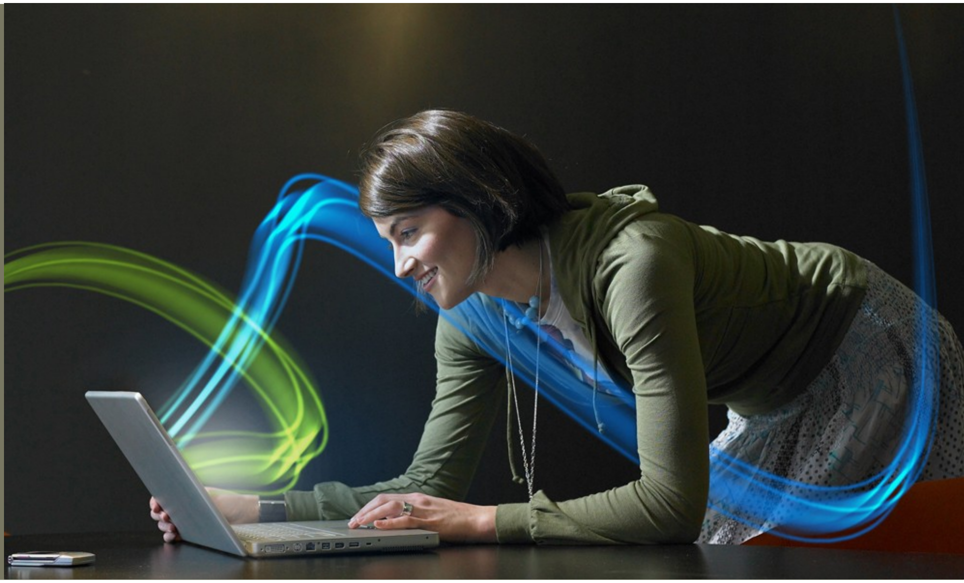
Introduction to XMPP



What is XMPP?

- eXtensible Messaging and Presence Protocol
- Bi-directional streaming XML
- Core: IETF RFC 6120, 7590, 6121
- Extensions: XMPP Standards Foundation (XSF)
  - Membership-based
  - Elected technical council
  - Unit of work: XMPP Extension Protocol (XEP)
  - Process: Experimental, Proposed, Draft, Final
- Goals:
  - Simple clients
  - Federate everything

Related RFCs

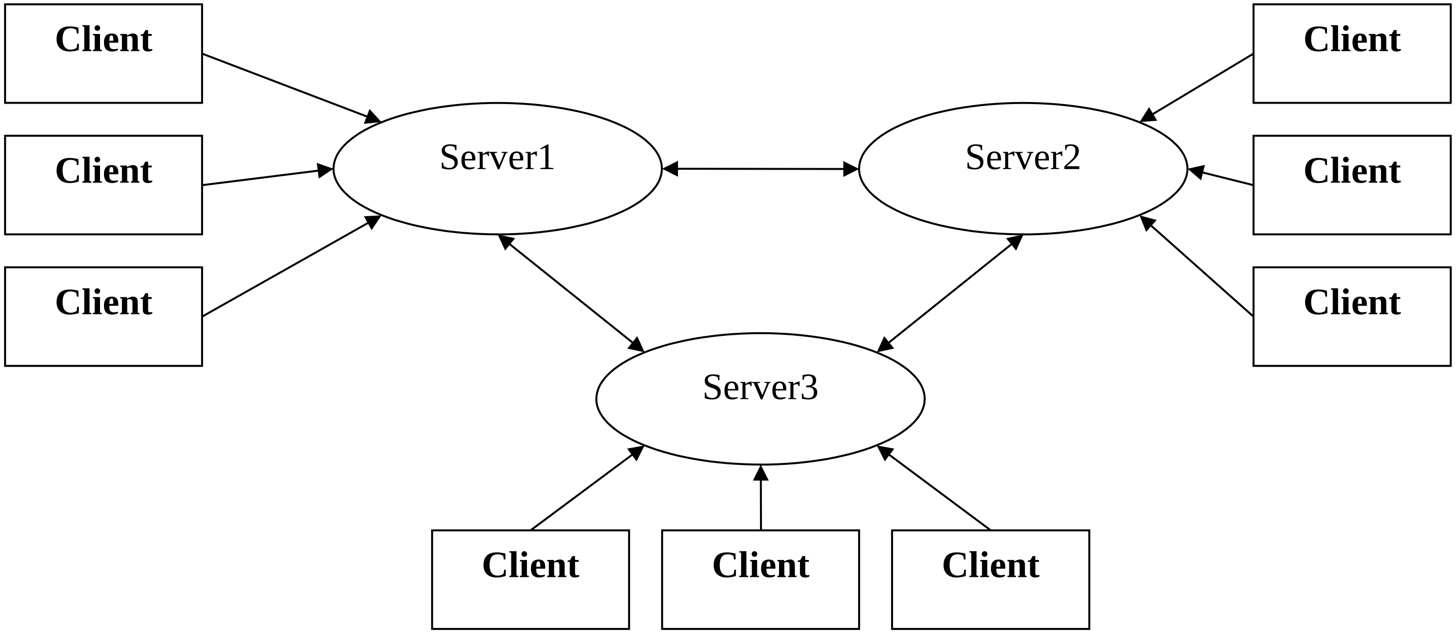
RFC	Name	Description
RFC 6120	XMPP Core	XMPP core features Updated by 7590 (TLS)
RFC 6121	XMPP IM	XMPP Instant Messaging and Presence
RFC 3922	XMPP CPIM	Mapping XMPP to Common Presence and Instant Messaging (CPIM)
RFC 3923	XMPP E2E	End-to-End Signing and Object Encryption for XMPP
RFC 5122	XMPP URI	Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for XMPP
RFC 4854	XMPP URN	Uniform Resource Name (URN) Namespace for XMPP

