Instant Messaging / Extensible Messaging and Presence Protocol
To: Wife
“It is going to be late tonight”

To: HeadOffice
“Can you send me the last business report.”

File transfer

To: EncyclopaediaBot
“Chat”

To: HeadOffice
“refers to any kind of communication over the internet, but is primarily meant to refer to direct text-based 1-on-1 or text-based group conversation”

Private contact

Mobile business partner

Messaging enabled encyclopaedia software
Outline

• Instant Messaging (IM) in general
  – Chat principle with a high degree of personalisation used in popular implementations as e.g. Skype or ICQ

• Extensible Messaging and Presence Protocol (XMPP)
  – Standardised XML based protocol for IM applications
  – Basis for the open, extensible and secure IM technology ‘Jabber’

• Overview of XMPP extensions and associated approaches
• Alternative XMPP connection approaches
• XMPP via
  – HTTP (BOSH)
  – WebSockets

The Lecture is partially based on materials from Dr. Daniel Schuster, lecture “Multimediakommunikation”
Instant Messaging

- Instant Messaging (IM) is a potential near-real-time communication based on the exchange of messages between two or more instances that have been registered to an Instant Messaging Service.
- In contrast to regular chat systems, IM has a higher degree of personalisation.
- IM Service works hand in hand with a Presence Service which is a means for storing, finding and distributing Presence Information of participating instances.
- Presence Information may be shown to a user in form of a contact list by an IM client.
- The IETF developed a model for Presence and Instance Messaging presented in RFC 2778 to define a uniform vocabulary for describing and comparing IM systems.
The Instant Messaging Service interacts with

1. Senders which send Instant Messages to a particular Instant Inbox Address by using the service for message delivery

2. Instant Inboxes which are associated with an Instant Inbox Address and accept and potentially temporary store Instant Messages from the Service

Instant Messages are distributed by an Instant Message Protocol
Presence Service

- The Presence Service communicates with
  1. Presentities which distribute Presence Information (e.g. status, network address, ...) of an entity to the Presence Service
  2. Watchers which receive Presence Information from the Presence Service – two types exist:
     - Fetchers explicitly request information in arbitrary or periodic (Poller) intervals
     - Subscribers request automatic notification from Presence Service of future changes in some Presentity’s Presence Information

- Presence Information is distributed by the Presence Protocol
Instant Messaging

- IETF model does not make any assumption about the location of the components, thus, rendering various implementation variants possible.
Instant Messaging

1. On user's initiation Client A establishes a transport layer connection to the IM server.

2. Presentity transmits the client's Presence Information to the IM server; within this step the IM Server might check if there are new messages in the associated IM Inbox and potentially deliver them.

3. Initial Fetcher requests Presence Information about relevant other instances (as e.g. private contacts of User A).

4. User A adds User B to contact list resulting in registration of the Subscriber of Client A to the Presence Service for status updates of Client B.

5. Client B connects to the server, informs the Presence Service that it is online now and receives status information about relevant contacts.

6. Subscriber of Client A is informed about the status update of Client B.

7. Client A sends a message to the Instant Inbox Address of Client B.

8. The Instant Message Service makes use of the Presence Service to look up if Client B is online at the moment and if this is the case it determines the client's current network address.

9. Because Client B is online the message is not stored on the server but directly forwarded to Client B that presents the message to User B.
• The Extensible Messaging and Presence Protocol is an open XML protocol for near-real-time messaging, presence and request-response services that can be used especially for IM systems
• Defined in RFCs 6120 to 6121 and extended by so called XMPP Extension Protocols (XEP)
• Main design goals were:
  – Decentralisation
  – Interoperability with further IM protocols
  – Simple client implementations
  – Security
  – Extensibility
XMPP

- Instant Inbox Address is called Jabber Identifier (JID)
  \[ \text{JID} = [ \text{user } "@" ] \text{ domain } [ "/" \text{ resource }] \]

- After registration to a XMPP (Jabber) server, a JID is assigned to a user that has the form user@server.com

- When a client connects to a server it binds a resource identifier to itself that has the form user@server.com/resource

- By this concept it is possible to be available via different user client devices though only having one user account

Client on mobile phone
JID: vincent@example.com/mobile

Connected to

Jabber server
JID: example.com

Client on Personal Computer
JID: vincent@example.com/pc
XMPP

- Architecture of systems using XMPP for communication are heavily server centric

- XMPP is based on two fundamental communication concepts:
  1. XML Streams
     - Container for exchange of XML elements between two instances
  2. XML Stanzas
     - XML elements with special semantics that exist at the child level of streams

Server stores contacts lists (XMPP term: roster), presence information etc. of participants
XML Streams

- The communication via XMPP can be compared to the exchange of two XML documents between client and server – one for each direction.
- After establishing a TCP connection and initial steps for authentication and encryption, a stream is opened that functions as envelope during complete communication and, thus, is closed when the client disconnects.

