TABLE OF CONTENTS

PROBLEM STATEMENT-UNDERSTANDING THE PROJECT

The idea of the project is to setup a client-server application using the concept of Java's UDP sockets. Upon receiving a request from the client, the server will serve temperature measurement values (in degrees Fahrenheit) to the client. The client sends a measurement ID to the server and the server will respond by sending the temperature measurement corresponding to the measurement ID back to the client using UDP's unreliable data transfer services. The measurement IDs and the corresponding measurement values can be found in the file data.txt which are present both at the client and the server. The client sends a request by randomly choosing a measurement ID from the text file and the server responds the corresponding temperature value by reading the same data.txt file.

With UDP's connectionless data transfer protocol, there is no handshaking between sender and receiver and hence there's no guarantee whether the data has arrived or is intact. However, due to a minimum of overhead in comparison with TCP, a much faster data transmission is possible over high quality physical links. In this project, the reliability of the communication link is ensured by adding extra features in the protocol like integrity check, timeout and, if needed, retransmission of packets as per the user selection.

The protocol requires a client and server setup using UDP socket parameters to facilitate a successful data transfer link. The required protocols are ensured in both server and client machine to ensure reliability and functionality of the link. While communicating, both server and client send a message of defined and fixed character sequence for server and client respectively. For the client transmitted message, there should be two more message elements: the request ID, the measurement ID. The request ID should be a 16-bit unsigned integer randomly generated by the client, and its role is to identify the request/response message pairs. Since there is no explicitly defined unsigned integer in Java except for char, it can't be used when performing arithmetic operations like modulo, division etc. Hence the ideal way is to Use the lower 16 bits of "int" and remove the higher bits with & 0x0000ffff when necessary. The whole message is then processed as per the given algorithm for deriving an integrity check number. The integrity check value is also a part of user's request message which is sent to the server. The integrity check field is then recalculated and matched to ensure theirs is no error or irregularity during transmission to server's end. In case of anomaly, the whole process restarts for and prompts user either to re-send the request or to request the file again.

Server message includes a response code which denotes the state of the message received so that the client can respond accordingly. Server checks for integrity, valid measurement and syntax match for the request etc. to generate the correct response code. If everything is okay, the server reads the requested text file from its given directory and includes it in the response message in the value. The response message with a response code 0 including the requested content and the integrity check value is converted into a byte array and sent as a response back to the client. The server goes back to step 1 and wait for another session.

The design of the program is then described in this report with necessary flowcharts, and UML class diagrams. The solution design segments are discussed in details with various output scenarios and correct functionalities.

ALGORITHM FOR THE CLIENT APPLICATION

Step 1.

- The application selects one of the available measurement IDs to receive the corresponding temperature measurement value from the server which is present in the data.txt. A function called "FileReadClient()" performs the action of reading the file and storing it in an array. A random variable, which is used as a key to the measurmentID array, is generated and the value corresponding to the array-key is returned from the function which is the random measurmentID that is selected from the text file.

Step 2.

- It generates a random request ID by importing random class. Since the request id should be a 16 bit unsigned Integer. The maximum value is set to 65535 during the generation.

Step 3.

- It assemble the request message as a sequence of characters (as a String object), including the integrity check field, converts it into a byte array using getBytes(), a String method and sends it to the server.
- It also starts a timer at this point with initial timeout value of 1 second. A counter called timeoutCount is initiated and is set to zero

Step 4.

- The client waits for the response from the server. If no response arrives and the timer expires, it resends the request, and the timeout interval is doubled at each timeout event (i.e., 1,2,4,8). During every timeout, the timeoutCount is incremented by 1. After the 4th timeout event i.e., when the timeoutCount=3, the client declares a communication failure and prints an error message on the screen.
- If the response arrives after several timeout events and retransmissions, the timeout value and the timeout counter is reset to their initial values.

Step 5.

- It calculates the integrity check value for the response message and compares it with the

integrity check value supplied in the response message.

- The algorithm for the checksum calculation is detailly explained in the later section. If they are not equal, the loop goes back to step2 and repeats sending the request message.

Step 6.

