XMPP/Jabber

The Extensible Messaging and Presence Protocol

Stefan Andonie and Niklaus Hofer

14. Juni 2015

Table of Contents I

Table of Contents II

Section 1

Geschichte

Subsection 1

Frühe Entwicklung

Das Jabber Projekt

- Jeremie Miller wollte eine offene und verteilte Alternative zu Instant Messengern wie Yahoo! Messenger, MSN, ICQ, ...
- Er begann mit der Entwicklung von jabberd, den er 1999 veröffentlichte.
- Zu Beginn befasste sich die Open Source Community des Jabber Projektes vorallem mit der Entwicklung des offenen jabberd Servers.

Das Jabber Protokoll

- Bald wurde aber klar, dass das wichtigere Produkt der Community das Protokoll war das sie schuf.
- Jabber funktioniert, ähnlich wie Email, Domänenübergreifend.
- Das Jabber Protokoll erlaubt es, unterschiedliche Implementationen von Server und Client zu schreiben die untereinander kompatibel sind.

Jabber, Inc.

- In den frühen Tagen von Jabber gründeten einige der Core Entwickler Jabber, Inc. Eine Firma die Produkte auf Basis von Jabber entwickelt.
- Jabber, Inc. hat Jabber XCP, eine auf Jabber/XMPP basierende Kommunikationsplatform entwickelt, die etwa von der US Regierung eingesetzt wird.
- Lange Zeit hat Jabber, Inc. die Standardisierung von XMPP vorangetrieben und Entwickler bezahlt um vollzeit am Standard und mit der Community zu arbeiten.
- 2008 hat Cisco Jabber, Inc. gekauft.

Subsection 2

Standartisierung durch die IETF

Entwicklung der RFCs

- 2002 gründete die IETF die XMPP working group, die das Protokoll in einer Reihe von RFCs formalisierte.
  insbesondere:
  - RFC 3920 [8]
  - RFC 3921 [9]
- diese RFCs wurden 2004 zu “proposed standards“.
- später wurden die obigen RFCs abgelöst durch:
  - RFC 6120 [11]
  - RFC 6121 [12]

Benennung des Protokolls

- Unter der IETF wurde das Protokoll zu XMPP - eXtensible Messaging and Presence Protocol - umbenannt.
- Dies wurde insbesondere als notwendig erachtet, um Verwechslungen mit der Firma Jabber, Inc. zu vermeiden.
- Und weil jedes gute Netzwerkprotokoll ein vier-buchstabiges Akronym das auf “P“ endet haben muss ;)

Erweiterbarkeit

- RFC 6120 spezifiziert das Protokoll das die Kommunikation ermöglicht.
- RFC 6121 hingegen spezifiziert die Nuter zu Nutzer Kommunikation (Messaging and Presence).
- Die Extensions werden XEP - XMPP Extension Protocols - genannt.
  - `https://xmpp.org/xmpp-protocols/xmpp-extensions/`
- Die Entwicklung des Protokolles und der Extensions wird heute von der XMPP Standards Foundation (XSF) geführt.

Section 2

Core protocol

Begriffe

- XML Stanza werden die Informationselemente genannt die in XMPP ausgetauscht werden.
- Es gibt drei Typen von XML Stanzas:
  - Message
  - Presence
  - IQ (Info/Query)
- Die JID identifiziert Nutzer und Clients
  - Die bare JID beschreibt einen Nutzer. Sie ist von der Form “username@example.com“
  - Die full JID beschreibt den Nutzer und die Resource (in der Regel einen bestimmten Client). Sie ist von der Form “username@example.com/resource“

Subsection 1

XMPP Verbindungen

Aufbau einer Verbindung I

- XMPP Verbindungen werden über TCP aufgebaut.
- Ein Client verbindet sich mit einem Server über eine langlebige TCP Verbindunge. Diese Verbindung bleibt so lange offen, wie der darin geöffnete XML Stream offen bleibt.
- Besteht erstmal eine TCP Verbindung, so wird darin ein XML Stream erstellt.
  - Dazu wird ein “<stream>“ Tag mit den entsprechende Header- und Namespace Informationen übermittelt.
  - Zum Beenden des Streams wird das Tag wieder geschlossen.
- Vorzugsweise wird eine TLS gesicherte Verbindung ausgehandelt.

Aufbau einer Verbindung II

- Nun kann sich der Client beim Server mittels SASL authentifizieren.
- Diese Schritte funktionieren auch ganz ähnlich bei einer Server-zu-Server Verbindung. Hier werden aber häufig schwachere Authentifizierungsverfahren verwendet.
- Zum Schluss muss der Client eine Resource binden, sprich er muss eine full JID angeben.
- Diese full JID erlaubt es dem Server mehrere simultan verbundene Clients desselber Nutzers (derselben bare JID) auseinander zu halten und korrekt anzusprechen.

XML Stanzas

- Die Informationselemente die in XMPP über den XML Stream verschickt werden, werden Stanza genannt.
- Level 1 Elemente von einem der drei oben genannten Typen sind valide XML Stanza.
- Wird auf ein Stanza eine Antwort erwartet (etwa bei einer Abfrage), so kann es mit einer id ausgestattet werden, damit die Antwort später richig eingeordnet werden kann.
- Ein Client schickt Stanzas immer an den Server. Dieser verarbeitet das Stanza entweder selbst (wenn es an ihn gerichtet ist), oder leitet es zum entsprechenden Client oder Server weiter.

Message Stanzas

- Eine Nachricht, die meistens an eine bare oder sogar an eine full JID gerichtet ist.
- Die Nachricht wird gepusht. Der Server wird also aktiv versuchen, die Nachricht zum richten Client zu Routen.

Presence Stanzas

- Diese Stanza werden meistens im Zusammenhang mit der pub-sub Architektur verwendet (siehe Folie 56).
- Ein Client schickt ein Presence Stanza ohne Angabe eines spezifischen Empfängers an den Server, der diese dann an alle Abonennten dieses Informationstypes weiterleitet.
- Typischerweise gebraucht, um etwa die Verfügbarkeit beim Chatten mitzuteilen.
- Der Push Mechanismus ist sowohl für den Server als auch für die Clients viel effizienter, als wenn alle Abonennten die Information regelmässig abfragen (pullen) würden.

