Distributed Computing Project

Submitted By: Darragh Elbel T00211193

Computing with Software Development

Date Submitted: 29/03/24

Contents

[Introduction 3](#_Toc162606577)

[Short Message Protocol 3](#_Toc162606578)

[Introduction 3](#_Toc162606579)

[Functionality 4](#_Toc162606580)

[Log-on 4](#_Toc162606581)

[Short Message Upload 6](#_Toc162606582)

[All Message Download 7](#_Toc162606583)

[Specific Message Download 8](#_Toc162606584)

[Log-off 8](#_Toc162606585)

[Protocol-Wide Error Handling 8](#_Toc162606586)

[Secure Communications 8](#_Toc162606587)

[Application Design and Implementation 9](#_Toc162606588)

[Design 9](#_Toc162606589)

[Objectives and Overview 9](#_Toc162606590)

[System Design 9](#_Toc162606591)

[Server-side Design 10](#_Toc162606592)

[Client-side Design 12](#_Toc162606593)

[Implementation 13](#_Toc162606594)

[Server Side Implementation 13](#_Toc162606595)

[Client Side Implementation 15](#_Toc162606596)

[SSL Demonstration 17](#_Toc162606597)

[User Manual 18](#_Toc162606598)

[Conclusions 20](#_Toc162606599)

# Introduction

The first objective of this project is to design a client-server messaging protocol, called Short Message Protocol, based off the key requirements that the protocol should facilitate logon, logoff, upload, and download messages in a secure fashion. Protocol design specifies how messages should be formatted and the sequences in which actions should occur, including comprehensive guidelines for handling errors. The second objective of this project is to design and implement the protocol on both the client and server side using Java. This work includes designing the classes that will be required and their properties, these designs will then be used to build a secure implementation of the defined SMP functions which will be tested to ensure it is both secured via SSL and can support concurrent connections from multiple clients simultaneously.

# Short Message Protocol

## Introduction

The purpose of SMP is to provide a protocol which allows clients to securely connect, upload, and download messages from a server. SMP consists of 5 core functions, these being logon, the ability for a user to connect to the server using username and password combinations. Logoff, allowing users to disconnect from the server. Message upload, allowing users to upload and save a short message to the server. Specific message download, allowing users to retrieve a specific message. Finally, all message download, allowing users to retrieve all messages at once. The server should be capable of storing up to 1000 short messages for every user and messages are identified by a message number from 0 to 999.

## Functionality

For each function requirement of the system (log-on, logoff, message upload, specific message download, bulk message download) the relevant section in this protocol definition contains a description of the request and response messages between client and the SMP server. Each section also contains an error table of function-specific errors which define the error code, error message, and description of the erro. The UML sequence diagram in figure 1 demonstrates the sequence of events in a client-server interaction. First, the client sends a handshake request which the server responds to, this is necessary for implementing SSL which is discussed in the ‘Secure Communications’ section of this document. Next the client sends a message to the server, the server performs error checking and if there are none responds to the message with the corresponding function. In the alternate scenario that an error is identified, the server simply returns the error message associated with the error.

sA diagram of a flowchart

Description automatically generated

Figure 1 Protocol Sequence Diagram

### Log-on

System log-on should be facilitated by verifying username and password combinations, transferred to the server for authentication using SSL. Usernames and passwords should be a maximum of 256 bytes each. Only valid ASCII characters should be permitted. The maximum size of a LOGON request message is 519 bytes. 5 bytes are allocated to the LOGON prefix, 1 byte allocated to the following space, 256 bytes maximum for the username, 1 byte allocated to the following space, and finally 256 bytes maximum for the password.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 Bytes | 1 Byte | 256 Bytes | 1 Byte | 256 Bytes |
| LOGON | Blank space (ASCII 255) | Username | Blank space (ASCII 255) | Password |

Example LOGON: “LOGON myusername mypassword”

On successful logon, the server should respond to the client with the message “USER AUTHENTICATED”. See error table 2.1.1 for details on authentication failure and other LOGON command errors.