What is XMPP ?

- The eXtensible Messaging and Presence Protocol (XMPP) is a TCP communications protocol based on XML that enables near-real-time exchange of structured data between two or more connected entities.
- Out-of-the-box features of XMPP include presence information and contact list maintenance.
- Due in part to its open nature and XML foundation, XMPP has been extended for use in publish-subscribe systems
  - Perfect for IoT applications.

https://www.infoworld.com/article/2972143/internet-of-things/real-time-protocols-for-iot-apps.html

XMPP Architecture

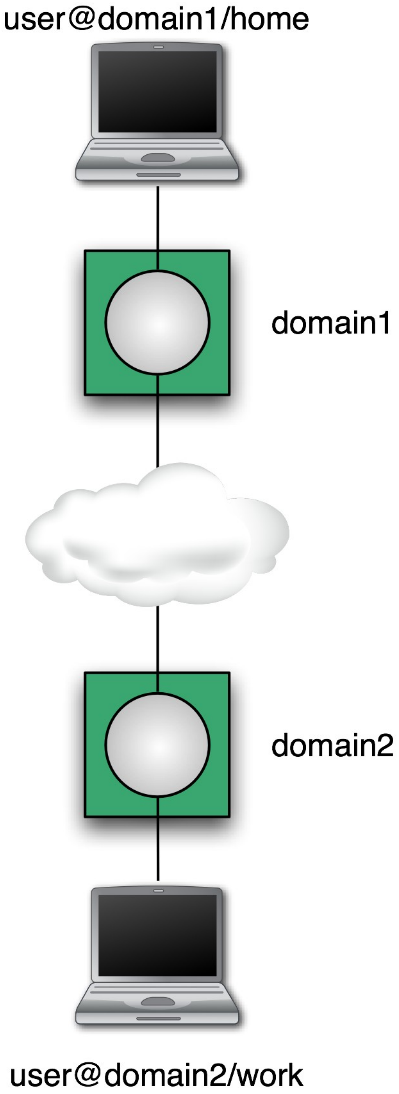


Server<->Server: Port 5269

Client<->Server: Port 5222

XMPP Architecture

- Addressing Scheme: `node@domain/resource`
  - JID = Jabber ID
  - Node: identity, e.g. user name
  - Domain: DNS domain name
  - Resource: device identifier
  - `node@domain` identifies a person
- Client talks to “local” server
  - Wherever the user account is hosted
  - Tied to directory if desired
  - Organizational policy enforced
- Servers talk to other servers
  - DNS lookup on domain portion of address
  - Dialback, MTLS for security
  - One connection for many conversations



XML Refresher

- Element
- Attribute
- Namespace
- Language
- Text

Attribute

```
<geoloc xmlns='http://jabber.org/protocol/geoloc'

        xml:lang='en'
        id='14'>
  <lat>38.9</lat>
  <lon>-77.1</lon>
  <locality>Arlington</locality>
  <region>VA</region>
</geoloc>
```