![Diagram showing XMPP communication](image-url)
XML Stanzas

- There exist three types of XML Stanzas that provide the building blocks for communication in an IM system:
  1. **message** – encapsulates chat data
     <message to="vincent@example.com" type="chat">
     <body>Hi Vincent!</body>
     </message>
  2. **presence** – exchanges presence information and manages subscription to Presence Information
     <presence>
     <show>dnd</show>
     <status>I have to work…</status>
     </presence>
     <presence to="vincent@example.com" type="subscribe"/>
     "do not disturb"
     Request for subscription
3. iq – requests or sets different information (e.g. requests the user's contact list (roster) from the server)

```
<iq id="roster0" type="get">
  <query xmlns="jabber:iq:roster"/>
</iq>
```

- Beside type "get" there exist types "set", "result" and "error"
- Communication with the iq element is done in a request-reply-manner
XMPP Messaging - details

```
<message from="alice@a.de/phone" to="bob@b.de" type="chat">
  <body>How are you?</body>
  <subject>Query</subject>
</message>
```

- Messages are not confirmed by receiver ("fire and forget" semantics)
- Several message types (attribute "type") available:
  - normal (usual messages), chat (messages within an IM session), groupchat (messages within chat rooms), headline (warnings and notifications), error (error messages)
Message delivery between servers (federated)

- XMPP server associated to a domain may be determined via DNS using SRV resource records
- Server negotiates a XMPP stream with the remote server and routes received stanzas for this server via the established stream
XMPP connection

Communication details:
1. Establishing a TCP connection
2. Open a XML Stream from client to server
3. Open a XML Stream from server to client
4. Authentication of instances (based on SASL – see RFC 4422) and optionally start encryption (based on Transport Layer Security – see RFC 5246)
5. Establishing of a new authenticated and potential encrypted stream for each direction
6. Register a resource at the server, thus, making it available through an address of the scheme node@domain.com/resource
7. Establishing a session on the server to finally activate instant messaging and presence functionality
8. Client requests roster items that are delivered by the server
9. Exchange of further XML Stanzas (e.g. messages that were stored by the server while the user has been offline)
10. Close XML Stream
XMPP connection - details

6. Registration of a resource

```
<iq type="set" id="bind1">  
<bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">  
<resource>pc</resource>  
</bind>  
</iq>
```

```
<iq id="bind1" type="result">  
<bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">  
<jid>bob@b.de/pc</jid>  
</bind>  
</iq>
```

7. Establishing a session

```
<iq type="set" id="session1">  
<session xmlns="urn:ietf:params:xml:ns:xmpp-session"/>  
</iq>
```

```
<iq type="result" id="session1"/>
```
8. Delivery of roster items

```xml
<iq from="bob@b.de/pc" type="get" id="roster1">
  <query xmlns="jabber:iq:roster"/>
</iq>

<iq to="bob@b.de/pc" type="result" id="roster1">
  <query xmlns="jabber:iq:roster">
    <item jid="elena@example.com" name="Elena" subscription="from">
      <group>Friends</group>
    </item>
    <item jid="marsellus@example.com" name="Boss" subscription="to">
      <group>Work</group>
    </item>
    <item jid="vincent@example.net" name="Vincent" subscription="both">
      <group>Work</group>
    </item>
  </query>
</iq>
```
XMPP subscription management

- XMPP can make the observed presentity's confirmation of subscription necessary
- During the subscription process all connected resources of involved users are informed about the progress
Instant Messaging

1. After Jules added Vincent to his contact list by giving his JID and a name that is presented in the user interface the roster data is updated by an iq-set operation.

2. Initiating client receives confirmation of a roster update and all connected resources are informed about this new roster item.

3. User client sends a presence-stanza (type ‘subscribe’, addressed to Vincent’s JID).

4. Server 1 informs connected resources of jules@abc.com about pending subscription state (which may be visualised by the user clients).

5. The presence-stanza is routed to the new contact's accountable server which forwards it to the available resource.

6. After Vincent has confirmed the subscription, the user client updates its roster information by sending an iq-stanza followed by a presence-stanza (type ‘subscribed’ and attribute ‘to’ has value ‘jules@abc.com’) to confirm subscription.

7. After pushing a roster update to all connected resources of vincent@example.com Server 2 confirms the update to the sending resource.

8. Server 2 forwards the presence-stanza of type ‘subscribed’ to Server 1 and sends presence information of all connected resources of vincent@xyz.com to the address jules@abc.com.

9. The subscription confirmation is forwarded to the user client.

10. Connected resources of Jules are informed by a pushed iq-set operation about the new subscription state and receive the presence information of all connected resources of vincent@xyz.com.
Distribution of presence information

- Server receives presence information from all connected devices and distributes this information

XMPP Client

XMPP Server

XMPP Client

Own presence is transmitted to registered resources

 alice@a.de/phone

andrew@a.de/pc

<presence/>
<presence from="alice@a.de/phone" to="andrew@a.de"/>
<presence from="andrew@a.de/phone" to="alice@a.de"/>
<presence/>
<presence from="andrew@a.de/pc" to="andrew@a.de"/>
• On behalf of its clients, a server may request presence information about clients connected to other servers ("probe")
• XMPP extensions are defined by the XMPP Standards Foundation as XMPP Extension Protocols (XEPs)
Besides the basic Client-Server and Server-to-Server communication, several alternative approaches exist, such as:

