XMPP and Android

Florian Schmaus

Ignite Realtime

2015-01-31

XMPP

e**X**tensible **M**essaging and **P**resence **P**rotocol

- Allows to exchange data in form of XML elements between entities
- Specified by
  - RFC 6120 (XMPP-Core), RFC 6121 (XMPP-IM), RFC 6122 (JID)
  - various "XMPP Extension Protocols" (XEPs)
- Specifies 3 root elements, called "stanzas"

message	send asynchronous, fire-and-forget, store-and-forward
iq	request-response (response is mandatory)
presence	multicast to subscribed entities, pub/sub paradigm

XMPP (cont.)

eXtensible Messaging and Presence Protocol

XML?!

- Allows to extend the protocol without breaking compatibility
- You can encapsulate any data you want, for example JSON (XEP-295)
    - Be careful when doing so, parsing is always a possible attack vector
    - See "BlackPwn: BlackPhone SilentText Type Confusion Vulnerability" [2], for a case where XMPP encapsulated JSON parsing went wrong

XMPP (cont.)

e**X**tensible **M**essaging and **P**resence **P**rotocol

XML?!

- Allows to extend the protocol without breaking compatibility
- You can encapsulate any data you want, for example JSON (XEP-295)
  - Be careful when doing so, parsing is always a possible attack vector
  - See “BlackPwn: BlackPhone SilentText Type Confusion Vulnerability” [2], for a case where XMPP encapsulated JSON parsing went wrong

XMPP is not strictly an IM protocol! [4]

It allows you to exchange data between entities, and can therefore be used as protocol for Instant Messaging (IM), Social Media, the Internet of Things (IoT), Multi-Agent Systems (MAS), . . .

Smack

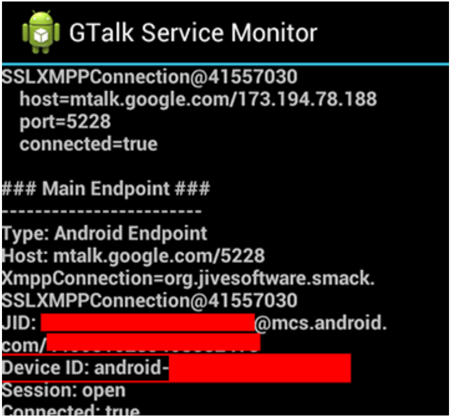
An Open Source XMPP Client Library written in Java for JVMs and Android

- Started by Jive Software in 2002
- Was first ported to Android by the end of 2007 [6]
- Jive founded the "Ignite Realtime" community 2009
- Rene Treffer created aSmack in 2009 for Buddycloud
- Native support for Android added with Smack 4.1 (beta)

Smack

An Open Source XMPP Client Library written in Java for JVMs and Android

- Started by Jive Software in 2002
- Was first ported to Android by the end of 2007 [6]
- Jive founded the “Ignite Realtime” community 2009
- Rene Treffer created aSmack in 2009 for Buddycloud
- Native support for Android added with Smack 4.1 (beta)
- Appears to be used (at least partly) by Google for the “GTalk Service”



GTalk Service Monitor

```
SSLXMPPConnection@41557030
  host=mtalk.google.com/173.194.78.188
  port=5228
  connected=true

### Main Endpoint ###
-----------------------
Type: Android Endpoint
Host: mtalk.google.com/5228
XmppConnection=org.jivesoftware.smack.
SSLXMPPConnection@41557030
JID: [redacted]@mcs.android.
com/[redacted]
Device ID: android-[redacted]
Session: open
Connected: true
```

Figure : GTalk Service Monitor on Android 2.2 / 2.3

Smack

Code example

```java
XMPPTCPConnection connection =
  new XMPPTCPConnection("sensor42", "pass", "example.org");
connection.connect().login();
```

Smack

Code example

```
XMPPTCPConnection connection =
  new XMPPTCPConnection("sensor42", "pass", "example.org");
connection.connect().login();
```

...

```
Message message = new Message("datasink@foocorp.org");
message.addPacketExtension(mydata);
connection.sendPacket(message);
```