IQ Stanzas I

- Hierbei handelt es sich um einen “request-response“ Mechanismus, mit dem ein Client gezielt Informationen abfragen kann (von einem Server oder einem anderen Client).
- Alle IQ (Info/Query) Stanzas müssen ein <id>Tag aufweisen.
- Es gibt vier Typen von IQ Stanzas:

  get Eine Abfrage/ ein Query. Sollte beantwortet werde.

  set Enthält Informationen die der Empfänger behalten soll, verwenden soll.

IQ Stanzas II

result Beantwortet ein Query.

error Informiert darüber, dass beim Verarbeiten eines get oder set IQs etwas schief gelaufen ist.

- Werden etwa verwendet, um den Roster zu verwalten (ab Folie 42).
- Oder um Verbindungen zwischen zwei Clients auszuhandeln (siehe Kapitel 3 und 4).

Subsection 2

Verteilte Infrastruktur

Grundidee

- Wie auch Email, soll XMPP Domänenübergreifend funktionieren.
- Dadurch, dass die JIDs den Domainnamen enthalten, ist ihre weltweite Eindeutigkeit sichergestellt.
- Ähnlich wie bei Email, müssen Server in der Lage sein, anhand der JID den entsprechenden Server zu finden.
- Es ist aber durchaus möglich, XMPP Server abgesondert (zum Beispiel im Firmennetzwerk) zu betreiben.

Finden von Servern I

- Wie auch bei Email, wir das DNS verwendet, um zu einer Domäne gehörende XMPP Server zu finden.
- Anders als bei Email, das den dedizierten MX Record hat, hat XMPP aber keinen eigenen Record Typ erhalten.
- Stattdessen verwendet XMPP, so wie auch SIP, die SRV Records [2].
  - Das hat ausserdem den Vorteil, dass nicht nur die Adresse des Servers, sondern auch der Port mitgeteilt werden kann.
  - Es gibt zwei SRV Records für XMPP, einen für die Client-zu-Server- und einen für die Server-zu-Server Verbindung.
- Als Fallback wird der A-record der Domäne verwendet.

Finden von Servern II

- Alternativ kann auch der FQDN des Servers verwendet werden. Die JID ist dann von der Art “foo@xmpp.example.com“.

Listing 1: XMPP SRV records für example.net

```
1  xmpp.example.net  IN  A   1.2.3.4
2  _xmpp-client._tcp.example.net. 68400 IN SRV 20 0 5222 xmpp.example.net.
3  _xmpp-server._tcp.example.net. 68400 IN SRV 20 0 5269 xmpp.example.net.
```

Client zu Server zu Server zu Client I

- Kommunizieren zwei Clients unterschiedlicher Server zusammen, so werden sie nicht direkt verbunden.
- Stattdessen läuft ihre Verbindung über die beiden Server, in einer Art Client-zu-Server-zu-Server-zu-Client Verbindung.
- ähnlich wie auch Emails von einem MUA zu einem anderen MUA über die Email Server wandern.

Client zu Server zu Server zu Client II

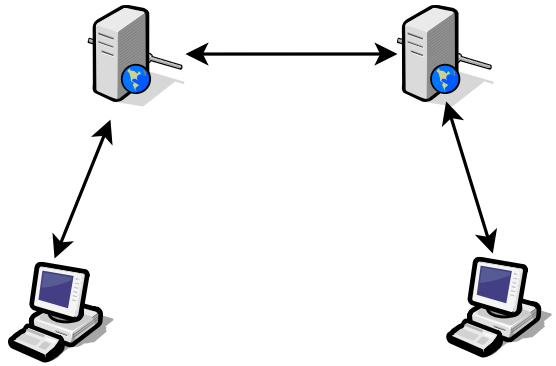


Abbildung: Zwei XMPP Clients verschiedener Domains die miteiannder kommunizieren.

Immer aktives TLS

- Eine grosse zahl von XMPP/Jabber Server Betreibern hat ein Manifest unterzeichnet, wonach sie ab dem 19 May 2014 alle XMPP/Jabber Verbindungen verschlüsseln wollen.
- Ausserdem sollen nur noch verschlüsselte Verbindungen zugelassen werden.
- Als Mittel der Wahl gilt TLS.
- `https://github.com/stpeter/manifesto/blob/master/manifesto.txt`
- Durch diese Massnahme sollen Nutzer von XMPP/Jabber Diensten bestmöglich geschützt werden.

Subsection 3

Beispiele

Client zu Server, Beispiel I

Listing 2: Einfaches Beispiel in dem sich ein Client zu einem Server verbindet

```
1  <!-- Client oeffnet den Stream -->
2  <stream:stream from='alice@im.example.com' to='im.example.com'
3          version='1.0' xml:lang='en' xmlns='jabber:client'
4          xmlns:stream='http://etherx.jabber.org/streams'>
5
6  <!-- Server antwortet in dem er den Antwortstream oeffnet -->
7  <stream:stream from='im.example.com' id='t7AMCin9zjMNwQKDnplntZPIDEI='
8          to='alice@im.example.com' version='1.0'
9          xml:lang='en' xmlns='jabber:client'
10         xmlns:stream='http://etherx.jabber.org/streams'>
```

- An dieser Stelle würder der Server dem Client mitteilen, dass er TLS unterstützt, woraufhin die beiden eine gesicherte Verbindung aushandeln würden.

Client zu Server, Beispiel II

- Sobald das geschehen ist, würde sich der Client mittels SALS beim Server authentifizieren.
- Über die sichere und authentifizierte Verbindung, bauen die beiden jetzt neue XML Streams auf.

