Simple Text Transfer Protocol (STTP)

Table of Contents

1. Introduction

1.1. Objective

The objective of the Simple Text Transfer Protocol (STTP) is to provide a possible implementation for a text-based bulletin board over TCP. The protocol itself is implemented within the data section of a TCP segment. This allows STTP to offer all services provided by TCP as well as additional services that are explained in this document.

It is important to note that this model only considers “always on” servers (i.e. no long-term data is stored on the server; all data is stored locally within the server program).

1.2. TCP Services

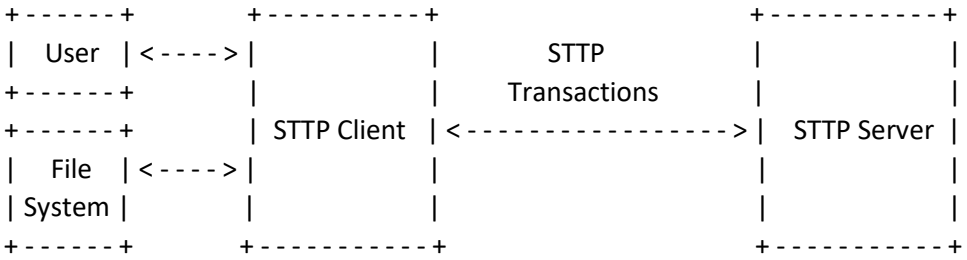
Since all messages are sent over a TCP connection, STTP ensures reliable transfer of information between the client and the server. Also, TCP creates a new session for each user that signs on which allows STTP to support multiple simultaneous connections.

1.2. STTP Services

This protocol will allow users to post messages on the public bulletin board, which anyone can read, and private messages to a group that is identified by a group ID and available only to those users that know the group ID. Users will be able to create/remove their own groups as well as being in control of who is a member of their groups. Any user can post messages to groups which they are a member and a user may also delete messages which they themselves have posted. Additionally, users can request to see all messages, new messages or specific messages based on subject line for any group which they are a member. Users will also be able to list all and new subject lines within a group as well as group members and messages posted by themselves. Finally, users can request the total number of messages or number of new messages posted to any group they are a member.

2. The STTP Model

The STTP design can be pictured as:



Once a TCP connection is established between the client and the server, it will remain open until the user requests to close the connection. This model requires the server to wait for the client to make requests based on user input and then respond accordingly. The format of these requests and responses will be discussed throughout the document.

2.1. STTP Client

An STTP client has three main responsibilities; file system management, requesting user input and generating requests to the server. File system management and requesting user input is crucial in generating valid requests to the server since every request requires that all fields are appropriately set. If they are not, the server may respond with information that the user was not expecting or more than likely the server will respond with an error. How to appropriately set these fields is further discussed in Section 3.2.1.

2.1.1. User

The STTP client should provide the user with available actions that can be executed by the server. When a user decides on an action, additional user input may be required to build the request depending on the desired action. Requirements for possible actions requested by the client will be presented in Section 3.2.1.

2.1.2. File System

File system management allows for seamless user specific sessions. An application implementing this protocol should create a file in the user's files system that contains the user's id (cookie) and a timestamp indicating what messages have been read by the user. This file should have 2 lines of text separated by a new line character. The first shall be the user's id and the second line shall be the timestamp. Upon sign out the information in this file shall be deleted. This will be further discussed in Section 4.

Note: It is not required that an application implement file system management although certain features may be lost if you choose not to. For example, a user may be required sign in every time they get open the application. Consult Section 4.1. before making this decision.

2.2. STTP Server

Once a TCP connection is established, an STTP server is required to wait for a request from the client before any action can be taken. When a request is received the server will parse though the transmission to generate a response. The client is responsible for ensuring that requests are formatted as described in Section 3.2.1. If the format does not match once it reaches the server then this behavior is undefined and application developer is free to implement this any way they choose. If a request to the server is not permitted by the user, then the server must return an error message.

