Internet Engineering Task Force	P5. , Ed.
Internet-Draft	Karlsruhe Institute of
Intended status: Informational	Technology
Expires: May 4, 2012	November 2011

TOC

Instant Messenger System
draft-kit-instantmessenger-01

Abstract

Specification for an Instant Messenger System.

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of Section 3 of RFC 3667. By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she become aware will be disclosed, in accordance with RFC 3668.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”

This Internet-Draft will expire on May 4, 2012.

Table of Contents

TOC

1. Introduction

TOC

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119** [RFC2119].

---

2. Server Client Communication

TOC

This section describes the communication between server and client implementations. The whole communication must be transported by TCP. A server communicates via a port between port 49152 and port 65535 and must annouce this port in his PeerPDU (SERVER-OPTION). A client must choose a new port for every channel he wants joining (i.e. one TCP connection per channel).

2.1. Communication Commands

TOC

There are no real PDUs but ASCII formated commands which are transported by a TCP stream. Every command must begin with a 32 bit block which represents the length of the real command including all parameters. Every command and parameter is null terminated.

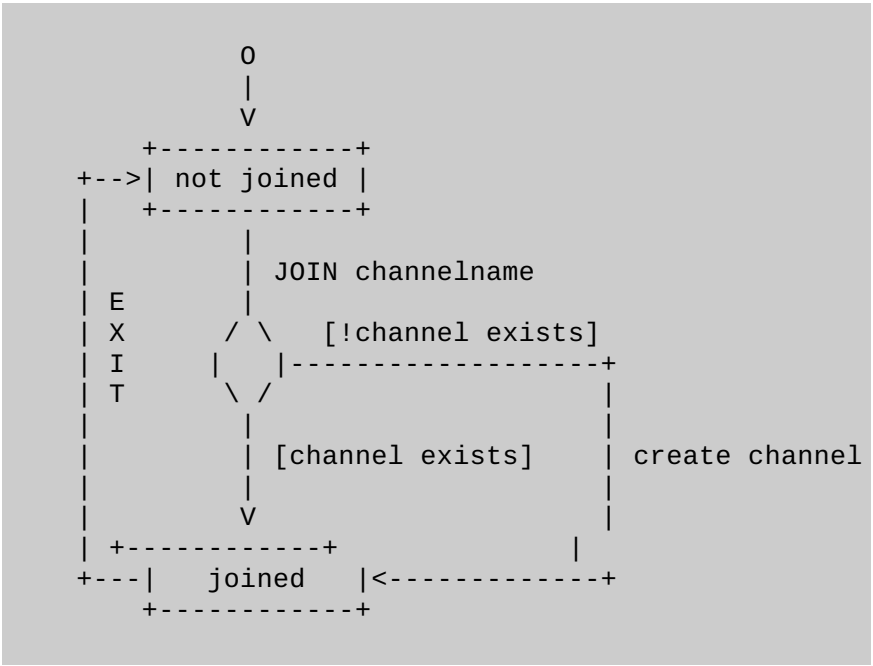
JOIN	channelName

Table 1: JOIN (Client -> Server)

To join a channel a client must send the JOIN command to the corresponding server as in table **1** shown. The first and only parameter is the name of the channel. It must be encoded in ASCII. The name of public channels start with the char '#', private ones with '@' (compare). If no channel with the given name exists, the server must create one.