Listing 3: Der Client bindet eine Resource und schickt erste Stanza

```
1  <!--    Der Server informiert den Client, dass
2          er die Funktion des Resourcen Bindens kennt -->
3  <stream:features>
4    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
5  </stream:features>
6
7  <!-- Der Client bindet nun die Resource foobar -->
8  <iq id='yhc13a95' type='set'>
9    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
10     <resource>foobar</resource>
```

Client zu Server, Beispiel III

```
11    </bind>
12  </iq>
13
14  <!--    Der Server antwortet auf das IQ (die Antwort bezieht
15          sich auf das set, weshalb sie dieselbe id traegt) -->
16  <iq id='yhc13a95' type='result'>
17    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
18      <jid>
19        alice@im.example.com/foobar
20      </jid>
21    </bind>
22  </iq>
23
24  <!--    Der Client kann jetzt XML Stanza schicken, in diesem Fall
25          Um eine Kommunikation mit Bob zu starten -->
26  <message from='alice@im.example.com/foobar'
27    id='ju2ba41c' to='bob@example.net' type='chat' xml:lang='en'>
28    <body>Hallo Bob!</body>
29  </message>
```

Server zu Server, Beispiel I

Listing 4: Der Verbindungsaufbau zwischen zwei Servern ist ganz ähnlich wie von einem Client zu einem Server

```
1  <stream:stream from='im.example.com'
2           to='example.net' version='1.0' xmlns='jabber:server'
3           xmlns:stream='http://etherx.jabber.org/streams'>
4
5  <stream from='example.net'
6           id='hTiXkW+ih9k2SqdGkk/AZi0OJ/Q=' to='im.example.com'
7           version='1.0' xmlns='http://etherx.jabber.org/streams'>
```

- Auch hier würden die beiden Server jetzt eine gesicherte Verbindung aufbauen und einander gegenseitig authentifizieren.

Server zu Server, Beispiel II

- Auch wenn SASL vorgesehen ist für Server zu Server Authentifiizierung, werden in der Praxis oft einfachere Methoden verwendet.
- Zum Schluss bauen die beiden Server über die gesicherte und authentifizierte Verbindung neue XML Streams auf.

Listing 5: Kommunikation der beiden Server

```
1  <!--    Der Server zu welchem die Verbindung aufgebaut wurde, informiert
2          sein Gegenueber ueber die von ihm unterstuetzten Funktionen.
3          In diesem Falle keine (Server kommunizieren nur miteinander,
4          um Client Nachrichten auszutauschen) -->
5  <features xmlns='http://etherx.jabber.org/streams'/>
6
7  <!--    Der Server im.example.com kann die Nachricht die er im ersten
       ↪ Beispiel
```

Server zu Server, Beispiel III

```
8           erhalten hat, jetzt an example.net weiterleiten, damit dieser
9           die Nachricht an ihren eigentlichen Empfaenger, bob@example.net
10          zustellen kann -->
11  <message from='alice@im.example.com/foobar' id='ju2ba41c'
12      to='bob@example.net' type='chat' xml:lang='en'>
13    <body>Hallo Bob!</body>
14  </message>
```

- Die beiden Server würden die Verbindung nun für eine Weile offen lassen. Sie können ja nicht wissen, ob diese noch gebraucht wird oder nicht.
- Wenn aber einer der beiden Server beschliesst, dass er die Verbindung nicht mehr braucht/will, kann er den XML Stream schliessen.

Server zu Server, Beispiel IV

Listing 6: Die Server schliessen die XML Streams

```
<!--    Der erste der beiden Server schliesst den Stream und teilt
        dem anderen dadurch indirekt mit, dass er die Verbindung
        beenden will -->
</stream:stream>

<!--    Da der zweite Server em ersten nichts mehr mitzuteilen hat,
        beendet auch er den XML Stream. -->
</stream>

<!--    Die beiden Server koennen jetzt alle noch bestehenden Verbindungen
        (TLS, TCP, ...) abbauen. -->
```

Section 3

Messaging

Direkter Chat

- RFC 6121 spezifiziert Chatsessionen zwischen zwei Nutzern.
- Ausserdem den Umgang mit der Liste von Kontakten und dem verteilen und abonnieren von Statusänderungen.
- RFC 6121 erfüllt die meisten Anforderungen von RFC 2779 [1].
- Dies ist wohl auch der Grund, weshalb 6121 als RFC und nicht als XEP verfasst worden ist.

Subsection 1

Roster

Über den Roster

- Der Roster (zu Deutsch etwa die Anschriftenliste) enthält Informationen zu allen Kontakten, wie etwa deren:
  - bare JID
  - Nickname
  - Gruppe in die der Kontakt eingeordnet ist
  - Informationen zu den Subskriptionen
- Der Roster wird vom Server gespeichert und verwaltet.
- Der Roser wird vom Server nach dessen Gutdünken nummeriert (versioniert).

Roster verwalten

- Zum Verwalten des Rosters werden IQ Stanzas verwendet.
- Mit einem “Roster Get“ kann der Client den Roster beim Server abfragen, dieser antwortet mit einem “Roster Result“.
- Für gewöhnlich verlangt der Client den Roster gleich nach dem Login.
- Zum Verwalten des Rosters werden IQ Stanzas vom Typ “set“ verwendet. Genannt wird das ein “roster set“.
- Wird der Roster eines Nutzer verändert, so hat der Server die Änderung, mittels eines “Roster Push“ allen verbundenen Resourcen dieses Nutzers mitzuteilen.

Kontakte verwalten

- Um einen neuen Kontakt hinzuzufügen, sendet der Client das entsprechende roster set an den Server. Dabei handelt es sich um ein XML Element, dass der Server (nach allfälliger Ergänzung) in den Roster aufnehmen kann.
- Ein Element im Roster zu Ändern funktioniert aequivalent zum Hinzufügen von Elementen. Die neu verschickte Version muss vom Server übernommen werden.
- Soll ein Kontakt aus dem Roster entfernt werden, so ist das Attribut “subscription“ auf “remove“ zu setzen.

Abfragen des Rosters, Beispiel I

Listing 7: Der Client fragt beim Server den Roster ab und erhält die entsprechende Antwort

```
1  <iq type='get' id='purple56c57744'>
2    <query xmlns='jabber:iq:roster'/>
3  </iq>
4
5  <iq id='purple56c57744' type='result' to='vimja@xmpp.honet.ch/8ff3db1d-3c60-4
       ↪ bea-b68e-fa3ad5f4fdf5'>
6    <query xmlns='jabber:iq:roster' ver='235'>
7      <item jid='draemmli@jabber.lugs.ch' subscription='both' name='draemmli'>
8              <group>Buddies</group>
9      </item>
10     <item jid='stefan@jabber.lugs.ch' subscription='both' name='Stefan (lugs)
           ↪ '>
11             <group>Buddies</group>
12     </item>
13     <item jid='lux@c-base.de' ask='subscribe' subscription='none'/>
14     <item jid='bash.vi@gmail.com' subscription='both' name='Bash Vi (gmail)'>
15             <group>Buddies</group>
16     </item>
```

Abfragen des Rosters, Beispiel II

```
17      <item jid='niklaus.hofer@gmail.com' subscription='both' name='Niklaus
            ↪ Hofer (gmail)'>
18              <group>Buddies</group>
19      </item>
20      <item jid='stefan@honet.ch' subscription='both'>
21              <group>Buddies</group>
22      </item>
23      <item jid='vimja@jabber.lugs.ch' subscription='both' name='vimja (lugs)'>
24              <group>Buddies</group>
25      </item>
26    </query>
27  </iq>
```

Löschen eines Kontaktes, Beispiel I

Stefan hat zwei Adressen. Eine davon ist alt und unbenutzt, die wollen wir entfernen.

