *clients* have been very bad.

[1] Isode has deployed XMPP over 9600 bit/s SATCOM and STANAG 5066 HF radio

XEP-0352: Client State Indication (CSI)

Clients indicate when they become “inactive” (screen goes off, app loses focus, etc.) or “active” with some simple nonza’s. Server does what it wants with that data (eg. don’t send presence or typing notifications and start sending push notifications).

```
<active xmlns='urn:xmpp:csi:0'/>
<inactive xmlns='urn:xmpp:csi:0'/>
```

XEP-0268: Mobile Considerations

Attempts to tell you everything you need to know about not eating your users' battery.

TL;DR — Implement CSI, and when you detect that something is already being sent/received: Send/receive as much data as you can at once so the radio can go back to sleep. Compression is also good.

Disclaimer: I wrote this one and it's still in progress; I tried to do my research, but your mileage may vary.

XEP-0198: Stream Management

- Stream resumption (very fast reconnects)
- Stanza acknowledgements

Has some problems around “zombie state’s” where clients are offline, but the stream hasn’t timed out yet. Like most things, the answer is probably MAM.

Services

- jabber.at
- Conversations.im
- Hipchat Data Center
- Cisco Jabber
- Google Cloud Print (GCP)
- Firebase Cloud Messaging (alerting)
- Jitsi Meet (video conferencing)
- Nintendo Switch notifications
- Welcome
- Zimbra Talk

Servers

- Prosody (Lua)
- Ejabberd (Erlang)
- MongooseIM (Erlang)
- M-Link (C++)
- Openfire (Java)
- Tigase (Java)

IM Clients

- Conversations (Android)
- ChatSecure (iOS)
- Dino (Linux, OSX, Windows)
- Swift (Linux, OSX, Windows)
- Gajim (Linux, Windows)
- Apple Messages (OSX)

Libraries

- JVM (Java, Clojure, Scala, etc.)
  - Smack
  - Babbler
- Python
  - aioxmpp
  - Words (Twisted)
- Lua: Verse
- Go: `mellium.im/xmpp`
- Rust: xmpp-rs
- JavaScript: Stanza.io

email & jid

sam@SamWhited.com

me

twitter & bitbucket

blog