#### 2.1.1 Error Table

|  |  |  |
| --- | --- | --- |
| Code | Error Message | Description |
| 001 | ERR01 – INVALID AUTH FORMAT | Returned when a request is detected as being of type LOGON, but the message contains a greater number of bytes (>519) than is permitted. |
| 002 | ERR02 – WRONG USERNAME/PASSWORD | Returned when user authentication fails due to an invalid username/password combination. |

### Short Message Upload

Messages upload via SMP are facilitated by the MSGUP command. Authenticated users should only be able to access their own messages, it should not be possible for any users to gain information on any other user of the system. A message upload packet should not exceed 1024 bytes, 6 of which are reserved for MSGUP and a following space.

|  |  |  |
| --- | --- | --- |
| 5 Bytes | 1 Byte | 1018 Bytes |
| MSGUP | Blank space (ASCII 255) | Message Content |

Example MSGUP: “MSGUP my test message”

On successful message, the server should respond to the client with the message “MESSAGE UPLOADED SUCCESSFULY”. See error table 2.1.1 for details on authentication failure and other MSGUP command errors.

#### 2.2.1 Error Table

|  |  |  |
| --- | --- | --- |
| Code | Error Message | Description |
| 003 | ERR03 – INVALID MESSAGE FORMAT | Returned when a request is detected as being of type MSGUP, but the request is greater than 1024 bytes in size. |
| 004 | ERR04 – NO MESSAGE | Returned when a MSGUP request does not contain message content. |

### All Message Download

Message download is facilitated by the MSGDL command. The MSGDL command is not accompanied by any other arguments and is 5 bytes in length. Downloaded messages should be returned in the format, if multiple messages exist they should be concatenated together with a blank space “ “ and a semicolon as a delimiter “;”.

“MSG-[MESSAGE NUMBER] [MESSAGE CONTENT];”

For example, if the user as only uploaded one message, it should look like the following “MSG1 – This is my first message”. If the user has multiple message, it would look like this “MSG1 – This is my first message; MSG-2 This is my second message”. When visible by the client, message numbers should NOT contain 0 prefixes. For example “MSG-01” is not a valid message identifier.

|  |
| --- |
| 5 Bytes |
| MSGDL |

The maximum possible return for MSGDL if the user has stored 999 messages is 1,025,875 bytes, approximately 1 megabyte, so the client and server implementations should both support the communication of messages this size in a single transfer.

#### 2.3.1 Error Table

|  |  |  |
| --- | --- | --- |
| Code | Error Message | Description |
| 005 | ERR05 – NO MESSAGES STORED | Returned when the logged-in user has no messages saved to the server. |

### Specific Message Download

Message download is facilitated by the SPMSG command. The SPMSG command is accompanied by a specific message id. For example “SPMSG 0” will return the first message uploaded by the user. Note that messages are 0 indexed.

Example SPMSG: “SPMSG 0”

|  |  |  |
| --- | --- | --- |
| 5 Bytes | 1 byte | 3 bytes |
| SPMSG | Blank space | 0-999 |

#### 2.4.1 Error Table

|  |  |  |
| --- | --- | --- |
| Code | Error Message | Description |
| 006 | ERR06 – NO MESSAGES WITH PROVIDED ID | Returned when there is no message saved with the provided ID. |

### Log-off

System log-off is facilitated through the LGOFF command. The MSGDL command is not accompanied by any other arguments and is 5 bytes is length. Once logged off, the user’s session should be terminated.

|  |
| --- |
| 5 Bytes |
| LGOFF |

#### 2.3.1 Error Table

SMP does not have any Log-off specific errors, failure to log-off due to not being signed in is facilitated by protocol-wide error handling.

### Protocol-Wide Error Handling

|  |  |  |
| --- | --- | --- |
| Code | Error Message | Description |
| 007 | ERR7 – INVALID COMMAND LENGTH | Returned when the first 6 bytes of a message are not 5 characters of any type plus a blank space. |
| 008 | ERR8 – INVALID COMMAND TYPE | Returned when the message type is not recognized. |

### Secure Communications

Secure communications between the client and server should be facilitated through the use of SSL using a combination of public-key and symmetric encryption. The SSL implementation should apply to all messages send across the network to the server, including messages from a client that have not yet been authenticated to ensure the secure transport of username and password combinations.