Listing 8: Wir entfernen einen Kontakt aus unserem Roster

```
<iq id='lx249' type='set'>
  <query xmlns='jabber:iq:roster' ver='236'>
    <item jid='stefan@jabber.lugs.ch' subscription='remove'/>
  </query>
</iq>

<!-- Der Server informiert uns, dass alles geklappt hat -->
<iq type='result' id='lx249'/>

<!-- Roster Push -->
<iq id='lx249' type='set'>
  <query xmlns='jabber:iq:roster' ver='236'>
    <item jid='stefan@jabber.lugs.ch' subscription='remove'/>
  </query>
</iq>
```

Löschen eines Kontaktes, Beispiel II

```
16
17  <!-- Wir bestaetigen den Erhalt des Roster Push -->
18  <iq type='result' id='lx249'/></query></iq>
```

Subsection 2

Pub-Sub

Die Idee

- Eine wichtige Funktion des Roster haben wir bisanhin noch nicht betrachtet: Das Verwalten von Subskriptionen.
- Möchte man künftig über die Präsenzänderungen eines Kontaktes informiert werden, so muss man bei diesem Kontakt nach der Bewilligung fragen.
- Der Roster speichert für alle Kontakte, welchen Subskriptionsstatus sie haben:

  from Wir teilen dem Kontakt unseren Status mit

  to Wir erhalten Nachrichten über Statusänderungen dieses Kontaktes

  both Sowohl als auch

Ablauf beim Subscriben I

- Um die Präsenz eines Kontaktes zu abonnieren, generiert unser Client einen “Subscription Request“.
- Dieser wird vom Client zu unserem Server geschickt.
- Der Server routed den Request weiter zum Server des Kontaktes (oder verarbeitet ihn selbst, falls der Kontakt auf demselber Server ist wie wir).
- Ausserdem fügt der Server den Kontakt zu unserem Roster hinzu mit dem Subskriptions Status “none“ und einem Vermerk, dass wir auf eine Antwort warten.

Ablauf beim Subscriben II

- Der Server des Kontaktes wird, sobald er den “Subscription Request“ erhält, diesen an alle zur Zeit angemeldeten Resourcen des Kontaktes senden.
  - Sollten zur Zeit keine Resourcen online sein, so behält der Server den Request, bis Resourcen verfügarbar sind.
- Alle Resourcen des Kontaktes die den Request erhalten, müssen diesen dem Kontakt präsentieren. Dieser entscheidet dann, ob er den Request annimmt oder ablehnt.
- Eine entsprechende Antwort wird von der Resource des Kontaktes an dessen Server geschickt.

Ablauf beim Subscriben III

- Wurde der Request angenommen, so trät des Kontaktes Server uns in den Roster des Kontaktes ein, mit dem Subskriptionsstatus “from“.
- Ausserdem sendet des Kontaktes Server die Nachricht, dass die Subskription angenommen wurde an unseren Server.
- Unser Server kann anschliessend den Subskriptions Status des Kontaktes in unserem Roster anpassen (auf “to“ setzen).
- Die meisten Clients werden, nimmt man einen Subskriptions Request an, einen solchigen in die andere Richtung senden.

Ablehnen eines Requests und Löschen von Subskriptionen

(Wir werden den Kontakt nicht länger über unseren Status informieren)

- Das Ablehnen eines Subcription Request und das Löschen einer Subskription laufen gleich ab:
  - Es wird ein “presence“ Stanza vom Typ “unsubscribed“ verschickt.
- Die Server müssen die Roster entsprechend anpassen.

Von einem Kontakt unsubscriben I

(wir wollen nicht länger über Statusänderungen des Kontaktes informiert werden)

- Ein “presence“ Stanza vom Typ “unsubscribe“ wird von unserem Client an unseren Server geschickt.
- Unser Server leitet das Stanza an den Server des Kontaktes weiter.
- Beide Server müssen die Roster entsprechend anpassen:
  - Hatten wir in unserem Roster für den Kontakt einen subscription status von “to“, so muss er auf “none“ geändert werden, war er aber auf “bot“, so muss er auf “from “ geändert werden.

Von einem Kontakt unsubscriben II

- Hatte der Kontakt in seinem Roster für uns einen subscription status von “from“ so muss er ihn auf “none“ ändern, war er zuvor “both“ so muss er jetzt zu “to“ geändert werden.

Aktualisieren des Status I

- Hat man die Funktionsweise des Rosters verstanden, so ist die Funktionsweise die dem Aktualisieren des Status zu Grunde liegt sehr intuitiv.
- Unser Client erstellt ein Stanza vom Typ “presence“ das kein “to“ Attribut aufweist und sendet dieses an unseren Server.
- Der Server Sammelt nun alle Kontakte unseres Rosters deren subscription status entweder auf “both“ oder “from“ steht.

Aktualisieren des Status II

- Für jeden der im vorigen Schritt selektierten Kontakte, erstellt unser Server ein “presence“ Stanza, dass er mit einem “from“ Attribut (befüllt mit unserer full JID) und einem “to“ Attribut (befüllt mit der bare JID des Kontaktes) ausstattet.
- Dieses Stanza sendet unser Server nun an den Server des Kontaktes.
- Dieser sendet das Stanza nun an alle verfügbaren Resourcen des Kontaktes.
- Die Resourcen des Kontaktes die das Stanza erhalten haben, können nun unseren Status entsprechend anzeigen.

Aktualisieren des Status III

- Der zu verwendende Status wird in einem <show>Tag innerhalb des “presence“ Stanzas transportiert.
- Der initiale “presence“ Status den der Client gleich nach dem Login übermittelt, hat kein <show>Tag. Es handelt sich um eines leeres “presence“ Stanza.