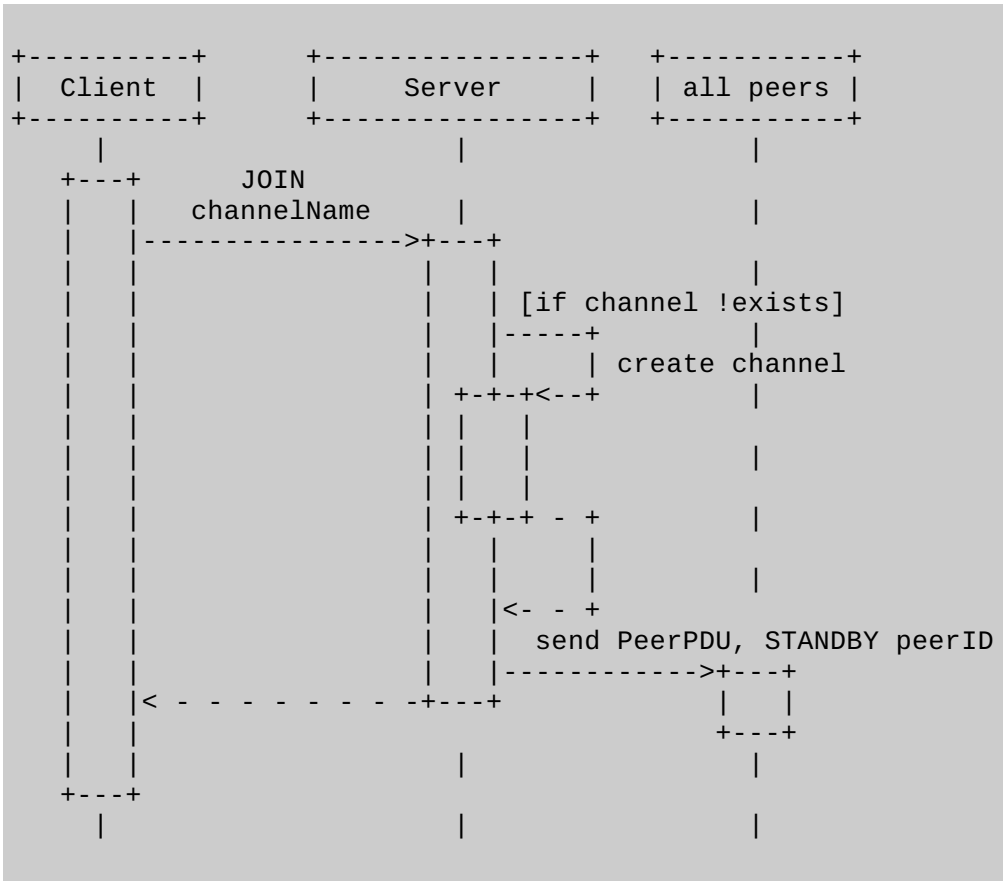
After performing this command succefully, the server involved must send an updated PeerPDU via multicast if this command created a new channel. The client involved must send an updated PeerPDU anyway. The updated PeerPDU includes the created/joined channel.

If a client has already joined a channel, every new join by the same ClientID to this specific channel should be ignored by the server.



Channel JOIN states

Figure 1



Sequence of interaction between client, server and other channelmembers for JOIN.

Figure 2

SAY	text

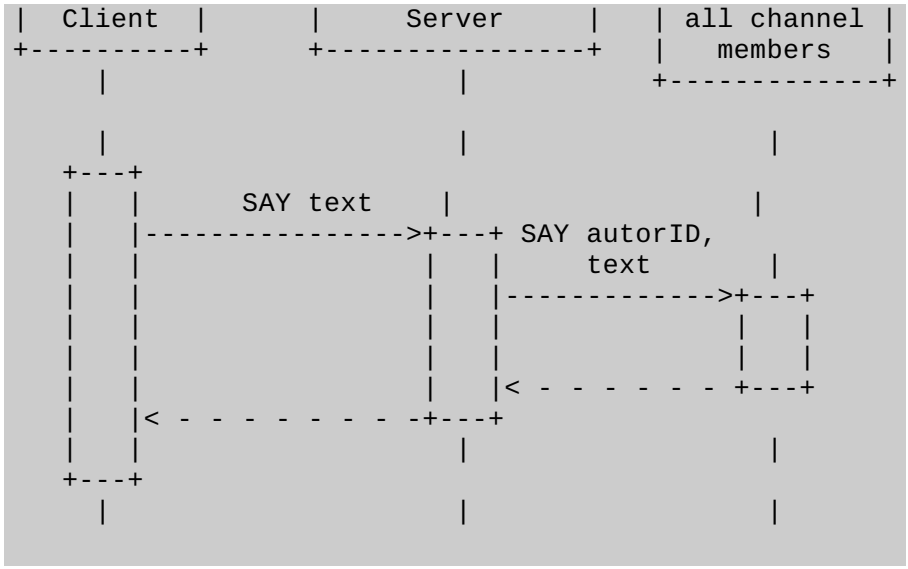
Table 2: SAY (Client -> Server)

When a user wants to send a message the client must send the SAY command to the corresponding server via the existing TCP connection. The SAY command is shown in table **2**. The passed parameter (text) must be the message the user sent. It shall be encoded in UTF-8.