Element

XMPP Streams

- Client connects TCP socket to server
- Client sends stream start tag:

```
<stream:stream xmlns='jabber:client'
               xmlns:stream='http://etherx.jabber.org/streams'
               to='example.com'
               version='1.0'>
```

- Server sends stream start tag back:

```
<stream:stream xmlns='jabber:client'
               xmlns:stream='http://etherx.jabber.org/streams'
               from='example.com’
               id='someid'
               version='1.0'>
```

- Each child element of stream: a “*stanza*”

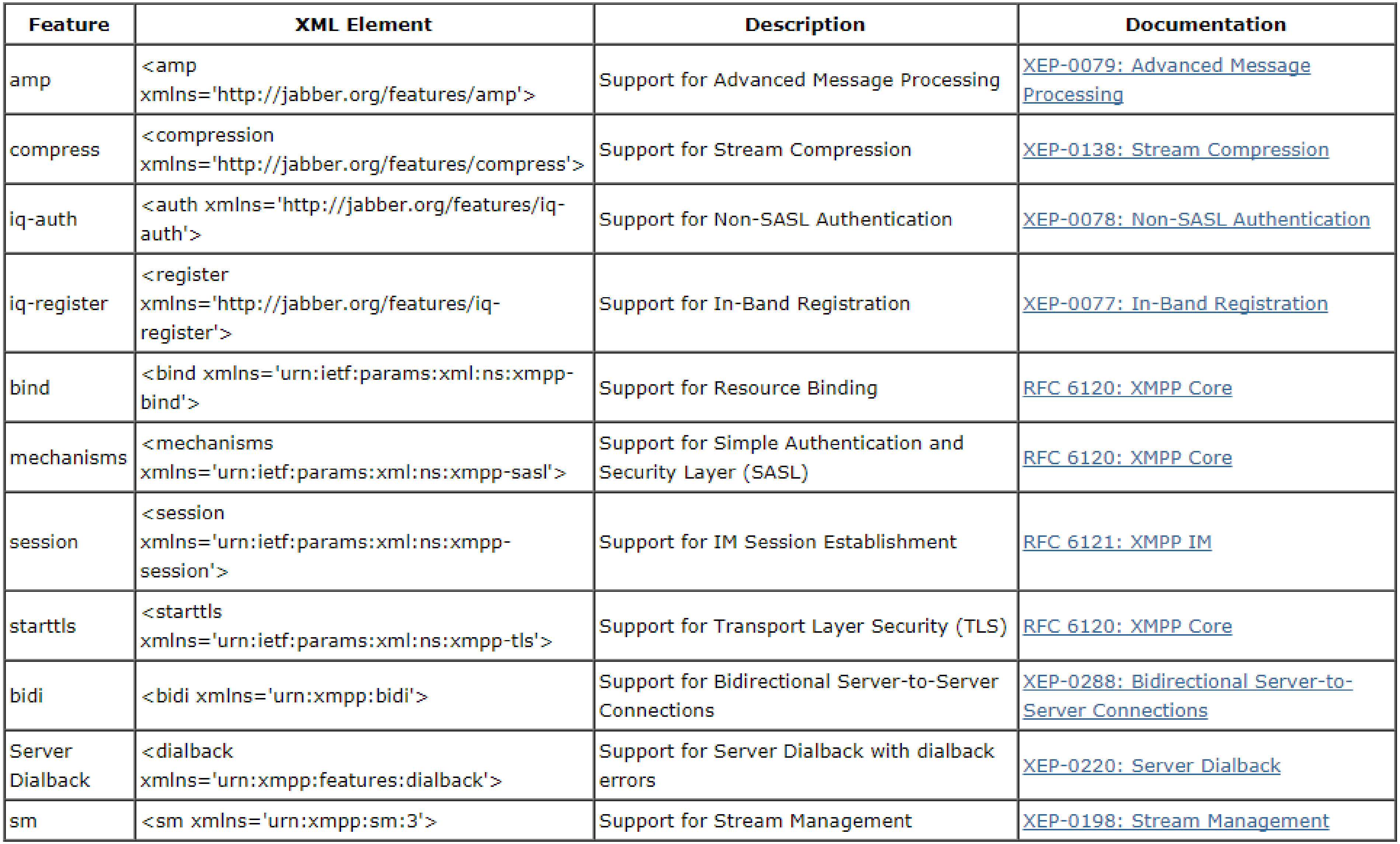
Stream features

- After stream start, server sends feature list:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
  </mechanisms>
  <compression xmlns='http://jabber.org/features/compress'>
    <method>zlib</method>
  </compression>
</stream:features>
```

- Client can negotiate any of these features

XML Stream Features



Feature	XML Element	Description	Documentation
amp	<amp xmlns='http://jabber.org/features/amp'>	Support for Advanced Message Processing	XEP-0079: Advanced Message Processing
compress	<compression xmlns='http://jabber.org/features/compress'>	Support for Stream Compression	XEP-0138: Stream Compression
iq-auth	<auth xmlns='http://jabber.org/features/iq-auth'>	Support for Non-SASL Authentication	XEP-0078: Non-SASL Authentication
iq-register	<register xmlns='http://jabber.org/features/iq-register'>	Support for In-Band Registration	XEP-0077: In-Band Registration
bind	<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>	Support for Resource Binding	RFC 6120: XMPP Core
mechanisms	<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>	Support for Simple Authentication and Security Layer (SASL)	RFC 6120: XMPP Core
session	<session xmlns='urn:ietf:params:xml:ns:xmpp-session'>	Support for IM Session Establishment	RFC 6121: XMPP IM
starttls	<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>	Support for Transport Layer Security (TLS)	RFC 6120: XMPP Core
bidi	<bidi xmlns='urn:xmpp:bidi'>	Support for Bidirectional Server-to-Server Connections	XEP-0288: Bidirectional Server-to-Server Connections
Server Dialback	<dialback xmlns='urn:xmpp:features:dialback'>	Support for Server Dialback with dialback errors	XEP-0220: Server Dialback
sm	<sm xmlns='urn:xmpp:sm:3'>	Support for Stream Management	XEP-0198: Stream Management

Security Stuff

- Start-TLS
  - Prove the identity of the server
  - Prove the identity of the user (optional)
  - Encryption
  - Data integrity
- SASL(Simple Authentication and Security Layer protocol) (RFC 4422)