Subsection 3

Chat

Aufbau eines Chats

- Frühere Versionen von XMPP haben vorgesehen, dass beim Beginn eines Chattes, dieser formal initiiert wird. Das ist in aktuellen Versionen aber nicht mehr vorgesehen.
- Stattdessen, sendet unser Client eine Nachricht (Stanza vom Typ “message“ mit einem <body>Tag), die an den Kontakt adressiert ist zu unserem Server.
- Die erste Nachricht eines Chattes sollte an die bare JID des Kontates adressiert sein und nicht an die full JID.
- Unser Server leitet das Stanza an den Server des Clients weiter.

Die Rolle von Resourcen und priorities

- Ist die Nachricht an die bare JID des Kontaktes adressiert, so ist es des Kontaktes Server überlassen, an welche Resource(n) er die Nachricht senden will.
- Der Server kann die priority sowie die presence der Resourcen hinzuziehen um abzuwägen, wohin er die Nachricht schicken soll.
- Er kann die Nachricht aber auch an alle verfügbaren Resource broadcasten.

Fortsetzung des Chats

- Die Antwort des Kontaktes ist an unsere full JID (diese war in unserer ursprünglichen Nachricht enthalten) adressiert. Dadurch wird gewährleistet, dass sie bei derselben Resource eintrifft, von welcher aus wir den Chat initiiert haben.
- Ausserdem ist die full JID der Resource von welcher aus der Kontakt die Antwort verschickt hat in dieser enthalten. Von nun an, sollten wir Nachrichten dieser Chat Session an diese full JID adressieren.
- Ein Chat kann ein <thread>Tag enthalten. Dieses hilft, die Nachrichten einer bestimmten Chat Session zuzuordnen, ist aber optional.

Offline Nachrichten

- Ist zum Zeitpunkt zu dem eine Nachricht für einen Nutzer beim Server eintrifft keine Resource dieses Nutzers verfügbar, so kann der Server die Nachricht speichern, bis wieder Resourcen verfügbar sind.
- Die Regeln dazu sind zu einem grossen Teil eine Frage der Implementation.
- XEP-0160[10] beschreibt best practices für diese Regeln.

Ein typischer Chat Verlauf, Beispiel I

Listing 9: In etwa so verläuft ein typischer Chat idealerweise

```
<!-- die initiale Nachricht geht an die bare JID von bob -->
<message from='alice@example.com/foobar'
         to='bob@example.net' type='chat'
         xml:lang='en'>
 <body>Hallo Bob!</body>
</message>

<!-- Die Antwort ist an die full JID von Alice gerichtet -->
<message from='bob@example.net/quxbaz'
         to='alice@example.com/foobar'
         type='chat' xml:lang='en'>
  <body>Hei Alice, wie gehts so?</body>
</message>

<!-- Von nun an kann Alice an Bobs full JID schreiben -->
<message from='alice@example.com/foobar'
         to='bob@example.net/quxbaz'
         type='chat' xml:lang='en'>
```

Ein typischer Chat Verlauf, Beispiel II

```
19    <body>Hmja, geht ganz gut, danke :)</body>
20  </message>
```

Section 4

Extensions

Subsection 1

In-Band Registrierung

- Definiert im XEP-0077 [13]
- Ermöglicht dem Nutzer eine Eigenständige registrierung beim Server.
- Der Nutzer kann sein Konto auf dem Server auflösen.
- Ermöglicht das ändern des Passwortes.
- Auf dem Server kann spezivziert werden welche Informationen vom Nutzer für eine Registrierung verlangt werden, die Clientsoftware kann das im Vorfeld der Registrierung beim server abfragen.
- Einige Server software bieten dem Nutzer ein Webgui an um sich zu registrieren. [5] [7]

Registrierung, Beispiel I

Listing 10: Der Client fragt die benötigten Felder beim Server ab

```
1       <iq type='get' id='reg1' to='shakespeare.lit'>
2         <query xmlns='jabber:iq:register'/>
3       </iq>
```

Im Vorfeld eine Registrierung sollte die Clientsoftware den Server fragen, welche Informationen für eine Registrierung benötigt werden, da ansonsten die Registrierung vom server abgelehnt wird.

Registrierung, Beispiel II

Listing 11: Beispiel für eine Antwort des Servers

```
1  <iq type='result' id='reg1'>
2    <query xmlns='jabber:iq:register'>
3      <instructions>
4        Choose a username and password for use with this service.
5        Please also provide your email address.
6      </instructions>
7      <username/>
8      <password/>
9      <email/>
10   </query>
11 </iq>
```

Falls der Client schon registriert ist fügt der Server ein lehres <registered\>-tag hinzu. Der Server entscheidet anhand der from Adresse ob der User schon vorhanden ist.

Registrierung, Beispiel III

Listing 12: Der Nutzer muss Invormationen zu allen benötigten Feldern and den Server übermittln

```
<iq type='set' id='reg2'>
  <query xmlns='jabber:iq:register'>
    <username>bill</username>
    <password>Calliope</password>
    <email>bard@shakespeare.lit</email>
  </query>
</iq>
```

Listing 13: der Server informiert das die Registrierung erfolgreich verarbeitet wurde

```
<iq type='result' id='reg2'/>
```

Konto schliessen, Beispiel I

Listing 14: Der Nutzer möchte ein Konto schliessen

```
<iq type='set' from='bill@shakespeare.lit/globe' id='unreg1'>
  <query xmlns='jabber:iq:register'>
    <remove/>
  </query>
</iq>
```

Listing 15: der Server informiert den Client das die Anfrage verarbeitet und das Konto geschlossen wurde

```
<iq type='result' to='bill@shakespeare.lit/globe' id='unreg1'/>
```

Subsection 2

Multiuser chat

Das Konzept von Chatrooms

- Multiuser-Chat, auch Text-conferencing genannt ist nicht im RFC6120 beschrieben.
- Die Erweiterung XEP-0045[14] beschäftigt sich mit damit.
- Das Resultat sind Chatrooms, ähnlich wie man sie von IRC her kennt mit topics, Moderatoren, invitations, ...
- Ähnlich wie bei IRC sind eine Menge Kommandos möglich um die Chatrooms zu kontrollieren, Teilnehmer zu verbannen, den Nick zu ändern...
- Normalerweise erhalten die Chatrooms eines IRC Servers eine eigene Subdomain, häufig “conference.example.com“.