- It checks the value of the response code. The code is extracted from the obtained message by first replacing all the non-digits with " " and splitting the obtained string based on space. If no errors occurred (i.e. if it code is 0), then it displays the measurement value received from the server on the screen.
- If some errors occurred, it displays an error message on the screen indicating the problem that has occured.
- If the error was that the request message's integrity check failed (response code 1), it asks the user if he/she would like to resend the request and go back to step2 and resend the request if needed. This feature is implemented in the function responseCompare(String).

ALGORITHM FOR SERVER APPLICATION

Step 1.

- The server waits for requests on the specified port. The UDP transmission and reception is setup by importing Datagram Packets and Datagram Sockets classes.

Step 2.

- Upon reception of a request message, the byteArray is converted to a String with the help of String constructor.
- The server verifies the integrity of the message by calculating the integrity check value and comparing it with the integrity check value provided in the request message.
- This is done with the help of checksum() function implemented at the server side. If they are different, it sends back an error response with response code 1.

Step 3.

- If the integrity check passes, the received message is to be checked for a Syntax Match. This includes checking that all opening and closing tags are present (no misspelling or invalid characters) and they are in the right order. The syntax of the element values should also be checked, e.g. that the number between the <measurement> and the </measurement> tags is a 16-bit unsigned integer. The number is checked if it is a 16 bit unsigned Integer, by verifying that it lies between 0 and 65535.
- If the syntax check fails, it should send back an error response with response code 2.
- The functionality for appending the code is implemented in the statusCode() function and the Syntax is verified in the SyntaxMatch function.

Step 4.

- It looks up for the measurement value that corresponds to the requested measurement ID, as defined in the data.txt file.
    - If measurement value exists for the ID, it returns the corresponding value
    - If there is no measurement value for the requested measurement ID, it sends back an error response with response code 3.
- The look up in the text file is implemented in the fileReadServer() function.

Step 5.

- It assembles the response message with response code 0 including the requested measurement value and the integrity check value, converts it into a byte array and sends the response back to the client.
- It also sends the response message with response code 1,2 and 3 but doesn't include the measurmentID and value in it.

Step 6.

- It should go back to step 1 and wait for another request message.

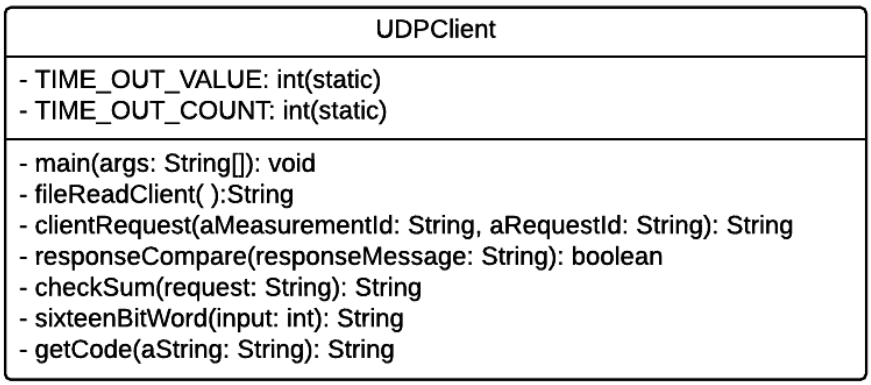
ALGORITHM FOR CHECKSUM IMPLEMENTATION

The integrity check is calculated as follows:

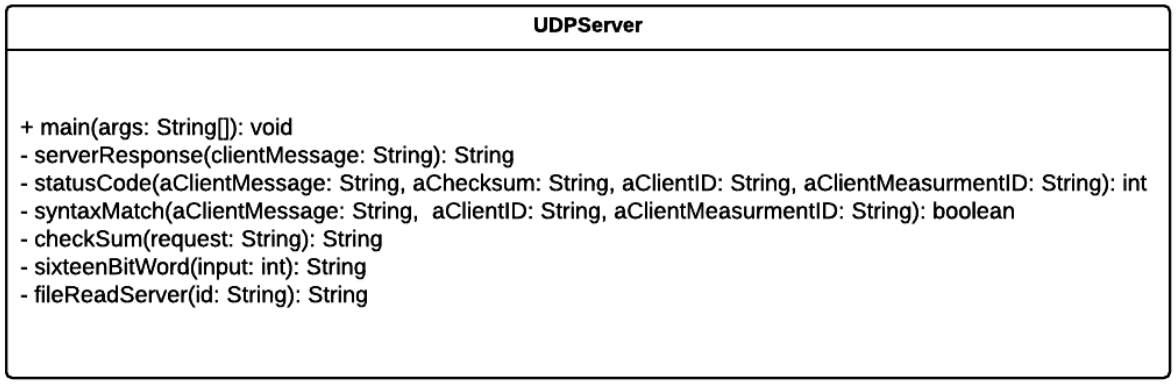
- The evaluation of checksum is done in the checksum() function both at the client and the server
- First, the character sequence is sent as an input to the function. The sequence is converted into a 16 bit word using the function sixteenBitWord().
- The sequence is checked for the length. If the length is even, it is directly divided by 2 else, 1 is added to make the value even and is divided by 2. This value is stored in a variable called xLength which will give the count of 16 bit words that can be generated from the input sequence(x[0] to x[xLength-1])
- The x[] array is declared as int since all the mathematical operations have to be performed. The ASCII code for the 1st character is made the most significant byte of the 1st 16-bit word, the ASCII code for the 2nd character is made the least significant byte of the 1st16-bit word, and so on, each 16-bit word contains the ASCII codes for two consecutive message characters.
- If the message consists of an odd number of characters, the least significant byte for the last
- 16-bit word is set to zero as per the algorithm mentioned.
- The index and S are 16 bit unsigned integers with the value ranging from 0 to 65535. Since the index is involved in arithmetic operations it is declared as int and the sign bit is made to 0 by multiplying it with 0x00007fff; Since s the checksum isn't involved in arithmetic operations (is used only in logical operation -XOR) it can be declared as char which is a 16-bit unsigned Int.
- This checksum value is converted to a String and is appended to the message to be sent.
- At the receiver, the same process is repeated, and the checksum value is re-calculated.
- Then, the calculated checksum value should be compared with the received checksum value, and if they are not equal, it is assumed that there are some bit errors in the received
- Correspondingly the status code is to be sent. If the checksum received matches with the calculated checksum then there is no bit error and status code 0 is sent, assuming the syntax of the received message is correct.
- If bit errors exist i.e., the calculated checksum is not equal to the obtained checksum then code 1 is sent