Smack

Code example

```
XMPPTCPConnection connection =
  new XMPPTCPConnection("sensor42", "pass", "example.org");
connection.connect().login();
```

…

```
Message message = new Message("datasink@foocorp.org");
message.addPacketExtension(mydata);
connection.sendPacket(message);
```

…

```
PacketFilter filter = new AndFilter(MessageTypeFilter.NORMAL,
   new PacketExtensionFilter("data", "http://foocorp.com"));
connection.addAsyncPacketListener(new PacketListener() {
  public void processPacket(Packet stanza) {  ...  }
}, filter);
```

Smack on Android

Use cases:

- A chat app (e.g. for your community)
- Push target
- Status monitoring
- Remote command execution

Smack on Android

Use cases:

- A chat app (e.g. for your community)
- Push target
- Status monitoring
- Remote command execution

Previously: aSmack

aSmack was a build environment which, in order to provide a working XMPP library on Android, applied various patches on top of Smack and added another 6 open sources libraries to the mix.

Smack on Android

Use cases:

- A chat app (e.g. for your community)
- Push target
- Status monitoring
- Remote command execution

Previously: aSmack

aSmack was a build environment which, in order to provide a working XMPP library on Android, applied various patches on top of Smack and added another 6 open sources libraries to the mix.

Now: Smack 4.1

Tested by gradle to build against `android.jar` (`-bootclasspath`. This guarantees that Smack runs on Android (min. API level 8). Smack 4.1 uses APIs provided by the Android runtime where possible.

Push Service

realized using XMPP on Android?

Let's assume we want to build a push service for Android based on XMPP.

Push Service

realized using XMPP on Android?

Let's assume we want to build a push service for Android based on XMPP.

"Why not simply use GCM?"

- Not all devices come with Google Services Framework
- You may don't want to depend on Google
- Have a single push mechanism: XMPP
- XMPP Push notifications are faster [3]
- Some Push service provider don't guarantee delivery

"I've heard that XMPP is not battery friendly!"

- More on that in a few minutes

Push Service

realized using XMPP on Android?

Let's assume we want to build a push service for Android based on XMPP.

"Why not simply use GCM?"

- Not all devices come with Google Services Framework
- You may don't want to depend on Google
- Have a single push mechanism: XMPP
- XMPP Push notifications are faster [3]
- Some Push service provider don't guarantee delivery

"I've heard that XMPP is not battery friendly!"

- More on that in a few minutes

XMPP is already been used for Push Services. But what are the pitfalls?

The Smartphone challenge

Using XMPP on Android

The Smartphone challenge

Using XMPP on Android

- Resource constraint system
  - Slow processor
  - Not much memory
  - Usually on Battery
  - May enter (deep) sleep mode

The Smartphone challenge

Using XMPP on Android

- Resource constraint system
  - Slow processor
  - Not much memory
  - Usually on Battery
  - May enter (deep) sleep mode
- Data connectivity in a mobile environment
  - Changing latency
  - Sometime no connectivity at all
  - Sometimes the connectivity changes (GSM / WiFi switch)

The Smartphone challenge

Using XMPP on Android

- Resource constraint system
  - Slow processor
  - Not much memory
  - Usually on Battery
  - May enter (deep) sleep mode
- Data connectivity in a mobile environment
  - Changing latency
  - Sometime no connectivity at all
  - Sometimes the connectivity changes (GSM / WiFi switch)

Lesson learned

Some (most?) XMPP implementations, especially older ones, where **not** designed with mobile devices in mind. For example Smack 3 will drop your whole connection state after `disconnect()`.

Running on a resource constraint system

Smack design decisions

- Smack uses efficient XML Pull Parsing [5]
- No Document Object Model (DOM), no problems.
  - DOM is memory intensive
  - and hard to use efficiently
  - You can still use it if you really want/need to.
- Smack is modular, you can pick the components you need and disable the others
- Smack is designed with minimal resource consumption in mind
  - Doesn't use JABX. But you can use JABX if you want.
  - We try our best to avoid memory-leaks

Approaches for data connectivity issues

XEP-198: Stream Management (SM)

Stanza Acknowledging

- *Stream* endpoints acknowledge the receipt of stanzas
- Every endpoint keeps a counter of received stanzas

Approaches for data connectivity issues

XEP-198: Stream Management (SM)

Stanza Acknowledging

- *Stream* endpoints acknowledge the receipt of stanzas
- Every endpoint keeps a counter of received stanzas

Stream Resumption

- With help of the counters, it's possible to *resume a stream*
- The TCP connection initially used by the stream can be replaced by another one
- This is useful for example
  - during short (a few minutes) connection interruptions
  - for the GSM-WiFi switch

Approaches for data connectivity issues (cont.)