Chatrooms, JIDs und Nicks

- Jeder Chatroom hat eine eigene “Room JID“ nach dem Format “roomname@conference.example.com“.
  - deshalt ist es auch sinnvoll eine eigene Subomain zu verwenden, damit Nutzernamen und Chatroom Namen nicht kollidieren und man sie gut auseinander halten kann.
- Die Teilnehmer die sich in einem Chatroom befinden sind identifiziert als “roomname@conference.example.com/NICK“. Dies nennt man die “Occupant JID“

Verbindungsaufbau

- Mittels eines IQ kann man bei einem Server die verfügbaren Chatrooms abfragen.
- Zum Betreten eines Chatrooms, wird der Status an den Channel an die eigenen Occupant JID.
- Optional kann ein Room ein Passwort verlangen, welches beim Betreten angegeben werden muss.
- Zum Ändern des eigenen Nicknamens, sendet man den eigenen Status an die neue Occupant JID.

Chat

- Um mit den Teilnehmern des Raumes zu schreiben, wird die Nachricht an die Room JID geschickt.
- Der Server kümmert sich darum, diese Nachricht an alle Teilnehmer zu verteilen.
- IQs werden vom Server selbst behandelt und nicht an die Teilnehmer weitergeleitet, so dass etwa die Initiierung eines Filetransfers nicht möglich ist.

Chatroom discovery, Beispiel

Listing 16: Der Client fragt beim Server nach einer Liste der verfübaren Chatrooms

```
<iq type='get' id='purple1f4ab212' to='conference.hofer.lan'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

Listing 17: Die Antwort des Servers

```
<iq type='result' id='purple1f4ab212' from='conference.hofer.lan' to='
    ↪ bar@hofer.lan/pidgin01'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='test@conference.hofer.lan' name='Test'/>
    <item jid='foo@conference.hofer.lan' name='foo'/>
    <item jid='example@conference.hofer.lan' name='example'/>
  </query>
</iq>
```

Raum Betreten und Nick ändern, Beispiel I

Listing 18: Betreten des Raumes “gentoo“ auf dem conference Server von jabber.org

```
1  <presence to='gentoo@conference.jabber.org/vimja'>
2          <status>I'm here right now</status>
3          <priority>5</priority>
4          <c xmlns='http://jabber.org/protocol/caps' node='http://pidgin.im/'
               ↪ hash='sha-1' ver='AcN1/PEN8nq7AHD+9jpxMV4U6YM=' ext='voice-v1
               ↪  camera-v1 video-v1'/>
5          <x xmlns='http://jabber.org/protocol/muc'/>
6  </presence>
```

- Der Server verschickt die Nachricht an alle Teilnehmer, damit alle mitbekommen, dass ein neuer Teilnehmer den Raum betreten hat.

Raum Betreten und Nick ändern, Beispiel II

- Neben den Presence Informationen der anderen Teilnehmer kriegen wir also auch die eigene zurück.
- Zum Schluss erhalten wir noch das Topic des Channels mitgeteilt.

Listing 19: Wir erhalten die Liste der Teilnehmer und das Topic des Channels.

```
1  <presence to='vimja@xmpp.honet.ch/ca363165-583a-41e7-9628-840fb12078cd' from=
       ↪ 'gentoo@conference.jabber.org/fsteinel'>
2          <c xmlns='http://jabber.org/protocol/caps' hash='sha-1' ver='/9
               ↪ bcBFrzR41+CYGgpdnYcTm2dpw=' node='http://slixmpp.com/ver
               ↪ /1.3.1'/>
3          <x xmlns='http://jabber.org/protocol/muc#user'>
4                  <item affiliation='none' role='participant'/>
5          </x>
6  </presence>
7
```

Raum Betreten und Nick ändern, Beispiel III

```
8  <!-- Weitere Teilnehmer hier -->
9
10 <presence to='vimja@xmpp.honet.ch/ca363165-583a-41e7-9628-840fb12078cd' from=
       ↪ 'gentoo@conference.jabber.org/vimja'>
11        <status>I'm here right now</status>
12        <priority>5</priority>
13        <c xmlns='http://jabber.org/protocol/caps' hash='sha-1' ext='voice-v1
              ↪  camera-v1 video-v1' ver='AcN1/PEN8nq7AHD+9jpxMV4U6YM=' node=
              ↪ 'http://pidgin.im/'/>
14        <x xmlns='http://jabber.org/protocol/muc#user'>
15              <item affiliation='none' role='participant'/>
16              <status code='110'/>
17        </x>
18 </presence>
19
20 <message type='groupchat' to='vimja@xmpp.honet.ch/ca363165-583a-41e7-9628-840
       ↪ fb12078cd' from='gentoo@conference.jabber.org'>
21        <subject>Gentoo Linux :: http://www.gentoo.org :: gentoo@irc.freenode
              ↪ .org :: Languages: EN, RU</subject>
22        <delay xmlns='urn:xmpp:delay' stamp='1970-01-01T00:00:00Z' from='
              ↪ gentoo@conference.jabber.org'/>
23        <x xmlns='jabber:x:delay' stamp='19700101T00:00:00' from='
              ↪ gentoo@conference.jabber.org'/>
```

Raum Betreten und Nick ändern, Beispiel IV

```
24 </message>
```

- Um unseren Nick zu ändern, schicken wir die Presence Information an die neue Occupant JID.
- Anschliessend kriegen alle Teilnehmer (also auch wir) zwei Dinge mitgeteilt:
  - Der alte Nick (die alte Occupant JID) ist nicht länger verfügbar.
  - Die Presence unter dem neuen Nick.

Raum Betreten und Nick ändern, Beispiel V

Listing 20: Hier ändern wir unseren Nicknamen und schauen, was passiert.