  - Authentication
  - Optional encryption (rarely used)
  - Pluggable (e.g. passwords, Kerberos, X.509, SAML, etc.)

Stanzas

- All have `to='JID'` and `from='JID'` addresses
  - To gives destination
  - From added by local server
- Each stanza routed separately
- All contents of stanza passed along
- Extend with any XML from your namespace
- Different types for delivery semantics

  `<message/>`: one direction, one recipient

  `<presence/>`: one direction, publish to many

  `<iq/>`: "info/query", request/response

See details
next page

Message

- Example:

```
<message xml:lang='en'
         to='romeo@example.net'
         from='juliet@example.com/balcony'
         type='chat'>
  <body>Wherefore art thou, Romeo?</body>
</message>
```

- Types: chat, groupchat, headline, error
- Body: plain text
- XHTML IM: XEP-0071

Presence express an entity's current network availability

- Example:

```
<presence>
  <show>dnd</show>
  <status>Meeting</status>
  <priority>1</priority>
</presence>
```

- Show: chat, available, away, xa, dnd
- Status: Human-readable text
- Priority: Which resource "most available"?

IQ Request

a structured request-response mechanism

- Example:

```
<iq type='get'
    id='roster_1'>
  <query xmlns='jabber:iq:roster'/>
</iq>
```

- Type: get, set, result, error
- ID: track the corresponding response
- Query/Namespace: what type of request?

IQ Response (Roster)

- Example:

```
<iq type='result'
    id='roster_1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
          name='Romeo'
          subscription='both'>
      <group>Friends</group>
    </item>
  </query>
</iq>
```

- Type: response
- ID matches request
- Subscription state: none, to, from, both

Subscribing to Presence

- Send a subscription request:

```
<presence to='juliet@example.com'
          type='subscribe'/>
```

- Approving a request:

```
<presence to='romeo@example.net'
          type='subscribed'/>
```

- Every time you change a subscription, you get a "roster push":

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
          subscription='from'/>
  </query>
</iq>
```

Extensibility Example: Message

- Use a new namespace
- Key: if you don't understand it, ignore it
- Example, CAP, XEP-0127:

```
<message to='weatherbot@jabber.org'
         from='KSTO@NWS.NOAA.GOV'>
  <alert xmlns='http://www.incident.com/cap/1.0'>
    <identifier>KSTO1055887203</identifier>
    <sent>2003-06-17T14:57:00-07:00</sent>
    <info>
      <category>Met</category>
      <event>SEVERE THUNDERSTORM</event>
...
    </info>
  </alert>
</message>
```