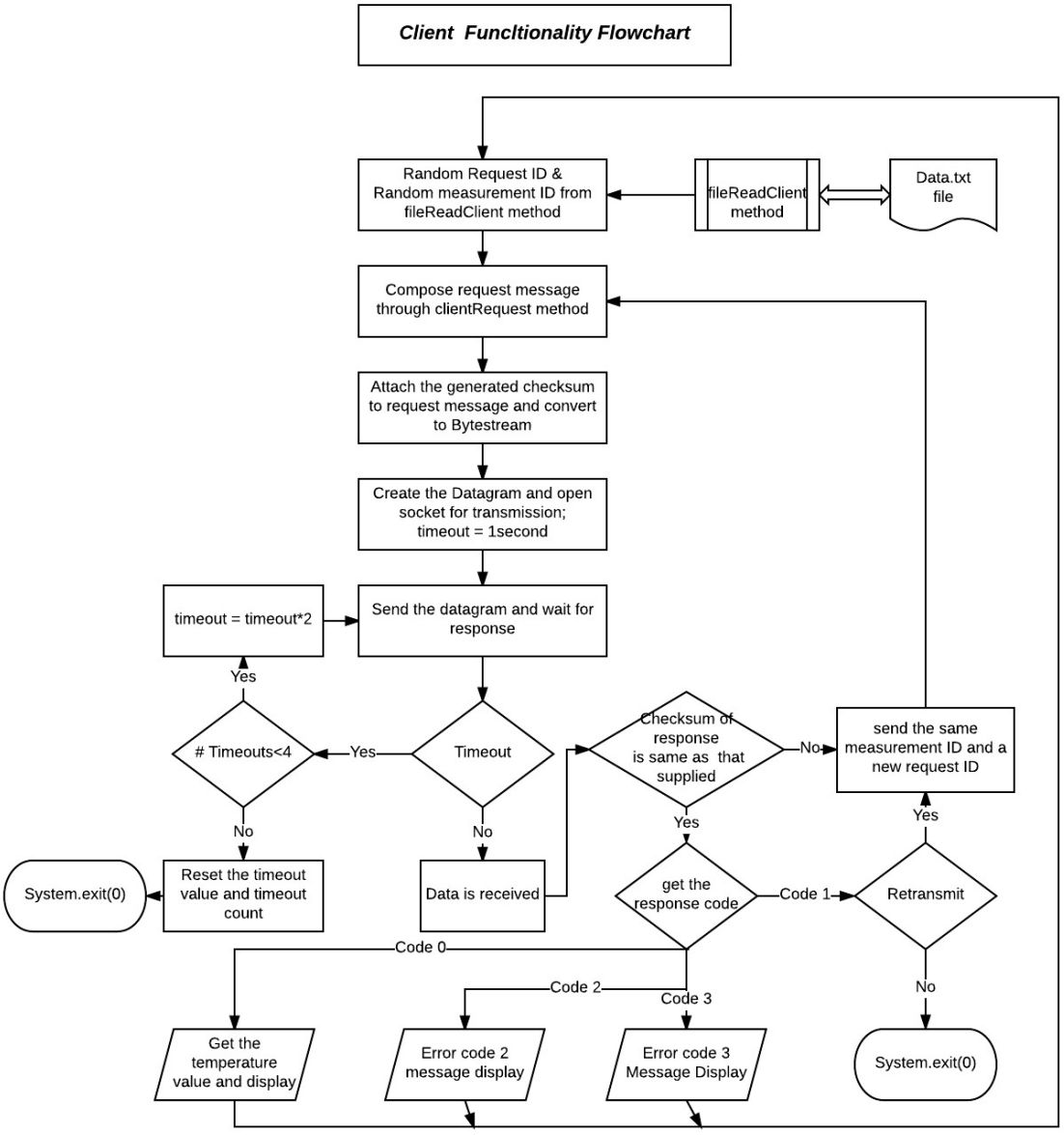
UML DIAGRAMS

FOR THE UDP CLIENT

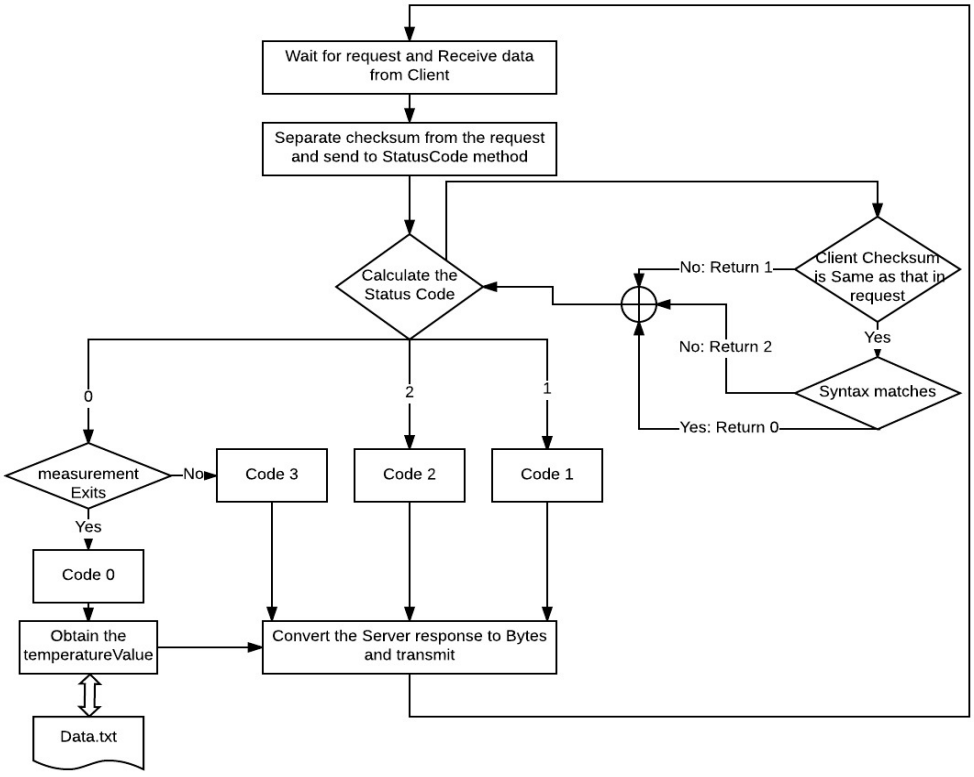


FOR THE UDP SERVER

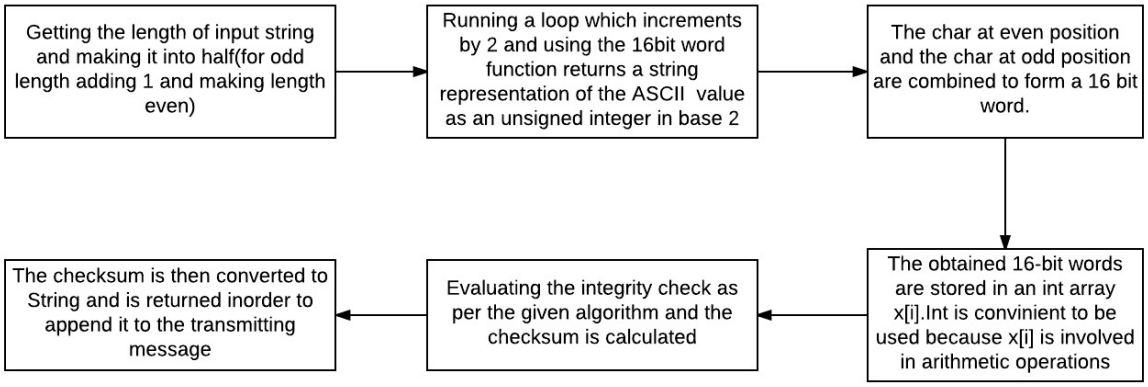



Client FuncItionality Flowchart
Random Request ID & Random measurement ID from fileReadClient method
fileReadClient method
Data.txt file
Compose request message through clientRequest method
Attach the generated checksum to request message and convert to Bytestream
Create the Datagram and open socket for transmission; timeout = 1second
Send the datagram and wait for response
timeout = timeout*2
Yes
Timeouts<4
Yes
Timeout
No
No
System.exit(0)