XEP-199: XMPP Ping, using Smack's `PingManager`

- Check "liveness" of XMPP connection by sending XMPP Pings
- Smack automatically sends server Pings in a configurable interval
- Server Ping will only be send if there was no stanza received within the interval

Approaches for data connectivity issues (cont.)

XEP-199: XMPP Ping, using Smack's `PingManager`

- Check "liveness" of XMPP connection by sending XMPP Pings
- Smack automatically sends server Pings in a configurable interval
- Server Ping will only be send if there was no stanza received within the interval

Android

Use Smack's `ServerPingWithAlaramManager` to reliable schedule server pings on Android.

Approaches for data connectivity issues (cont.)

XEP-199: XMPP Ping, using Smack's `PingManager`

- Check "liveness" of XMPP connection by sending XMPP Pings
- Smack automatically sends server Pings in a configurable interval
- Server Ping will only be send if there was no stanza received within the interval

Android

Use Smack's `ServerPingWithAlaramManager` to reliable schedule server pings on Android.

If the connection silently breaks, i.e. no `SIGPIPE`, then there is nothing you can do to detect that besides draining the battery by increasing the ping interval.

About XMPP's battery consumption

- Sending and receiving data involves power consumption
- If the mobile device sends a stanza it usually has a good reason
- It's the receiving side you have to take care of

About XMPP's battery consumption

- Sending and receiving data involves power consumption
- If the mobile device sends a stanza it usually has a good reason
- It's the receiving side you have to take care of

Solution

Distinguish between incoming stanzas that

1. require immediate delivery
2. can be delivered later
3. should not be delivered at all

About XMPP's battery consumption

- Sending and receiving data involves power consumption
- If the mobile device sends a stanza it usually has a good reason
- It's the receiving side you have to take care of

Solution

Distinguish between incoming stanzas that

1. require immediate delivery
2. can be delivered later
3. should not be delivered at all

Typical examples:

1. (Certain) Message stanzas
2. Presence stanzas if the user is inactive (next)
3. Stanzas send by an malicious entity (slide after next)

About XMPP's battery consumption (cont.)

Incoming presence stanzas are often the cause of unnecessary power consumption.

About XMPP's battery consumption (cont.)

Incoming presence stanzas are often the cause of unnecessary power consumption.

- No presence information required if the user isn't looking at the roster
- Idea: Delay presence delivery until user is active
- XEP-352: Client State Indication

Further techniques to decrease power consumption

- Avoid network I/O by using XEP-115: Entity Capabilities
- Minimize data size (as recommend by XEP-286: XMPP on Mobile Devices)
- Use compression (XEP-138: Stream Compression)
  - **Warning:** Using compression opens an attack vector (cf. CRIME/BEAST attacks) [1]

About XMPP's battery consumption

Preventing malicious users from stealing your battery charge

A malicious entity (user) could drain the victims battery if it knows

- your bare JID, and the only connected resource is the mobile client
- your full JID

by sending stanzas to the victims mobile device.

About XMPP's battery consumption

Preventing malicious users from stealing your battery charge

A malicious entity (user) could drain the victims battery if it knows

- your bare JID, and the only connected resource is the mobile client
- your full JID

by sending stanzas to the victims mobile device.