Common Alerting Protocol (CAP) Over XMPP

Extensibility Example: Presence

- Keep presence stanzas small
- Example: Entity Capabilities, XEP-0115:

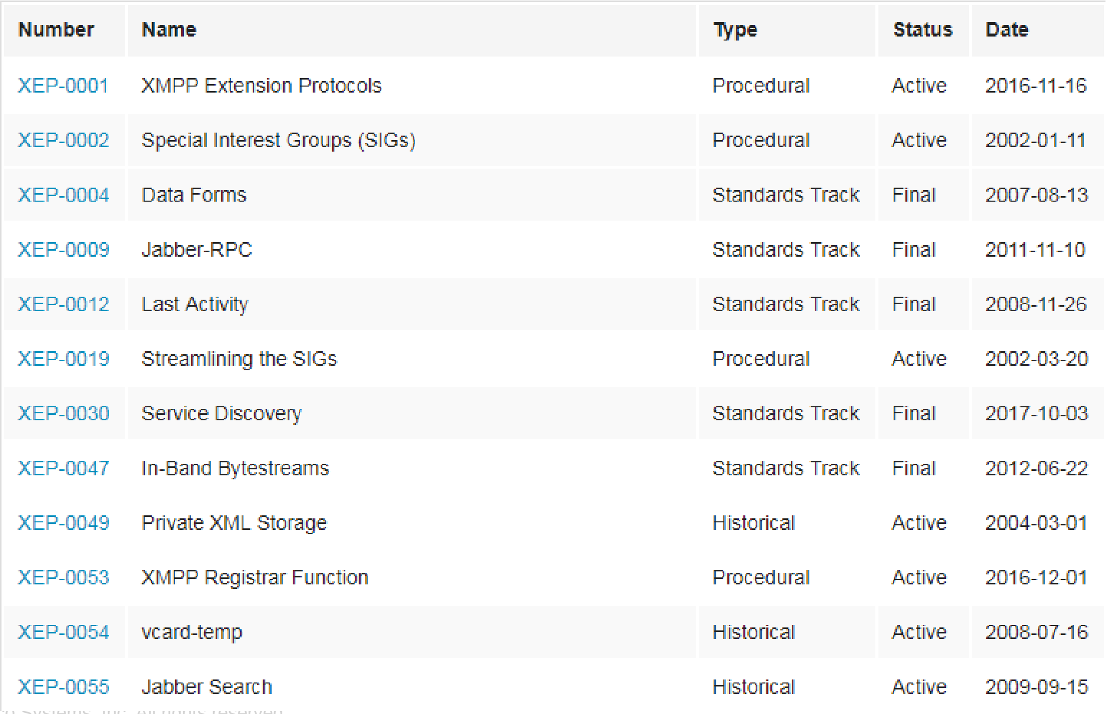
```
<presence from='bard@shakespeare.lit/globe'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://www.chatopus.com'
     ver='zHyEOgxTrkpSdGcQKH8EFPLsriY='/>
</presence>
```

- Ver attribute is hash of all features of this client
- Hash -> Feature list is cached

It defines an XMPP protocol extension for broadcasting and dynamically discovering client, device, or generic entity capabilities.

XMPP Extensions

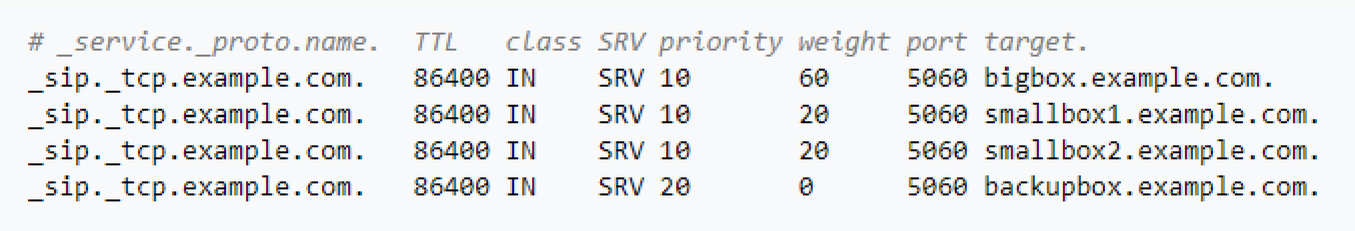
- Many already exist: http://www.xmpp.org/extensions/
- Add new ones
  - Custom: use a namespace you control, make up protocol
  - Standardized: write a XEP.



