Contents

1 Introduction

This document describes the current TPC (Teams Public Chat) protocol. The TPC protocol has been developed for use on systems using the TCP/IP network protocol, but there is no reason why this can be implemented for other network stacks.

TPC itself is a collaborative communication application able to manage several communication teams, where discussions are organized in the following format:

- threads (initial post and additional comments) in a specific channel
- discussion (personal messages)

Through the use of a client-server model, this protocol is adapted to for use on multiple machines. The intended implementation is for a single server multi-client use case.

1.1 Servers

The server is the backbone of TPC, providing a point to which clients may connect to communicate with each other. The only intended network configuration for communication between TPC servers and clients is that of a star topology [see Fig. 1] where the server acts as a central distribution point for all communication to and from clients.

```
      [Client]                [Client]                 [Client]
              \                   |                   /
               \                  |                  /
                \___________      |      ___________/
                            \     |     /
                             \    |    /
                              \   |   /
      [Client]----------------[Server]-----------------[Client]
                              /   |   \
                             /    |    \
                 ___________/     |     \___________
                /                 |                 \
               /                  |                  \
              /                   |                   \
      [Client]                [Client]                 [Client]
```

[Fig. 1. Format of TPC network]

1.2 Clients

A client is anything that tries to connect to the server. Each client must possess a *user_name,* having a maximum length of thirty two(32) characters, in order to be considered as logged in.

Each *user_name* is also linked to a *user_uuid* allowing for user differentiation. See the protocol grammar rules for what may and may not be used in a *username*.

1.3 Teams

A team is a named group of one or more channels. A team is created explicitly by a user with the command in an undefined context. If the team exists, any client can reference the channel using an appropriate syntax.

Team names are strings of a length of up to 32 characters. Apart from the length restriction, the only restriction on a *team_name* is that it must be surrounded with quotation marks. Each team name is also linked to a *team_uuid* allowing for team differentiation.

Upon creation teams should also be assigned a team description. These have the same restrictions as the team_name except that the length is limited to 255 characters.

To become part of an existing team, a user is required to '/subscribe' to the channel. If the team does not exist prior to the subscription trial, the subscription shall always fail. If the channel already exists, a subscription requests should never fail unless a server system error has been encountered.

As part of the protocol, a user may be a subscribed to several teams at once, but a limit of five (5) teams is recommended as being ample for both experienced and novice users.

1.4 Channels

A channel is a named group of one or more threads. A channel is created explicitly by a user, in a defined team context. If the channel exists, any client subscribed to the relevant team and within the relevant team context can reference the channel by using the appropriate syntax.

Channel names are strings of a length of up to 32 characters. Apart from the length requirement, the only restriction on a *channel_name* is that it must be surrounded with quotation marks. Each *channel_name* is also linked to a *channel_uuid* allowing for channel differentiation.

Upon creation a channel must also be assigned a channel description. These have the same restrictions as the channel_name except that the length is limited to 255 characters.

To create a new channel a user is required to '/create' the channel within a defined team context.

To switch the context to an existing channel the user must employ the '/use' command within an already defined team context. If the channel does not exist prior to trying a context switch, the context switch shall always fail. If the channel already exists, a context switch should never fail unless a system error has been encountered.

1.5 Threads

A thread is a named group of one or more comments. A thread is created explicitly by a user, in the appropriate context. If the thread exists, any client subscribed to the relevant team and within the relevant context can reference the channel using an appropriate syntax.

Thread names are strings of a length of up to 32 characters. Apart from the length requirement, the only restriction on the thread_*name* is that it must be surrounded with quotation marks. Each *thread_name* is also linked to a *thread_uuid* allowing for channel differentiation.

Upon creation a thread must also be assigned an initial thread message. These have the same restrictions as the channel_name except that the length is limited to 512 characters.

To create a new thread a user is required to '/create' the thread within a defined channel context.

To switch the context to an existing thread the user must employ the '/use' command within an already defined channel context. If the thread does not exist prior to trying a context switch, the context switch shall always fail. If the thread already exists, a context switch should never fail unless a system error has been encountered.

2. The TPC Specification

The protocol as described herein is for use with only client to server connections.