Possible solution:
XEP-16: Privacy Lists

- Enables server-side blocking of stanzas
- Create a list that
  1. Allows stanzas from JIDs that are **subscribed to your presence**
  2. Allows stanzas from your XMPP service
     - otherwise you may just locked yourself out of the service
  3. Blocks everything else

Using Smack's XMPPTCPConnection on Android

- Create an `android.app.Service` which holds the reference to and manages your `XMPPTCPConnection`
- Model the service as Finite-State Machine, with those states:
  - Disconnected
  - Connecting
  - Connected
  - Disconnecting
  - WaitingForNetwork
  - WaitingForRetry
- Register `BroadcastReceiver` for `android.net.conn.CONNECTIVITY_CHANGE`
  - Check in receiver if the data connectivity really changed
  - If so, call `XMPPTCPConnection.instantShutdown()` followed by `connect()` to re-establish (and possible resume) XMPP stream

XMPP Login Duration

XMPP Login takes to long.
Number with 80ms round-trip

Phase	Time
TCP connect incl. DNS	60ms
Client-Server Initial Stream	80ms
TLS RFC 6120 § 9.1.1	420ms
SASL RFC 6120 § 9.1.2	470ms
Compression XEP-138	160ms
Stream Management XEP-198	190ms
Roster retrieval using versioning	80ms
Privacy List already set	80ms
Total (Real)	1750ms
Total (Sum. Parts)	1540ms

XMPP Login Duration

XMPP Login takes to long.
Number with 80ms round-trip

- Could use XEP-305: XMPP Quickstart
- Not supported by Smack and still not enough
- Should be possible to resume stream in under 200ms
- Work in progress

Phase	Time
TCP connect incl. DNS	60ms
Client-Server Initial Stream	80ms
TLS RFC 6120 § 9.1.1	420ms
SASL RFC 6120 § 9.1.2	470ms
Compression XEP-138	160ms
Stream Management XEP-198	190ms
Roster retrieval using versioning	80ms
Privacy List already set	80ms
Total (Real)	1750ms
Total (Sum. Parts)	1540ms

Any help with Smack is appreciated.

- Top priority: Add support for XEP-174: Serverless messaging (XMPP via zeroconf/link-local)
  - Guardian Project's ChatSecure wants to switch to Smack 4.1
  - They need XMPP link-local support
  - ChatSecure is currently locked-in using an old version of aSmack
- More open tasks at `https://github.com/igniterealtime/Smack/wiki/Smack-Jobs`

Any help with Smack is appreciated.

- Top priority: Add support for XEP-174: Serverless messaging (XMPP via zeroconf/link-local)
  - Guardian Project's ChatSecure wants to switch to Smack 4.1
  - They need XMPP link-local support
  - ChatSecure is currently locked-in using an old version of aSmack
- More open tasks at `https://github.com/igniterealtime/Smack/wiki/Smack-Jobs`

Thanks for your attention.
Meet me in 30min at the Realtime Lounge (Building K, Level 2) if you have further questions.

References I

Thijs Alkemade. *HTTPS Attacks and XMPP 2: CRIME & BREACH*. https://blog.thijsalkema.de/me/blog//blog/2014/08/07/https-attacks-and-xmpp-2-crime-and-breach/. Aug. 7, 2014.

Mark Dowd. *BlackPwn: BlackPhone SilentText Type Confusion Vulnerability*. http://blog.azimuthsecurity.com/2015/01/blackpwn-blackphone-silenttext-type.html. Jan. 27, 2015.

Huber Flores and Satish Srirama. “Mobile Cloud Messaging Supported by XMPP Primitives”. In: *Proceeding of the Fourth ACM Workshop on Mobile Cloud Computing and Services*. MCS ’13. Taipei, Taiwan: ACM, 2013, pp. 17–24. ISBN: 978-1-4503-2072-6. DOI: 10.1145/2482981.2482983. URL: http://doi.acm.org/10.1145/2482981.2482983.

References II

- Adrian Hornsby and Rod Walsh. “From Instant Messaging to Cloud Computing, an XMPP review”. In: *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on*. IEEE, Jan. 1, 2010, pp. 1–6. URL: http://dx.doi.org/10.1109/ISCE.2010.5523293.

- Tej M V Uttam et al. “Analyzing XML Parsers Performance for Android Platform”. In: *International Journal of Computer Science and Information Technologies*. Vol. 2. India.

- Davanum Srinivas. *Android - Just Use Smack API for XMPP*. https://davanum.wordpress.com/2007/12/31/android-just-use-smack-api-for-xmpp/. Dec. 31, 2007.