- Jabber component protocol (XEP-0114) e.g. for accessing services via XMPP
- Serverless Messaging (XEP-0174) for direct XMPP client interaction e.g. in ad-hoc networks
- XMPP communication via HTTP (BOSH) or WebSockets
Problem:
- (continuous) TCP connection may be inappropriate in several use cases, e.g.:
  - mobile networks:
    - Connection interruptions lead to XMPP reconnects
  - mobile devices:
    - Connection interruptions occur in case an application is suspended in order to save power
    - restrictive firewall configuration

Solution:
- HTTP long-polling:
  - BOSH: Bidirectional Streams over Synchronous HTTP (XEP-0124)
BOSH: Connection Establishment

**XMPP Client**

HTTP "refresh" every 60 seconds

**POST /webclient HTTP/1.1**
Host: bosh.a.de

```xml
<body secure="true" rid="90029201" to="a.de" ...
wait="60" xmlns="urn:xmpp:xmpp"/>
```

HTTP/1.1 200 OK

```xml
<body requests="2" sid="3m1ts1htd1s" ...
<stream:features> .... </stream:features>
</body>
```

**BOSH Connection Manager**

maximum of 2 requests at once

... session ID used for associating messages to session
BOSH Communication

**Client:**

POST /webclient HTTP/1.1
Host: bosh.wonderland.lit
Content-Type: text/xml; charset=utf-8
Content-Length: 205

```xml
<body rid="90029205" sid="3m1ts1htd1s"
xmlns="http://jabber.org/protocol/httpbind">
  <message to="sister@realworld.lit"
xmlns="jabber:client">
    <body>Help, I fell down the rabbit hole!</body>
  </message>
</body>
```

**Server:**

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 163

```xml
<body sid="3m1ts1htd1s"
xmlns="http://jabber.org/protocol/httpbind">
  <message from="sister@realworld.lit/home">
    <body>Oh! How can I help?</body>
  </message>
</body>
```
BOSH: Long-polling

- BOSH leverages HTTP 1.1 connection keep-alive and pipelining mechanism
XMPP via WebSockets

- Alternative to long-polling approach: WebSockets
- Defined in RFC 7395

GET /xmpp-WebSocket HTTP/1.1
Host: xmpp.a.de
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBzb25jZQ==
Origin: http://xmpp.a.de
...
Sec-WebSocket-Protocol: xmpp
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
...
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: xmpp
XMPP via WebSockets

[WSFH] = WebSocket Frame
Header 2-14 Byte

XMPP Server
+ WebSockets module

alice@a.de/web

[WSFH]<open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
           to="xmpp.a.de"
           version="1.0" />

[WSFH]<open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
           from="xmpp.a.de"
           id="++TR84Sm6A3hnt3Q065SnAbbk3Y="
           xml:lang="en"
           version="1.0" />

[WSFH]<message xmlns="jabber:client"
                to="andrew@a.de" xml:lang="en">
  <body>Every WebSocket message is parsable by itself.</body>
</message>

<open> (and <close>) instead of <stream> and </stream>

Namespace is embedded into each message

For each stanza one WebSocket message is used.
**XMPP via WebSockets**

- Comparison of pure XMPP, XMPP via BOSH and XMPP via WebSockets:

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>BOSH</th>
<th>WebSockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead</td>
<td>none</td>
<td>huge</td>
<td>low</td>
</tr>
<tr>
<td>Reaction on suspend</td>
<td>Lost connection</td>
<td>Connection available</td>
<td>Lost connection</td>
</tr>
<tr>
<td>Compression</td>
<td>Stream Compression</td>
<td>HTTP Compression</td>
<td>WebSocket Compression</td>
</tr>
<tr>
<td>Interoperability with firewalls</td>
<td>medium</td>
<td>very good</td>
<td>good</td>
</tr>
</tbody>
</table>
Instant Messenger

- In order to communicate to instances that are available in different IM networks (e.g. Jabber, ICQ etc.) there exist two possibilities:

  1. Multiple protocol client
     - The user client manages different accounts for each IM network, thus, supporting more than one protocol
  2. Protocol translation
     - IM network deploys gateways for translation between protocols
     - XMPP transports is one practically used gateway mechanism (see XMPP Extension Protocol 100)

![Diagram of Instant Messenger](image-url)
Conclusion

XMPP Client

XMPP Client

XMPP Client

Presence Service

HTTP (BOSH)

Component

XMPP Server a.de

BOSH Connection Manager

WebSocket Module

WebSocket + Stream Management

XMPP Server b.de

User/Message/... data

pokergame.a.de

alice@a.de/web

carol@local/pc

bob@b.de/pc

Serverless Messaging
References

RFCs


Further links

- XMPP home: http://www.xmpp.org
- XMPP extensions: https://xmpp.org/extensions
- Pidgin (XMPP client): http://www.pidgin.im/
- Gajim (XMPP client): http://www.gajim.org/