2.1 Character codes

The protocol is restricted for use with the ASCII character set. The protocol is based on a set of codes which are composed of eight (8) bits, making up an octet. Each message may be composed of any number of these octets; however, some octet values are used for control codes which act as message delimiters.

2.2 Messages

Servers and clients send each other messages which may or may not generate a reply. If the message contains a valid command, as described in later sections, the client should expect a reply as specified but it is not advised to wait forever for the reply; client to server communication is essentially asynchronous in nature. Each TPC message may consist of up to two main parts: the command, and the command arguments (of which there may be up to 3). The command, and all its arguments are separated by one (or more) ASCII whitespace character(s) (0x20 or 0x09).

The command **must** be a valid TPC command. No support for translation between TPC commands and specific number codes is supported.

TPC messages are always lines of characters terminated with a CR-LF (Carriage Return - Line Feed) pair. See later sections for more details about current implementations.

2.3 Numeric replies

Most of the messages sent to the server generate a reply of some sort. All replies are numeric and are used for both errors and normal replies. The numeric reply must be sent as one message consisting of, the three-digit numeric and the response message. A numeric reply is not allowed to originate from a client; any such messages received by a server are dropped for reasons of inconsistent input. In all other respects, a numeric reply is just like a normal message, except that the keyword is made up of 3 numeric digits rather than a string of letters.

3. TPC Concepts

This section is devoted to describing the actual concepts behind the organization of the TPC protocol and how the current implementations deliver different classes of messages.

3.1 One-to-one communication

One to one communication is only instantiated by the clients. How this works is quite simple. Client 1 makes a request to the server to send a message to client 2. The server receives this demand and only delivers this message to the desired recipient. A one-to-one message should never be a broadcasted over the network as this defeats the entire purpose of one-to-one communication.

3.2 One-to-many communication

One-to-many communications can only be implemented in one fashion. That is by using comments in threads. Once a user is subscribed to a team and finds himself in the appropriate sub context, the posting of a comment in a desired thread automatically makes the text field publicly readable to anyone participating in a similar context.

4. Message details

On the following pages are descriptions of each message recognized by the TPC server and client. All commands described in this section must be implemented by any server intending on using this protocol.

The server to which a client is connected is required to parse the complete message, returning any appropriate errors. If the server encounters a fatal error while parsing a message, an error must be sent back to the client and the parsing terminated. A fatal error may be an incorrect command, a destination which is otherwise unknown to the server (team, channel, thread, channel or user uuids fit this category), not enough parameters, too many parameters or missing quotes.

If a full set of parameters is presented, then each must be checked for validity and appropriate responses sent back to the client.

Note: If a "?" is present before a parameter, this means it is optional

4.1 Connection Registration

The commands described here are used to register a connection with an TPC server as a user. User identification (/login) is not required for a connection to be registered but is required for all the other command (except /help) to be executed correctly.

4.1.1 /login message

Command: /login

Parameters: ["user_name"]

The /login message is used at the beginning of a connection to specify a user name either at the creation of a new user to connect using an existing account.

Since it is possible for the same username to be used across by several users across several machines, a user_uuid is assigned at user creation. This gives rise to a new constraint: a user name can only be used once on the same machine. This is because the user_uuid links the username to the host name.

Numeric Replies:

- ERR_NEEDMOREPARAMS
- ERR_TOOMANYPARAMS
- ERR_ALREADYCONNECTED
- REPL_LOGIN

Examples:

- /login “someuser” ;User either registering themselves or reconnecting to the user on the current host named “someuser”

4.1.2 /logout message

Command: /logout

Parameters: No parameters required

A client session is ended with a /logout message. The server must close the connection to a client which sends a /logout message. If a "Quit Message" is given, this will be sent instead of the default message, the nickname. If a client is not connected the server does nothing and does not reply. If, for some other reason, a client connection is closed without the client issuing a QUIT command (e.g. client dies and EOF occurs on socket), the server is required to fill in the quit message with some sort of message reflecting the nature of the event which caused it to happen.

Numeric Replies:

- ERR_TOOMANYPARAMS
- ERR_NOTCONNECTED
- REPL_LOGOUT

Examples:

- /logout ; Current connected user logging out.