2.2.1. Server Error Conditions

Requests not permitted by a user include:

1. Posting to a group that the user is not a member of

2. Deleting a group that the user did not create or does not exist

3. Deleting a message that the user did not post

4. Requesting the number of messages or members of a group which the user is not a member of

5. Listing message subject of a group which the user is not a member

6. Creating a group with the same ID as one that already exists

7. Removing/ Adding member to a group the user did not create

8. Invalid user IDs

3. STTP Transactions

Transactions between STTP client and server requires that four total transmissions be exchanged. First the client makes a request, then the server sends back the length of the response, client then sends back an acknowledgement that the length has been received and finally the server will send the actual response to the request. The reason for this structure is further discussed below along with the format of each transmission that is made within the transaction.

3.1. Basic Structure

STTP transactions can be pictured as:

```
        |- - - - - -Request - - - - > |
        |< - - - - - Length- - - - - - |
Client  |                             | Server
        |- - - - - - - OK - - - - - - - >|
        |< - - - - -Response - - - - |
```

When a client would like to make a request, it will first generate a request based on the format discussed in Section 3.2.1 then send it to the server. The server will then parse this request and generate an appropriate response based on the format discussed in Section 3.2.4.

Since messages on the bulletin board, number of messages on any bulletin board, number of groups and number of users are all arbitrary length, the number of sent bytes in the response is hard to predict. So, to ensure that the client receives all bytes, the server will first send the length of the response it has generated to the client. The client will then send “OK”, indicating that it has received the length of the response that is about to be sent. Once the server has received the “OK” it will then send the response to the server.

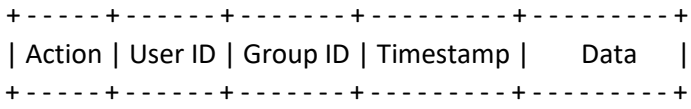
3.2 Transmission Structure

Each transmission within a transaction has a different required structure. The Request and Response transmission both have multiple fields that vary based on what needs to be sent. Whereas the Length and OK transmissions only contain one field.

For transmissions with multiple fields (e.g. Request/Response) each field is required to be separated by an end of text character and the transmission must be terminated with an end of transmission character. If the data field contains numerous segments of text, then each segment must be separated a page break.

3.2.1 Client to Server Request

The client to server request can be broken down into 5 fields and pictured as:



The action field is used to specify what the client is requesting from the server. This field can be set to either GET, POST, DELETE, ADD, REMOVE, LIST, NEW, EXIT, COUNT. How these actions are implemented at the server is specified below. The user ID field is the only field within a user session that should never change, it allows the server to identify which user is making the request. The group ID field is required to specify to which group a user would like to perform an action. The timestamp field indicates which messages a user has already viewed. Finally, the data field is used to specify any additional information that an action may require.

3.2.1.1. GET

The GET action allows a client to get all messages or new messages from any group that the user specified by user ID is a member. It also allows this user to filter all messages within a group that contain a specified subject line. The timestamp should only be updated when the user requests to get all messages or new messages from a group. For all GET requests the user ID should be retrieved from the users file system and the group ID should be input by the user.

To get all messages from a group, the timestamp field and data field must be set to ‘0’.

To get new messages from a group, the timestamp should be retrieved from the users file system and the data field shall be set to ‘0’.

To filter messages in a group, the timestamp should be set to either ‘0’ or the timestamp received from the users file system based on whether the user would like to filter all messages or just new messages. The data field shall be set to the filter criteria that the user specifies. Note that it may be useful to first list all subject lines within the desired group before asking the user for filter criteria.

3.2.1.2. POST

The POST action allows a client to post a new message to any group that the user specified by user ID is a member. This action requires that user ID and timestamp be retrieved from the users file system. The group ID shall be input by the user. The data field shall contain the contents of the message the user would like to post, including subject line and body.

