Polite Messaging : Version 1.0

March 9, 2021

Polite Messaging (PM) is a peer-to-peer, asynchronous messaging system in which peers store and exchange messages. This specification describes the protocol used to communicate between nodes. The terminology of RFC 2119 is used : MUST, SHOULD and MAY.

1 Terminology

PM uses a number of basic building blocks:

Peer One of the two participants in the protocol.

Time Time is given as an integer which is the number of seconds since the start of the UNIX Epoch.

String Up to 65535 bytes. Unless otherwise specified this is should be interpreted as text encoded using UTF-8.

Line A line is a string where the last character that is new line (`\n`).

Parts If a line is described as having several parts then is a single space character between the parts.

2 Transport

PM runs over TCP. Implementations MAY be a TCP client or TCP server or both. Once the connection is established, it does not matter which is which. Implementations SHOULD use TCP port 20111.

3 Message Format

The main purpose of PM is to store and exchange messages. Each message consists of four or more lines. These lines are interpreted as *headers* until the `Contents` header has been found. All lines after that are interpreted as *body*. The following headers MUST appear in the message:

`Message-id: SHA-256` *hash*
This MUST be the first header. The hash MUST be the SHA-256 sum of the rest of the headers and the body of the message.

`Time-sent:` *time*
The *time* is when the message was created.

`From:` *person*
This identifies who sent the message. *person* MAY be an e-mail address.

`Contents:` *number*
This MUST be the last header. The *number* gives how many lines of the body follow.

The following headers MAY appear in the message:

`To:` *person*
This identifies who the message is going to. *person* MAY be an e-mail address.

`Subject:` *subject*
This gives the what the message is about.

`Topic:` *topic*
This gives the general topic of the message. *topic* MAY be a hash-tag.

Implementations MUST allow other undocumented headers to allow for future versions of the protocol. Undocumented headers SHOULD be ignored.

4 Example Message

This is an example of a correctly formatted message:

```
Message-id: SHA-256 bc18ecb5316e029af586fdec9fd533f413b16652bafe079b23e021a6d8ed69aa
Time-sent: 1614686400
From: martin.brain@city.ac.uk
Topic: #announcements
Subject: Hello!
Contents: 2
Hello everyone!
This is the first message sent using PM.
```

5 Storage

The intent of PM is that messages flood across the network so that they can reach all participants and so that the network will still operate even if some nodes are unavailable.

Implementations SHOULD store the messages that they can and SHOULD make them available to others. They SHOULD store them while the program is running and MAY store them longer for example using files or a database. Implementations or the people operating them MAY choose to delete messages. They SHOULD use the following criteria:

- The age of the message given by `Time-sent`.
- The `Topic` or `Subject` of the message.
- The `Contents` of the message.

Implementations MUST store the initial example message given above and MUST give it when requested.

6 Requests and Reponses

Communication in PM consists of a number of *requests* each of which may have a *response*. Either peer MAY make as many requests as they like. Both peers MUST respond to all requests they receive (if there is a response specified). Both peers MUST respond to requests while waiting for a response to avoid deadlocking if both make a request simultaneously. If a peer receives an invalid, unknown, incorrect, corrupt or out of order request it SHOULD end the interaction with that peer and close the socket.

6.1 PROTOCOL?

Both peers MUST send a protocol request as the first request they send. A protocol request is a single line with three parts:

`PROTOCOL?` *version identifier*

Here *version* is a positive integer giving the protocol version. This is the specification of version 1. Implementations MUST accept higher version numbers. However the rest of the communication should only use features that are in the *highest common protocol version*. *identifier* is a string that identifies the peer. Implementations SHOULD use something that allows other users to identify the owner of the system.

There is no response to a protocol request.

6.2 TIME?

A time request is a single line:

`TIME?`

The response is a single line with two parts:

`NOW` *time*

time is the current time at the peer. Peers SHOULD make sure their time is accurate.

6.3 LIST?

A list request is one or more lines. The first line has three parts:

`LIST?` *since headers*

Here *since* is any time in the past. Peers MUST NOT request times that are in the future. *headers* is an integer which MUST be 0 or more. This gives the number of following lines which contain headers (see section 3).
The response is one or more lines. The first line has two parts:

`MESSAGES` *count*

The responding peer finds every message it has stored with:

1. A `Time-sent` header that is greater than or equal to *since*.
2. All of the headers that are given in the request

count is the number of messages that it has found. It then outputs the hash from the `Message-id` header of each of the messages.

6.4 GET?

A get request is one line. It has three parts:

`GET? SHA-256` *hash*
Here *hash* must be an SHA-256 sum.

There are two possible responses. A peer can respond with a single line:

`SORRY`

or it can respond with multiple lines, the first is:

`FOUND`

then it must send the message with the requested message ID.

6.5 BYE!

A bye request is a single line:

`BYE!`

There is no response but both peers must close the socket.

7 Full Example of PM

This is a full example of two peers communicating using PM. The writing on the left hand side is what one peer sends and the writing on the right is the other peer.

They first agree on a protocol version. Then the peer on the left checks the time and asks for what new messages there are on the topic `#announcements`. Finally they request the body of the message and then say goodbye.

```
PROTOCOL? 1 Left
                                PROTOCOL? 1 Right
TIME?
                                NOW 1614690000
LIST? 1614680000 1
Topic: #announcements
                                MESSAGES 1
                                bc18ecb5316e029af586fdec9fd533f413b16652bafe079b23e021a6d8ed69aa
GET? bc18ecb5316e029af586fdec9fd533f413b16652bafe079b23e021a6d8ed69aa
                                FOUND
                                Message-id: SHA-256 bc18ecb5316e029af586fdec9fd533f413b16652bafe079b23
                                Time-sent: 1614686400
                                From: martin.brain@city.ac.uk
                                Topic: #announcements
                                Subject: Hello!
                                Contents: 2
                                Hello everyone!
                                This is the first message sent using PM.
BYE!
```