4.2 Team operations

This group of messages is concerned with manipulating channels, their properties (channel modes), and their contents (typically clients). In implementing these, a number of race conditions are inevitable when clients at opposing ends of a network send commands which will ultimately clash. It is also required that servers keep a nickname history to ensure that wherever a <nick> parameter is given, the server check its history in case it has recently been changed.

4.2.1 /subscribe command

Command: /subscribe
Parameters: [“team_uuid”]

The /subscribe command is used by client to subscribe to the events of a team and its sub directories (enable reception of all events from a team). There are no restrictions on a user joining a team.

Once a user has joined a team, they receive notice about all commands their server receives which affect the team. This includes /subscribe, /unsubscribe and /create.

Numeric Replies:

- ERR_NEEDMOREPARAMS
- ERR_TOOMANYPARAMS
- ERR_NOSUCHTEAM
- ERR_NOTCONNECTED
- ERR_NEEDQUOTES
- RPL_SUBSCRIBE

Examples:

- /subscribe “someuuid” ; join team identified by someuuid.

4.2.2 /subscribed command

Command: /subscribed

Parameters: ?[“team_uuid”]

The /subscribed command is used to obtain a listing of all the teams a user subscribed to or in the case the team_uuid is specified it gives a listing of all the users subscribed to a specific team.

Numeric Replies:

- ERR_TOOMANYPARAMS
- ERR_NOSUCHTEAM
- ERR_NOTCONNNECTED
- REPL_SUBSCRIBED

Examples:

- /subscribed ; show subscribed teams of current user
- /subscribed “some_uuid” ; show all users subscribed to this team

4.2.3 /unsubscribe command

Command: /unsubscribe

Parameters: [“team_uuid”]

The /unsubscribe command is a used in the event where a user no longer wants to participate in a team. When unsubscribed a user shall no longer receive notifications of events occurring in that team.

Numeric Replies:
- ERR_NEEDMOREPARAMS
- ERR_TOOMANYPARAMS
- ERR_NOSUCHTEAM
- ERR_NOTCONNNECTED
- ERR_NOTSUBBED

4.3 Context Operations

4.3.4 /use command

Command: /use

Parameters: ?[“team_uuid”] ?[“channel_uuid”] ?[“thread_uuid”]

The use command’s purpose is context switching. It supports multiple optional commands, each of which, if included, gives rise to a different result. By default, the context of every user is undefined. The possible successful outcomes are the following:

- When no options are specified the context is changed to be undefined
- When the [“team_uuid”] is specified, the context is change to the team identified by the specified id.
- When [“team_uuid”] and [“channel_uuid”] are specified the context is changed to the channel identified by channel_uuid that is found under the team identified by team_uuid
- When [“team_uuid”], [“channel_uuid”] and [“thread_uuid”] are specified the context is changed to the thread identified by thread_uuid which is found under the channel identified by channel_uuid which in turn is found under the team identified by team_uuid

Numeric Replies:
- ERR_TOOMANYPARAMS
- ERR_NOTCONNECTED
- ERR_NOSUCHTEAM
- ERR_NOSUCHCHANNEL
- ERR_NOSUCHTHREAD
- REPL_USE

4.3.5 /create command

Command: /create

Parameters: Based on context

The create command is used for creating different sub resources based on the context. It can be used for creating teams, channels, threads and comments. This command must be used in combination with the /use command which is used for setting the context. The possible successful outcomes are the following:

- When the context is not defined (/use) the command sequence /create [“team_name”] [“team_description”], creates a new team with the name team_name and with a description team_description. Each team is also assigned a team_uuid on creation
- When team_uuid is defined (/use “team_uuid”) the command sequence /create [“channel_name”] [“channel_description”] ”], creates a new channel with the name channel_name and with a description channel_description. Each channel is also assigned a channel_uuid on creation
- When team_uuid and channel_uuid are defined (/use “team_uuid” “channel_uuid” the command sequence /create [“thread_title”] [“thread_message”], creates a new thread with the title thread_title and with a message thread_message. Each channel is also assigned a channel_uuid on creation
- When team_uuid, channel_uuid and thread_uuid are defined (/use “team_uuid” “channel_uuid” “thread_uuid”) the command sequence /create [“comment_body”], creates a new comment with the text comment_body. Comments do not have unique identifiers