SAY	authorId	text

Table 3: SAY (Server -> Client)

After receiving the SAY command specified above in table **3**, a server must also send a SAY command to all members of the corresponding channel via the existing TCP connection. The first parameter (authorId) must be the id of the client which performed the SAY command. Finally the last parameter shall be the text the user wrote.



Sequence of interaction between client, server and other channelmembers for SAY.

Figure 3

STANDBY	PeerId

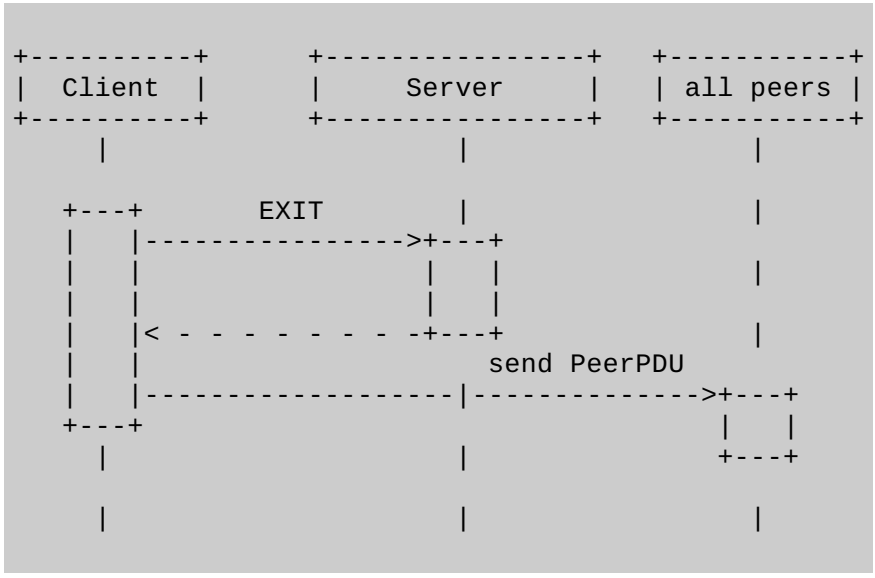
Table 4: STANDBY (Client -> Server)

If possible, a server implementation has to choose a channel member as standby peer in case of failovers as in **2.3** described. The chosen peer must be announced by the STANDBY command as in table **4** shown. This command must be send after every JOIN of a new channel member and if the chosen standbyPeer chances (e.g. if the standby peer leaves the channel).

EXIT

Table 5: EXIT (Client -> Server)

To leave a channel regular a client should send the EXIT command to the channel server via the existing TCP connection before closing it. The EXIT command is shown in table **5**, no additional parameters are required. It is explicitely not defined what happens if the last member of a channel sends EXIT.



Sequence of interaction between client, server and other channelmembers for EXIT.

Figure 4

Immediately after performing this command, a client has to send an updated PeerPDU. If there isn't any channelmembership by client left the TCP connection should be closed by the involved server.

INVITE	clientId1	...	clientIdn

Table 6: INVITE (Client -> Server)

This must be sent from a client to a server when a user wants to invite others to a channel. It can have numerous parameters but it must have one at absolute minimum.

A server must multicast a PeerPDU with the SERVER-INVITE-OPTION in it.

See for further details.

2.2. Runtime requirements

TOC

This section specifies all data requirements for a peer in general, a client and a server implementation. All peer data requirements must be stored by server and client implemenation as well. Any peer implementation must store following information to interact correctly during communication flow:

- All Interfaces which are used by the peer implementation.
- TCP socket connection information consisting of an ip number and a port number of own interfaces.
- discovery information about each peer. This information set depends in detail on client or server peerPDU as in <todo> specified.
- Each peer discovery information has a time to life (TTL). This Timer information must be saved for storing only current information about other peers. Dead or unreachable peers can be detected with this information. See section **2.4** for futher information.

A client implementation must store following additional information:

- For every channel in the network: channelName, memberlist and serverId.

A server implementation must store following additional information:

- The public channels information set is structured as in table **7** specified. In case of a failover situation the peer with standbyId may provide a new channel. See Section **2.3** for details. The member list consiting of peer ids of every joined peer.
- The private channels information. In general same as public channel information expect that this information will NEVER be broadcasted with by peerPDU.

channelName	standbyId	member list

Table 7: The channel information

For sending a history of channel conversation a server implementation may store passed messages. This messages may send to a new connected peer.