The data section should be formatted as:

Subject Line [page break] Body [end of transmission]

3.2.1.3. DELETE

The DELETE action allows a user to delete any messages that they themselves have posted. This action requires that user ID and timestamp be retrieved from the user file system. The group ID

shall be input by the user and the data field shall contain a message ID input by the user. All message ID's of messages posted by the user can be obtained using the LIST action, consult Section 3.2.1.6.

3.2.1.4. ADD

The ADD action allow a user to create their own group or add members to a group that they added. The user ID and timestamp shall be received from the user file system. The group ID should be input by the user. If the group ID does not exist, then the group should be added.

If the data section is '0' and the group does not exist, then the group will be created with the only member being the user that created it. Alternatively, you could ask the user if they would like to add members upon creation. If they would, each user ID should be separated with a page break within the data section.

If the data section is not '0' and the group exists, then the members specified in the data section will be added to the group. If the data section is '0', then the server will assume a group is trying to be created with a group ID that already exists and return an error.

3.2.1.5. REMOVE

The REMOVE action allows a user to delete a group or remove members from a group that they have created. The user ID and timestamp shall be received from the user file system. The group ID should be input by the user. This action requires that the user ID field is the same as the user ID identified as the owner of the group at the server. If not, an error shall be returned.

To remove a group, set the data field to '0'.

To remove a member(s) from a group, the user ID's you would like to remove shall be specified in the data section separated by a page break.

3.2.1.6. LIST

The LIST action serves multiple purposes including listing all groups a user is a member of, list all users in a specified group, list all messages post by the user within a specified group and listing all or new subject lines within a group. The user ID and timestamp shall be received from the user file system whereas the group ID should be input by the user. Each specific action requires that the data field and timestamp field be set appropriately.

To list all groups a user is a member of, set the data field equal to 'GROUPS'.

To list all users in a group, set the data field equal to 'USERS'.

To list all messages posted by a user within a group, set the data field equal to 'MESSAGES'.

To list all or new subject lines within a group, set the data field equal to '0' for all message subject lines or set it equal to the timestamp received from the user file system for new message subject lines.

3.2.1.7. NEW

The NEW action allows the client to create a new user by obtaining a unique user ID and a timestamp from the server so the client can create a file in the users file system allowing for seamless unique user sessions. How the sessions are created is further explained in Section 4.1.

To create a new user, set the user ID, group ID and timestamp fields all equal to ‘0’. If an application requires user log in with a user name and password the data field can be set with these values separated by page break.

3.2.1.8. EXIT

The EXIT action allows for the client to end a user session with the server. This will close the TCP connection that was previously established and should update the user’s information if any is stored at the server. The client should rewrite the new timestamp data to the users file system before exiting if updates are not done incrementally.

To exit a user session, the user ID and timestamp shall be received from the users file system. The group ID and data fields shall be set to ‘0’. If a premature exit is required that the user ID and timestamp fields may be set to ‘0’.

3.2.1.9 COUNT

The COUNT action allows the user to receive the total number of messages or number of new messages posted to a group. The user ID and timestamp shall be received from the users file system. The group ID should be input by the user and the data section should be set based on whether the user wants total number of messages or only the number of new messages. If the user wants the total number of messages the data shall be set to ‘0’ whereas if the user wants the number of new messages, the data field shall equal the timestamp field.

3.2.1.10. LOAD

The LOAD action allows the user to sign into an already existing account. The user ID, group ID and timestamp shall be set to ‘0’. The data field should contain the users sign in information. If both a username and password are required to sign in, they must be separated by a page break.

For proper execution of this command it is required that the user IDs mapped to timestamps be stored at the server so that a session can pick up where it left off. If additional user information must be preserved in between sessions, then it is required that this information also be stored at the server (e.g. username and password).