# Application Design and Implementation

## Design

### Objectives and Overview

The objective of this system design is to provide a comprehensive guide for the development of a client and server that can communicate messages securely via the Short Message Protocol. The system will be implemented in Java, and will allow for the concurrent connection of clients to a single server securely through TLS/SSL. The system will use EchoServer3, EchoServerThread, EchoClient2, EchoClientHelper, and MyStreamSocket as a starting point. In addition to these, the classes UserSession, SMPClientUI, and GlobalErrorMessages (an Enum) were added.

### System Design

The following sequence diagram illustrates the client connecting to the server, logging on, and ending the session, demonstrating connection creation, message handling on the server side, and session termination.

A diagram of a process

Description automatically generated

### Server-side Design

#### Full Class UML

As can be seen in the following UML diagram of the classes that comprise the server side of the SMP application, the SMPServer class serves as the application’s entry point. The SMPServer is responsible for creating SMPServerThread instances, each of which encapsulates a UserSession object when a client connects and authenticates. SMPServerThread threads rely on the MyStreamSocket class for managing socket connections from clients.

A screenshot of a computer

Description automatically generated

#### SMPServer

SMPServer will remain mainly unchanged from the EchoServer3 implementation, the only modification required being the addition of SSL handling.

#### UserSession

The UserSession class is used by a server thread to store user messages.

#### SMPServerThread

SMPServerThread requires the addition of 8 new functions.

* parseReqType – Takes in a string and determines if it is a valid request
* getErrType – Takes in a GlobalErrMessage Enum, maps these Enums to error response strings.
* getCommand – Takes in a string and calls the correct function based on the request type.
* logon – Accepts username and passwords, creates a new UserSession object.
* logoff – Sets UserSession to null
* uploadMessage – Takes in a message and adds it to the current user’s messages.
* downloadMessages – Returns all of the users messages.
* downloadSpecificMessage – Accepts a specific message ID and returns that message.

#### GlobalErrorMessages

An Enum class used for mapping error types to responses. Each of these errors comes from the SMP protocol defined earlier in this document.

A screenshot of a computer program

Description automatically generated

### Client-side Design

#### SMPClientUI

A Java Swing UI, maps buttons to text messages which are sent to the server.

#### SMPClientHelper

Based on EchoClient2Helper, the only changes required being to integrate SSL.

# Implementation

### Server Side Implementation

The following UML diagram demonstrates the relationship between the server-side classes of the application. SMPServer is the entry point to the application, which is used to create a SMPServerThread for each connected client, this is a one to many relationship. SMPServerThread objects then in turn creates a new user UserSession object which contains an array list of messages, along with username and password attributes. When initialized a thread does not have a UserSession, but once the LOGON command is sent to the server via the logon function, a UserSession object is instantiated. Designing the system in this way means that each user has their own thread and UserSession object to house their messages allowing for concurrent connections from multiple clients.

A diagram of a computer

Description automatically generated

Contained within the run method of SMPServerThread, the following code snippet is responsible for checking all incoming messages for errors, if the parseReqType method returns a NO\_ERR Enum, the message is processed, however if it is not the getErrType function is called to parse the error type and return it.

A computer screen shot of a message

Description automatically generated

By using parseReqType commands are essentially pre-screened before any attempts are made to parse the message and perform any related logic.

A computer screen with text and images

Description automatically generated

Determining messages types is handled by the getCommand function, which in this implementation uses a switch statement to check the incoming message type, before calling the relevant function for that request. This is acceptable for this version of SMP as there are a total of 5 possible commands, but if the number of commands was to grow in future iterations considerations should be made to refactor this switch statement using an approach such as the Strategy pattern. The one exception to command handling is in the case of LGOFF, the thread monitors all incoming messages for a termination message, and if LGOFF is detected the server will return the output of the logoff() method and close the socket before any other processing is performed.

A screen shot of a computer code

Description automatically generated

The command implementations are quite simple, downloadMessages and downloadSpecificMessage for example. downloadMessages simply loops through the arraylist of messages in the UserSession object and uses a stringbuidler to concatenate them together in the format specified by the protocol, arraylist indexes are used as message identifiers. By using indexes as identifiers the downloadSpecificMessage is made equally simple, as the user is requesting the index of the message as its id.