2.3. Standby and Failover

TOC

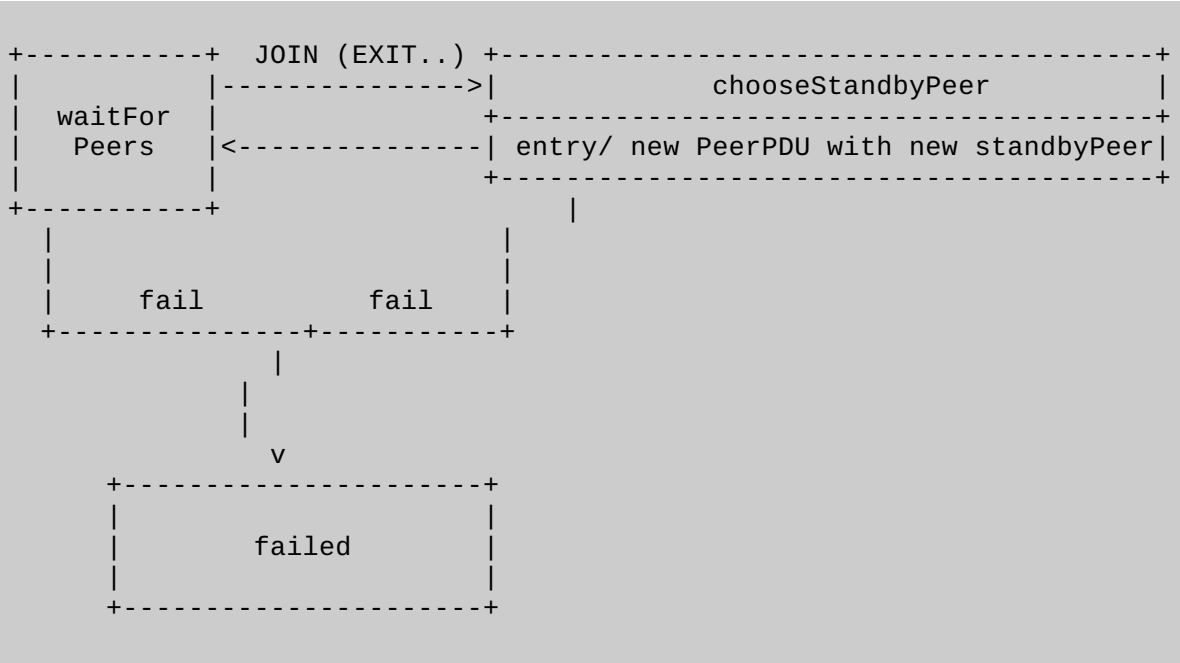
2.3.1. Standby-Mechanism

TOC

The channel server waits for the commands JOIN or EXIT.

If a new user joins a channel the server must choose a new standby peer which resumes the server function when the current channel host fails. The algorithm for setting a new standby peer must be chosen by the programmers (For example the latency, the amount of running channels, the amount of joined channels and many more). When the server has chosen a new standby peer it sends a STANDBY command to all channelmembers with the current standbyId. So every peer knows which peers resumes the server after a failover.

If a peer exits the channel the server has to check if the leaving peer was the old standby peer. If it was the standby peer the server has to choose a new one and publish this via the STANDBY command.