3.2.2. Server to Client Length

The server to client Length transmission contains only one field. This field contains an integer value indicating the number of bytes the server will send the client in its response. For ease of parsing this transmission it is not required that it be terminated with an end of transmission character.

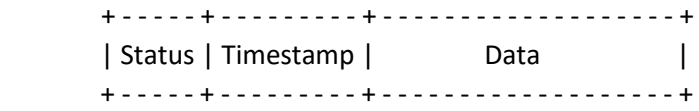
3.2.3. Client to Server OK

The client to sever OK transmission is exactly what it sounds like. It consists of one field the contains the string ‘OK’ to indicate to the server that the Length transmission has been received

and the client is now ready to receive the response. As with the Length transmission, the OK transmission is not required to be terminated with an end of transmission character.

3.2.4 Server to Client Response

The server to client request can be broken down into 3 fields and pictured as:



The status field is used to specify the result of the request that the client previously made. This field can be set to either OK, REFRESH or ERROR. How these statuses should be interpreted by the client is specified below. The timestamp field is used to update the users timestamp when a GET all or GET new request is made, otherwise it should always be set to the timestamp set in the users request. Finally, the data field is used to send back data that the user has requested.

3.2.4.1. OK

The OK status signifies that the user request has be successfully executed by the server. The format of the data section will vary based on what request is made, these formats are discussed below. If a request does not require data from the server then the data field should be set to ‘0’.

GET Request data format can be viewed as:

User ID [page break] Subject [page break] Body [page break] Timestamp [page break] . . . . User ID[page break] Subject [page break] Body [page break] Timestamp [end of transmission]

LIST Request data format can be viewed as:

Content [page break] Content [page break] . . . . [page break] Content [end of transmission]

The only LIST request that should not follow this format is LIST MESSAGES. The format of LIST MESSAGES data should follow the same format as the GET Request.

Other Requests:

All other requests require that no data or only a single data field be returned so there is no special formatting. Although remember to always terminate the data section with the end of transmission character as discussed in Section 3.2.

3.2.4.2. REFRESH

The REFRESH status is a supplementary feature that is not required to be implemented. Its only purpose is to add additional functionality that application developers may find useful. This status allows the server to signify to the client that new data has been successfully added to the server. Since the refresh status is used to signify updates to the server the data field should always be set to ‘0’.

3.2.4.3. ERROR

The ERROR status allows the server to indicate to the client that the request that was made is prohibited. The data field should be set to a string indicating why the request made by the client was invalid so the client can avoid this error in the future.

4. User Sessions

4.1. Session Initiation

Once a TCP connection is established the client is responsible for loading a user session or requesting a new user session. Upon initiation of a user session the client should search for an application specific file in the users file system. If this file exists and contains both a cookie and timestamp value in the format discussed in Section 2.1.2. then it may be assumed that an active user session exists. If no such file exists or if the file exists but is not of the correct format, then a user session must be loaded or created.

4.2. Session Termination

When a user chooses to exit, they should be given the option of signing off before terminating the connection. If the user chooses to stay signed in, then the user's file system should be updated with a timestamp that accurately reflects which messages have been read and conform to the format discussed in Section 2.1.2. If a user chooses to sign off, the client must either reformat the application specific file or delete the application specific file if one exists in the users file system. It is up to the developer how he/she would like to implement this.

5. Conclusion

This paper proposes a Simple Text Transfer Protocol (STTP) that provides a possible implementation for text-based bulletin board application over TCP. It is naïve implementation that provides a good starting point for any developer.

5.1 Considerations

This protocol does not discuss how to remove user accounts from the application since there are many ways an application can choose to handle this. For example, an application may desire that only the user id becomes invalid or it may prefer that all traces of a user be deleted (i.e. all messages posted by that user as well as rendering the user id invalid).

If a developer requires this feature for their application, then it can be accomplished using the REMOVE action and carefully setting the other fields to not conflict with removing a group or user from a group.