Reset the timeout value and timeout count
Data is received
Checksum of response is same as that supplied
No
send the same measurement ID and a new request ID
Yes
Yes
get the response code
Code 1
Retransmit
Code 0
Code 2
Code 3
No
Get the temperature value and display
Error code 2 message display
Error code 3 Message Display
System.exit(0)

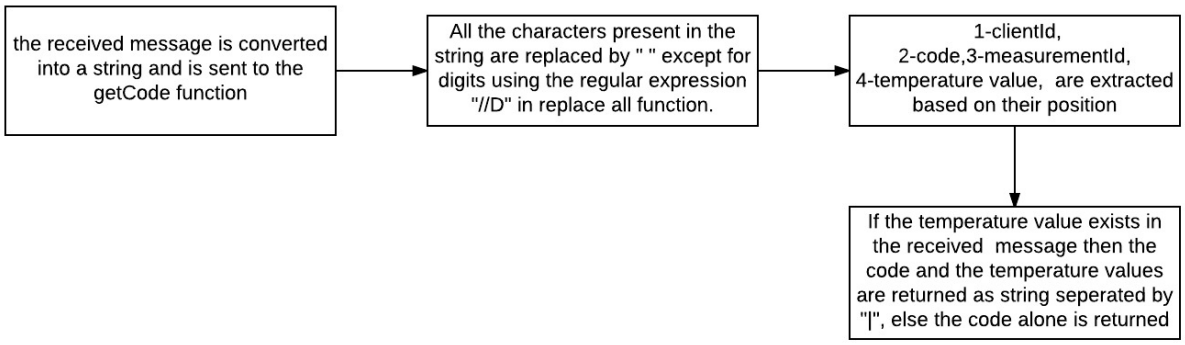
Server Functionality Flowchart



Block Diagram for Checksum calculation



Block diagram for extracting the code from the received message



SAMPLE INPUT AND OUTPUT

Case1: When server responds before the timeout and the checksum is correct/incorrect (Code 0 and 1)