Standby-Mechanism

Figure 5

2.3.2. Failover

TOC

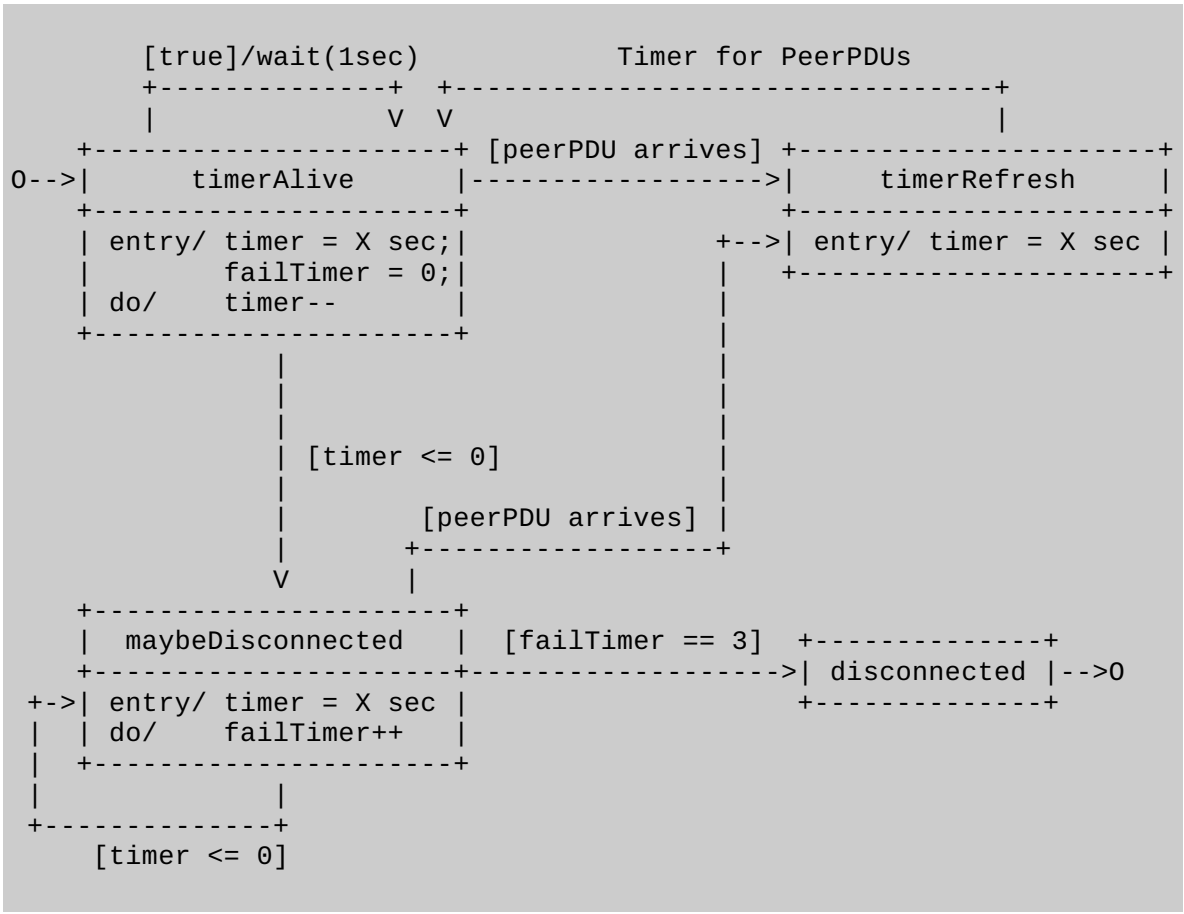
The server isn't reachable anymore and after the peer waited three times for a new peerPDU it checks the latest standby peer stored in the **Table 7**. Now the peer must automatically connect via a JOIN to the standby peer. The channel id and name remain constant after the fail. If the standby mechanism fails there is no other mechanism to catch the error and the channel is lost.

2.4. peerPDU Timer

TOC

To check if a peer failed every peer holds a timer for every other discoverd peer in the network. If a new peerPDU arrives from the corresponding peer the timer must be updated. If the peerPDU doesn't arrive until timer <= 0 the peer state must be set to 'maybeDisconnected'. During three runs a new peerPDU should arrive to get back to the

'timerRefresh' state. If no peerPDU arrives the peer is set to 'disconnected' and may be deleted from the peer list.



peerPDU timeout timer

Figure 6

3. Security Considerations

TOC

4. References

TOC

4.1. Normative References

TOC

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

[min_ref] authSurName, authInitials., "Minimal Reference," 2006.

4.2. Informative References

TOC

[DOMINATION] Mad Dominators, Inc., "Ultimate Plan for Taking Over the World," 1984.

[I-D.narten-iana-considerations-rfc2434bis] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008 (TXT).

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML," RFC 2629, June 1999 (TXT, HTML, XML).

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations," BCP 72, RFC 3552, July 2003 (TXT).

TOC

Appendix A. Additional Stuff

This becomes an Appendix.

TOC

Author's Address

Projektgruppe 5 (editor)
Karlsruhe Institute of Technology
Karlsruhe, Baden-WÃ¼rttemberg
DE

Phone:
Email: tm-projekt5@ira.uka.de

TOC

Full Copyright Statement

Intellectual Property