Numeric Replies:
- ERR_TOOMANYPARAMS
- ERR_NEEDQUOTES
- ERR_NOTCONNECTED
- REPL_CREATE

4.3.6 /list command

Command: /list

Parameters: None

The list message is used to list all the sub resources of the context defined. This command must be used in combination with the /use command which is used for setting the context. These are the following situations for which list is defined:

- When the context is not defined (/use), /list lists all existing teams
- When team_uuid is defined (/use “team_uuid”, /list lists all existing channels
- When team_uuid and channel_uuid are defined (/use “team_uuid” “channel_uuid”) /list lists all existing threads

- When team_uuid, channel_uuid and thread_uuid are defined (/use "team_uuid" "channel_uuid" "thread_uuid"), /list lists all existing replies

Numeric Replies:
- ERR_TOOMANYPARAMS
- ERR_NOTCONNECTED
- RPL_LIST

4.3.7 /info command

Command: /list

Parameters: None

The info command is used for displaying detailed information about a sub resource. This command must be used in combination with the /use command which is used for setting the context. The possible successful outcomes are the following:

- When the context is not defined (/use) the /info command displays detailed information about the currently logged in users.
- When team_uuid is defined (/use "team_uuid") the /info command displays detailed information about the currently selected team.
- When team_uuid and channel_uuid are defined (/use "team_uuid" "channel_uuid") the /info command displays detailed information about the currently selected channel.
- When team_uuid, channel_uuid and thread_uuid are defined (/use "team_uuid" "channel_uuid" "thread_uuid") the /info command displays detailed information about the currently selected thread.

Numeric Replies:
- ERR_TOOMANYPARAMS
- ERR_NOTCONNECTED
- RPL_LIST

4.4 Sending messages

This group of commands is concerned with direct messaging between users. This type of communication is used when the current user wants to send a text message that does not need to be publicly seen to another user.

4.4.1 /send command

Command: /send

Parameters: [“user_uuid”] [“message_body”]

/send is used to send private messages between users. user_uuid is the user id of the receiver of the message. “message_body” is the text message that the user wants to send.

Numeric Replies:

- ERR_NOSUCHUSER
- ERR_NEEDMOREPARAMS
- ERR_TOOMANYPARAMS
- RPL_SEND

Examples:

- /send “someuuid” “Hello are you receiving this message?” ; Message to someuuid.

4.4.2 /messages command

Command: /messages

Parameters: [“user_uuid”]

The /messages command is used to list all the messaging history between the current user and the user identified by user_uuid.

Numeric Replies:

- ERR_NOSUCHUUID
- ERR_NEEDMOREPARAMS
- ERR_TOOMANYPARAMS
- ERR_NEEDQUOTES
- ERR_NOTCONNECTED
- RPL_MESSAGES

4.5 User based queries

User queries are a group of commands which are primarily concerned with finding details on a user or a group of users.

4.5.1 /users command

Command: /users

Parameters: None

The /users message is used by a client to generate a query which returns a list of all the users that exist on the domain. It can be executed in any context once the user is logged in.

Numeric Replies:

- ERR_TOOMANYPARAMS,
- ERR_NOTCONNECTED
- ERR_SYS
- RPL_USERS

Examples:

- /users ; List all existing users in domain

4.5.2 /user command

Command: /user

Parameters: [“user_uuid”]

This message is used to query information about a specific user. The server will answer this message with a listing of information including:

- user_name
- user_uuid
- subscribed teams.

Numeric Replies:

- ERR_NOSUCHUSER
- ERR_NEEDMOREPARAMS
- ERR_TOOMANYPARAMS
- RPL_USER

Examples:

- /user “user_uuid” ; return available user information

4.6 Miscellaneous messages

Messages in this category do not fit into any of the above categories but are nonetheless still a part of and required by the protocol.

4.6.1 /help command

Command: /help
Parameters: None

This is the only command that does not need authentication to generate a valid response. The /help command is used to show information on the various commands implemented on the sever.