Number	Name	Type	Status	Date
XEP-0001	XMPP Extension Protocols	Procedural	Active	2016-11-16
XEP-0002	Special Interest Groups (SIGs)	Procedural	Active	2002-01-11
XEP-0004	Data Forms	Standards Track	Final	2007-08-13
XEP-0009	Jabber-RPC	Standards Track	Final	2011-11-10
XEP-0012	Last Activity	Standards Track	Final	2008-11-26
XEP-0019	Streamlining the SIGs	Procedural	Active	2002-03-20
XEP-0030	Service Discovery	Standards Track	Final	2017-10-03
XEP-0047	In-Band Bytestreams	Standards Track	Final	2012-06-22
XEP-0049	Private XML Storage	Historical	Active	2004-03-01
XEP-0053	XMPP Registrar Function	Procedural	Active	2016-12-01
XEP-0054	vcard-temp	Historical	Active	2008-07-16
XEP-0055	Jabber Search	Historical	Active	2009-09-15

Federation: DNS

- Starts with: non-local domain in to address



```
# _service._proto.name.  TTL   class SRV priority weight port target.
_sip._tcp.example.com.   86400 IN    SRV 10       60     5060 bigbox.example.com.
_sip._tcp.example.com.   86400 IN    SRV 10       20     5060 smallbox1.example.com.
_sip._tcp.example.com.   86400 IN    SRV 10       20     5060 smallbox2.example.com.
_sip._tcp.example.com.   86400 IN    SRV 20       0      5060 backupbox.example.com.
```

- Look up this DNS SRV record: (service record)
  `_xmpp-server._tcp.`*`domain`*
- Example: jabber.com:
  `10 0 5269 jabber.com.`
  - Priority: Which one to try first if multiple
  - Weight: Within a priority, what percentage chance?
  - Port: TCP port number
  - Target: Machine to connect to

Federation: Security

- Old-style: dialback
  - Connect back to domain claimed by initiator
  - Check secret claimed by initiator
  - "Someone said they were example.com; was that you?"
- New-style: Mutual TLS
  - Initiator presents "client" certificate
  - Responder presents "server" certificate
  - Both certificates signed by trusted CA
- All stanzas *must* have from with correct domain

Bandwidth minimization

- TLS compression
  - Not implemented in all SSL/TLS stacks
  - Some want compression w/o encryption
- XEP-0138: Stream Compression
  - Defines zlib mechanism (2-3x or more compression)
  - Others can be added
  - Concern: battery drain vs. radio transmission
- XEP-0198: Stanza Acknowledgements
  - Quick reconnects
  - Avoid re-synchronizing state on startup
- Partial rosters
- Privacy lists
- Others being pursued

Latency

- Most critical on startup
  - Several handshakes and stream restarts
  - Can be minimized by client assuming server configuration
  - Example: don't wait for `<stream:features>`
- Once running
  - Stanza size matters: try to stay under 8kB,
    take larger blocks out of band if possible
  - Configure federation to keep links open,
    first stanza will be slow
  - Beware of DoS protection, "karma"

Reading List

- RFCs
  - 6120: Core
  - 6121: IM & Presence
  - 5122: XMPP URIs

- XEP highlights
  - 4: Forms
  - 30: Disco
  - 45: Chat rooms
  - 60: Pub/Sub
  - 71: XHTML
  - 115: Capabilities
  - 163: PEP

Advantages of XMPP

- The primary advantage is XMPP's decentralized nature.
- XMPP works similar to email, operating across a distributed network of transfer agents rather than relying on a single, central server or broker (as CoAP and MQTT do).
- As with email, it’s easy for anyone to run their own XMPP server, allowing device manufacturers and API operators to create and manage their own network of devices.
- And because anyone can run their own server, if security is required, that server could be isolated on a company intranet behind secure authentication protocols using built-in TLS encryption.

Disadvantages of XMPP

- One of the largest flaws is the lack of end-to-end encryption. While there are many use cases in which encryption may not yet be necessary, most IoT devices will ultimately need it. The lack of end-to-end encryption is a major downside for IoT manufacturers.

  https://wiki.xmpp.org/web/XMPP_E2E_Security

- Another downside is the lack of Quality of Service (QoS). Making sure that messages are delivered is even more important in the IoT world than it was in the instant messaging world. If your alarm system doesn’t receive the message to turn itself on, then that vacation you’ve been planning could easily be ruined.