```
<presence to='gentoo@conference.jabber.org/vimja42'>
        <status>I'm here right now</status>
        <priority>5</priority>
        <c xmlns='http://jabber.org/protocol/caps' node='http://pidgin.im/'
            ↪ hash='sha-1' ver='AcN1/PEN8nq7AHD+9jpxMV4U6YM=' ext='voice-v1
            ↪  camera-v1 video-v1'/>
</presence>

<presence type='unavailable' to='vimja@xmpp.honet.ch/ca363165-583a-41e7
    ↪ -9628-840fb12078cd' from='gentoo@conference.jabber.org/vimja'>
        <x xmlns='http://jabber.org/protocol/muc#user'>
                <item role='participant' affiliation='none' nick='vimja42'/>
                <status code='110'/>
                <status code='303'/>
        </x>
</presence>

<presence to='vimja@xmpp.honet.ch/ca363165-583a-41e7-9628-840fb12078cd' from=
    ↪ 'gentoo@conference.jabber.org/vimja42'>
```

Raum Betreten und Nick ändern, Beispiel VI

```
16          <status>I'm here right now</status>
17          <priority>5</priority>
18          <c xmlns='http://jabber.org/protocol/caps' hash='sha-1' ext='voice-v1
                ↪  camera-v1 video-v1' ver='AcN1/PEN8nq7AHD+9jpxMV4U6YM=' node=
                ↪ 'http://pidgin.im/'/>
19          <x xmlns='http://jabber.org/protocol/muc#user'>
20                  <item affiliation='none' role='participant'/>
21                  <status code='110'/>
22          </x>
23  </presence>
```

Chaten im MUC, Beispiel

Listing 21: Schicken wir eine Nachricht an die Room JID, so verteilt der Server diese an alle Teilnehmer

```
1  <message type='groupchat' id='purpled3ba7607' to='gentoo@conference.jabber.
       ↪ org'>
2          <body>hey</body>
3  </message>
4  <message type='groupchat' to='vimja@xmpp.honet.ch/ca363165-583a-41e7-9628-840
       ↪ fb12078cd' from='gentoo@conference.jabber.org/vimja'>
5          <body>hey</body>
6  </message>
```

Subsection 3

File transfer

SI File Transfer I

- Stream Initiation (SI) File Transfer beschreibt ein Verfahren zum Aushandeln eines Bytestreams über welchen eine Datei übertragen wird.
- Der Sender der Datei schickt einen “Stream Initiation Offer“. Dieser enthält Metainformationen zur Datei und zum Stream:
  - Name und Grösse der Datei (mandatory)
  - MD5 Checksumme der Datei zum Prüfen ob die Übertragung korrekt verlief (optional)
  - Weitere Metainformationen
  - Die von Sender unterstützten Streamingverfahren.

SI File Transfer II

- Mit dem “Stream Initiation Result“ Teilt der Empfänger dem Sender mit, dass er die Datei gerne empfangen würde und welches der vom Sender offerierten Streamingverfahren er dazu verwenden möchte.
- Das Aushandeln der Dateiübertragung geschieht mittels IQ (Info/Query) Stanzas.

SI File Transfer Bytestreams

- SI File Transfer nach XEP-0096 [6] kennt zwei Bytestreams die verwendet werden können:
  - IBB - In-Band Bytestreams (XEP-0047[3])
  - Oub-of-Band SOCKS5 Bytestreams (XEP-0065[17])
- IBB verwendet encodiert die Daten mit BASE64 und verschickt sie anschliessend als IQ Stanzas über XMPP.
- IBB ist sehr langsam
- Die SOCKS5 Bytestreams sind die bevorzugte Variante zur Dateiübertragung, da sie viel schneller sind und den Server weniger belasten.
- Der SOCKS5 Bytestream kann wahlweise direkt zwischen den Clients (P2P) oder über einen Proxy aufgebaut werden.

Andere Arten der Dateiübertragung

- Zu den oben beschriebenen Methoden gibt es neure Alternativen, die aber nur von wenigen Clients unterstützt werden.
- XEP-0129[16]: WebDAV File Transfer
  - Erlaubt das Senden einer Datei, auch wenn der Empfänger zur Zeit nicht verfügbar ist.
  - Die Datei wird dazu auf einem WebDAV Speicher hinterlegt, von wo sie der Empfänger später abrufen kann.
- XEP-0234[15]: Jingle File Transfer
  - Verwendet Jingle P2P Verbindungen um Dateien zu übertragen.
  - Anders als bei der Jingle Audio-/Video- Telefonie, werden TCP Streams verwendet.

Initialiseren eines Filetransfers, Beispiel I

Listing 22: foo bietet bar den Transfer der Datei “XMPP_logo.svg.png“. Er bietet Bytestreams (XEP-0065 + XEP-0096) und in-band-bytestream (ibb, XEP-0047) an.

```
1  <iq from="foo@andonie.lan/pidgin01" to="bar@hofer.lan/pidgin01" type="set" id
       ↪ ="purple3a8b361c">
2    <si xmlns="http://jabber.org/protocol/si" id="purple3a8b361d" profile="
         ↪ http://jabber.org/protocol/si/profile/file-transfer">
3      <file xmlns="http://jabber.org/protocol/si/profile/file-transfer" name="
           ↪ XMPP_logo.svg.png" size="72357"/>
4      <feature xmlns="http://jabber.org/protocol/feature-neg">
5        <x xmlns="jabber:x:data" type="form">
6          <field var="stream-method" type="list-single">
7            <option>
8              <value>http://jabber.org/protocol/bytestreams</value>
9            </option>
10           <option>
11             <value>http://jabber.org/protocol/ibb</value>
12           </option>
13         </field>
14       </x>
```

Initialiseren eines Filetransfers, Beispiel II

```
15      </feature>
16    </si>
17  </iq>
```

Listing 23: (der Client von)bar entscheidet sich für den Bytestream und teilt dies foo mit

```
1   <iq type="result" to="foo@andonie.lan/pidgin01" id="purple3a8b361c">
2     <si xmlns="http://jabber.org/protocol/si">
3       <feature xmlns="http://jabber.org/protocol/feature-neg">
4         <x xmlns="jabber:x:data" type="submit">
5           <field var="stream-method">
6             <value>http://jabber.org/protocol/bytestreams</value>
7           </field>
8         </x>
9       </feature>
10    </si>
11  </iq>
```

Initialiseren eines Filetransfers, Beispiel III

Listing 24: foo teilt nun bar mit, von wo dieser die Datei streamen kann

```
1  <iq from="foo@andonie.lan/pidgin01" to="bar@hofer.lan/pidgin01" type="set" id
       ↪ ="purple3a8b361e">
2    <query xmlns="http://jabber.org/protocol/bytestreams" sid="purple3a8b361d">
3      <streamhost jid="foo@andonie.lan/pidgin01" host="192.168.0.173" port="
           ↪ 35774"/>
4    </query>
5  </iq>
```

Initialiseren eines Filetransfers, Beispiel IV

Listing 25: bar startet nun den Download und informiert foo darüber, welchen Streamhost er dazu verwendet