Numeric Replies:

- RPL_HELP

5. Replies

The following is a list of numeric replies which are generated in response to the commands given above. Each numeric is given with its number, name and reply string.

5.1 Error Replies

ERR_ALREADYCONNECTED 400:
       Used to indicate that a user is already logged in
ERR_ALREADYSUBBED 401:
       Used to indicate a user is already subscribed to a channel
ERR_CHANNELEXISTS 402:
       Used to indicate that a channel with a given name already
exists
ERR_INVALIDCOMMAND 403:
       Used to indicate that the sent command is invalid
ERR_NEEDMOREPARAMS:
       Used to indicate that the command was missing arguments
ERR_NOSUCHCHANNEL 405:
       Used to indicate that no such channel exists with the uuid
specified
ERR_NOSUCHTEAM 406:
       Used to indicate that no such team exists with the uuid
specified
ERR_NOSUCHTHREAD 407:
       Used to indicate that no such thread exists with the uuid
specified
ERR_NOSUCHUSER 408:
       Used to indicate that no such user exists with the uuid
specified
ERR_NOTCONNECTED 409:
       Used to indicate that the user needs to be logged in to execute
       the command
ERR_NOTSUBBED 410:
       Used to indicate that the user needs to be subscribed to the
       team to person the action
ERR_TEAMEXISTS 411:
       Used to indicate that a team with a given name already exists
ERR_THREADEXISTS 412:
       Used to indicate that a thread with a given name already exists
ERR_TOOMANYPARAMS 413:
       Used to indicate that too many arguments were specified for the
       command.

5.2 Command responses

```
RSP_CREATE_CHANNNEL 500:
       Used to inform user that the channel has been successfully
       created
RSP_CREATE_COMMENT 501:
       Used to inform user that the comment has been successfully
       created
RSP_CREATE_TEAM 502:
       Used to inform user that the team has been successfully created
RSP_CREATE_THREAD 503:
       Used to inform user that the thread has been successfully
       created
RSP_HELP 504:
       Used to inform user that the help command has been successfully
       executed
RSP_LIST_CHANNEL 505:
       Used to inform user that the list channels command has been
       successfully executed
RSP_LIST_COMMENT 506:
       Used to inform user that the list comments command has been
       successfully executed
RSP_LIST_TEAM 507:
       Used to inform user that the list teams command has been
       successfully executed
RSP_LIST_THREAD 508:
       Used to inform user that the list threads command has been
       successfully executed
RSP_INFO_CHANNEL 509:
       Used to inform user that the info channel command has been
       successfully executed
RSP_INFO_TEAM 510:
       Used to inform user that the info team command has been
       successfully executed
RSP_INFO_THREAD 511:
       Used to inform user that the info thread command has been
       successfully executed
RSP_INFO_USER 512:
       Used to inform user that the info user command has been
       successfully executed


RSP_LOGIN 513:
       Used to inform users of the domain that a user has been
       successfully logged in
RSP_LOGOUT 514:
       Used to inform users of the domain that a user has been
       successfully logged out
RSP_SEND 515:
       Used to inform user that their message has been successfully
       sent
```

RSP_SUBSCRIBE 516:
    Used to inform user that they were successfully subscribed to a team

RSP_SUBSCRIBED 517:
    Used to inform the user that the subscribed command has been executed successfully

RSP_UNSUBSCRIBE 518:
    Used to inform the user that the unsubscribe command has been executed successfully

RSP_USE 519:
    Used to inform the user that the use command has been executed successfully

RSP_USER 520:
    Used to inform the user that the user command has been executed successfully

RSP_USERS 521:
    Used to inform the user that the users command has been executed successfully

RSP_MESSAGES 522:
    Used to inform the user that the messages command has been executed successfully

5.3 Notifications

NOTIF_MSGRCV 602:
    Used to notify a user that they have received a private message

NOTIF_THREADMSG 603:
    Used to notify team that a response was posted in a thread

NOTIF_TEAMCREATE 604:
    Used to notify domain that a team has been create

NOTIF_CHANCREATE 605:
    Used to notify a team that a channel has been created

NOTIF_THREADCREATE 606:
    Used to notify team that a thread has been created