A computer screen with text and images

Description automatically generated

SSL is implemented in the SMPServer class keystores were generated in the same way as in the SSL lab we covered, and a SSLServerSocketFactory was used to create a sslServerSocket which was wrapped by the MyStreamSocket class.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

<https://www.ibm.com/docs/en/sdk-java-technology/8?topic=interfaces-sslsocketfactory-sslserversocketfactory-classes>

<https://stackoverflow.com/questions/53323855/sslserversocket-and-certificate-setup>

### Client Side Implementation

The client side of the application is comprised of three classes, SMPCilentUI, SMPClientHelper, and MyStreamSocket. SMPClientUI is responsible for the client GUI, and handles logic for sending messages of various types, SMPClientHelper is responsible making the connection to the server and handling messages in and out.

A diagram of a computer

Description automatically generated

The following code snippet is responsible for the implementation of SSL on the client side within SMPClientHelper.



As can be seen in the class client class diagram, SMPClientUI has logoff, logon, downloadMessageByID, sendMessage, and downloadMessages methods, each mapping to one of the command functions defined in SMPServerThread. These functions call SMPClientHelper with the relevant command. And are then responsible for displaying the message response to the GUI.

A screen shot of a computer program

Description automatically generated

Some of these methods such as logOff manipulate the UI to a greater degree, which calls the resetUIForLoggedOutUser method. resetUIForLoggedOutUser is responsible resetting the UI to a pre-login state.

A screen shot of a computer program

Description automatically generated

### SSL Demonstration

To ensure that SSL is in place on the server side within the run function of SMPServer thread I create a new SSLSession object using a MyStreamSocket instance (which was used to wrap an SSLSocket in SMPServer), and print the result of getCipherSuite() and getProtocol().



To make sure SSL was working as expected, I examined the packets of the original EchoServer3 application using Wireshark, and could clearly see request and responses in plain text.

A computer screen shot of a keyboard

Description automatically generated

After examining the original EchoServer3 application, I checked the requests and responses of the SMPServer application and was able to verify that message were not visible in plain text.

A computer screen shot of a computer

Description automatically generated

I was curious as to why I was not seeing TLS as the protocol type, the issue was that since SSL/TLS was not running on the standard port Wireshark was not picking up on it, after editing preferences and adding port 7 to the configuration I was able to see Client Hello and Server Hello messages as well as the correctly displayed protocol.



### User Manual

#### Running The Server

From the CLI navigate to the server folder and the run the following command “java SMPServer”. The server should now be running, you will see “Waiting for connection” printed to the terminal.

#### Running Client(s)

In the client folder, run the command “java SMPClientUI”. A new client window should appear as seen in the following screenshot. Click the “LOGON” button and click “OK” on the login pane, no real username or password is required.

A screenshot of a computer

Description automatically generated

#### Uploading a Message

To upload a message, click on the highlighted text input box, type your message, and click the “Upload Message” button. You should see a message “Server: Message uploaded successfully” in the messages panel.

A screenshot of a computer

Description automatically generated

#### Downloading All Messages

Click the “Download All Messages” button, all of your messages will be printed to the output pane, along with their relevant IDs.

A screenshot of a computer message

Description automatically generated

#### Download Specific Message

Click the “Download by ID” message and input the number of the message you would like to retrieve.

A screenshot of a computer

Description automatically generated

#### Log Off

Click the “Log Off” button to log off. You will be returned to a blank output panel with only a single “LOGON” button being visible.

# Conclusions

From the given requirements this project documents a proposed design of the Simple Message Protocol, defining the message and response formats for each of SMP’s core functions (logon, logoff, message upload, message download). These definitions include allowed keywords, message lengths, as well as defining the possible errors that can be thrown by the functions of the protocol. The protocol definition also lays out how TLS/SSL should be implemented to ensure secure communications

After protocol design the next step was to design the server and client sides of the application which would go on to implement SMP.

Finally, after developing the implementation SSL encryption was added, it was verified to be working by using Wireshark to examine the packets being transferred by the application.