Client Side:

```
client request message:
<request>
    <id>10754</id>
    <measurement>23197</measurement>
</request>
37445
Sending request to Sever
Receiving response from the server
the temp value is: 97.17
==============================
client request message:
<request>
    <id>59381</id>
    <measurement>13267</measurement>
</request>
41317
Sending request to Sever
Receiving response from the server

the request Message is:
<request>
    <id>57130</id>
    <measurement>13267</measurement>
</request>
35428
sending data again
Receiving response from the server

the temp value is: 100.53
==============================

client request message:
<request>
    <id>24817</id>
    <measurement>32853</measurement>
</request>
62425
Sending request to Sever
Receiving response from the server

the temp value is: 90.40
==============================
```

client request message:
<request>
<id>16721</id>
<measurement>58071</measurement>
</request>
55933
sending data again
Receiving response from the server

the temp value is: 105.94
==========

client request message:
<request>
<id>57388</id>
<measurement>5733</measurement>
</request>
14276
Sending request to Sever
Receiving response from the server

Error: integrity check failure. The request has one or more bit errors. Do you want to re-send the message?(Yes = y/No= n):
y
the request Message is:
<request>
<id>36188</id>
<measurement> 5733</measurement>
</request>
51288
sending data again
Receiving response from the server

the temp value is: 108.78
==========

Server Side:

Receiving request from client
the client message is:
<request>
<id>10754</id>
<measurement>23197</measurement>
</request>
37445
the server response is:
<response>
<id>10754</id>
<code>0</code><measurement>23197</measurement>
<value>97.17</value>
</response>
34768

Client Touch-based

Receiving request from client
the client message is:
<request>
<id>57130</id>
<measurement>13267</measurement>
</request>
35428
the server response is:
<response>
<id>57130</id>
<code>0</code><measurement>13267</measurement>
<value>100.53</value>
</response>
9787
Client Touch-based

Receiving request from client
the client message is:
<request>
<id>24817</id>
<measurement>32853</measurement>
</request>
62425
the server response is:
<response>
<id>24817</id>
<code>0</code><measurement>32853</measurement>
<value>90.40</value>
</response>
58235
Client Touch-based

Receiving request from client
the client message is:
<request>
<id>16721</id>
<measurement>58071</measurement>
</request>
55933
the server response is:
<response>
<id>16721</id>
<code>0</code><measurement>58071</measurement>
<value>105.94</value>
</response>
2532
Client Touch-based

Receiving request from client
the client message is:
<request>
<id>57388</id>
<measurement>5733</measurement>
</request>
142763
the server response is:
<response>
<id>57388</id>
<code>1</code>
</response>
43955
Client Touch-based

Receiving request from client
the client message is:
<request>
<id>36188</id>
<measurement> 5733</measurement>
</request>
51288
the server response is:
<response>
<id>36188</id>
<code>0</code><measurement>5733</measurement>
<value>108.78</value>
</response>
48705
Client Touch-based

Receiving request from client

Case 2: Non-Existing ID Output (Code 3)

Client Side:

client request message:

<request>
<id>6994</id>
<measurement>300</measurement>
</request>
63064
Sending request to Sever
Receiving response from the server

Error: non-existent measurement. The measurement with the requested measurement ID does not exist.

Server Side:

Receiving request from client
the client message is:
<request>
<id>6994</id>
<measurement>300</measurement>
</request>
63064
the server response is:
<response>
<id>6994</id>
<code>3</code>
</response>
41067
Client Touch-based
Receiving request from client

CASE 3: When the syntax is wrong in the received message(Code 2):

Output when the syntax is wrong (<measurement/> is removed):

Client Side:

client request message:
<request>
<id>61786</id>
<measurement>11647
</request>
45632
Sending request to Sever
Receiving response from the server

Error: malformed request. The syntax of the request message is not correct.

Server Side:

Receiving request from client
the client message is:
<request>
<id>61786</id>
<measurement>11647
</request>
45632
the server response is:
<response>
<id>61786</id>
<code>2</code>
</response>
33861
Client Touch-based

Receiving request from client