```
1  <iq type="result" to="foo@andonie.lan/pidgin01" id="purple3a8b361e">
2    <query xmlns="http://jabber.org/protocol/bytestreams">
3      <streamhost-used jid="foo@andonie.lan/pidgin01"/>
4    </query>
5  </iq>
```

- Der eigentliche Transfer der Datei geschieht ausserhalb der XMPP Verbindung und ist deshalb hier nicht sichtbar.

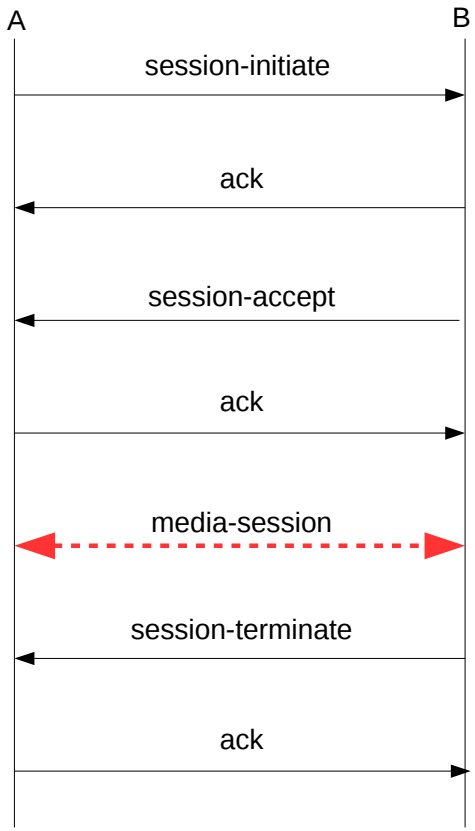
Subsection 4

Jingle

Übersicht

- Entwickelt von Google für Google Talk.
- Standartisiert in XEP-0166. [4]
- P2P Verbindungen für den Austausch von Medien.
- Es gibt zwei Arten wie Daten übertragen werden:
  - media streaming RTP
  - Datagram
- Inizialisierung und Abbau der Session über XMPP.
- Ermöglicht Dienste wie VoIP oder Vidochats.

Verbindungsaufbau zwischen Alice und Bob



Jingle Verbindungsaufbau, Beispiel I

Listing 26: Romeo möchte eine Session initialisieren

```
<iq from='romeo@montague.lit/orchard'
    id='zid615d9'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'
          action='session-initiate'
          initiator='romeo@montague.lit/orchard'
          sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='this-is-a-stub'>
      <description xmlns='urn:xmpp:jingle:apps:stub:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:stub:0'/>
    </content>
  </jingle>
</iq>
```

Jingle Verbindungsaufbau, Beispiel II

Listing 27: Juliet akzeptiert die Verbindung

```
<iq from='juliet@capulet.lit/balcony'
    id='rc61n59s'
    to='romeo@montague.lit/orchard'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'
          action='session-accept'
          responder='juliet@capulet.lit/balcony'
          sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='this-is-a-stub'>
      <description xmlns='urn:xmpp:jingle:apps:stub:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:stub:0'/>
    </content>
  </jingle>
</iq>
```

Jingle Verbindungsaufbau, Beispiel III

Listing 28: Juliet beendet die Session

```
<iq from='juliet@capulet.lit/balcony'
    id='le71fa63'
    to='romeo@montague.lit/orchard'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'
          action='session-terminate'
          sid='a73sjjvkla37jfea'>
    <reason>
      <success/>
    </reason>
  </jingle>
</iq>
```

Jingle Verbindungsaufbau, Beispiel IV

Listing 29: Beispiel für die Acknowledgements die zwischendurch gesendet werden

```
1 <iq from='juliet@capulet.lit/balcony'
2     id='ph37a419'
3     to='romeo@montague.lit/orchard'
4     type='result'/>
```

Bibliography I


M. Day, S. Aggarwal, G. Mohr, and J. Vincent.
Instant Messaging / Presence Protocol Requirements.
RFC 2779 (Informational), February 2000.


A. Gulbrandsen, P. Vixie, and L. Esibov.
A DNS RR for specifying the location of services (DNS SRV).
RFC 2782 (Proposed Standard), February 2000.
Updated by RFC 6335.


J. Karneges and P. Saint-Andre.
In-Band Bytestreams.
XEP-0047 (Standards Track), June 2012.

Bibliography II

S. Ludwig, J. Beda, P. Saint-Andre, R. McQueen, S. Egan, and J. Hildebrand.
Jingle.
XEP-0166 (Standards Track), December 2009.

mfoss.
mod_register_web - Web to register account.
https://www.ejabberd.im/mod_register_web, May 2008.

T. Muldowney, M. Miller, R. Eatmon, and P. Saint-Andre.
SI File Transfer.
XEP-0096 (Standards Track), April 2004.

Bibliography III

prosody modules.
mod_register_web.
https://code.google.com/p/prosody-modules/wiki/mod_register_web, January 2014.

P. Saint-Andre.
Extensible Messaging and Presence Protocol (XMPP): Core.
RFC 3920 (Proposed Standard), October 2004.
Obsoleted by RFC 6120, updated by RFC 6122.

Bibliography IV


P. Saint-Andre.
Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.
RFC 3921 (Proposed Standard), October 2004.
Obsoleted by RFC 6121.


P. Saint-Andre.
Best Practices for Handling Offline Messages.
XEP-0160 (Informational), January 2006.

Bibliography V

P. Saint-Andre.
Extensible Messaging and Presence Protocol (XMPP): Core.
RFC 6120 (Proposed Standard), March 2011.

P. Saint-Andre.
Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.
RFC 6121 (Proposed Standard), March 2011.

P. Saint-Andre.
In-Band Registration.
XEP-0077 (Standards Track), January 2012.

Bibliography VI


P. Saint-Andre.
Multi-User Chat.
XEP-0045 (Standards Track), February 2012.


P. Saint-Andre.
Jingle File Transfer.
XEP-0234 (Standards Track), August 2014.


P. Saint-Andre and D. Smith.
WebDAV File Transfers.
XEP-0129 (Standards Track), April 2007.

Bibliography VII


D. Smith, M. Miller, P. Saint-Andre, and J. Karneges.
SOCKS5 Bytestreams.
XEP-0065 (Standards Track), April 2011.