PROJECT REPORT

(December 4, 2016)

Reliable UDP Network to Process Temperature Conversion

Submitted by

SAHIL KHANNA 114840134
ANIRUDH RATHI 114640468

Submitted to

Professor (Dr.) Zoltan Safar
ENTS 640
Masters in Telecommunication
University of Maryland College Park

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."

Anirudh Rathi **Sahil Khanna.**

Contents

Overview 3

Client Side 3

- Functions Used: 4
  - a) calculateChecksum(string): 4
  - b) integrityCheck(string): 6
  - c) getMeasurementID(): 6
- URL Diagram: 6
- Problems Faced: 7

Server Side 8

- Functions Used: 9
  - a) integrityCheck(byte, int): 9
  - b) getMeasurementValue(int): 9
  - c) syntaxCheck(string): 9
  - d) checkMeasurementIDMatch(int): 11
- URL Diagram: 11
- Problems Faced: 12

Sample Outputs 12

- Perfect transmission: 12
- Timeout: 13
- Syntax error: 13
- Integrity Check: 13

Figures:

Figure 1: Flow Chart of Client Code Process 3
Figure 2: Flow chart for the checksum code 5
Figure 3: Client Class URL Diagram 6
Figure 4: Flow Diagram for Server Side 8
Figure 5: Flow chart for the syntax code. 10
Figure 6: Server Class URL Diagram 11

Overview

The project has the aim to construct a reliable approach for User Datagram Protocol transmission. A database has been provided to the server which contains measurement ID and corresponding temperature measurement values. The client sends a measurement ID and the server returns corresponding temperature value. During this process, the server and the client perform few actions to make this transaction reliable:

- Syntax Check.
- Integrity Check.
- Valid Measurement ID check.
- Timeout
- Retransmission

Client Side

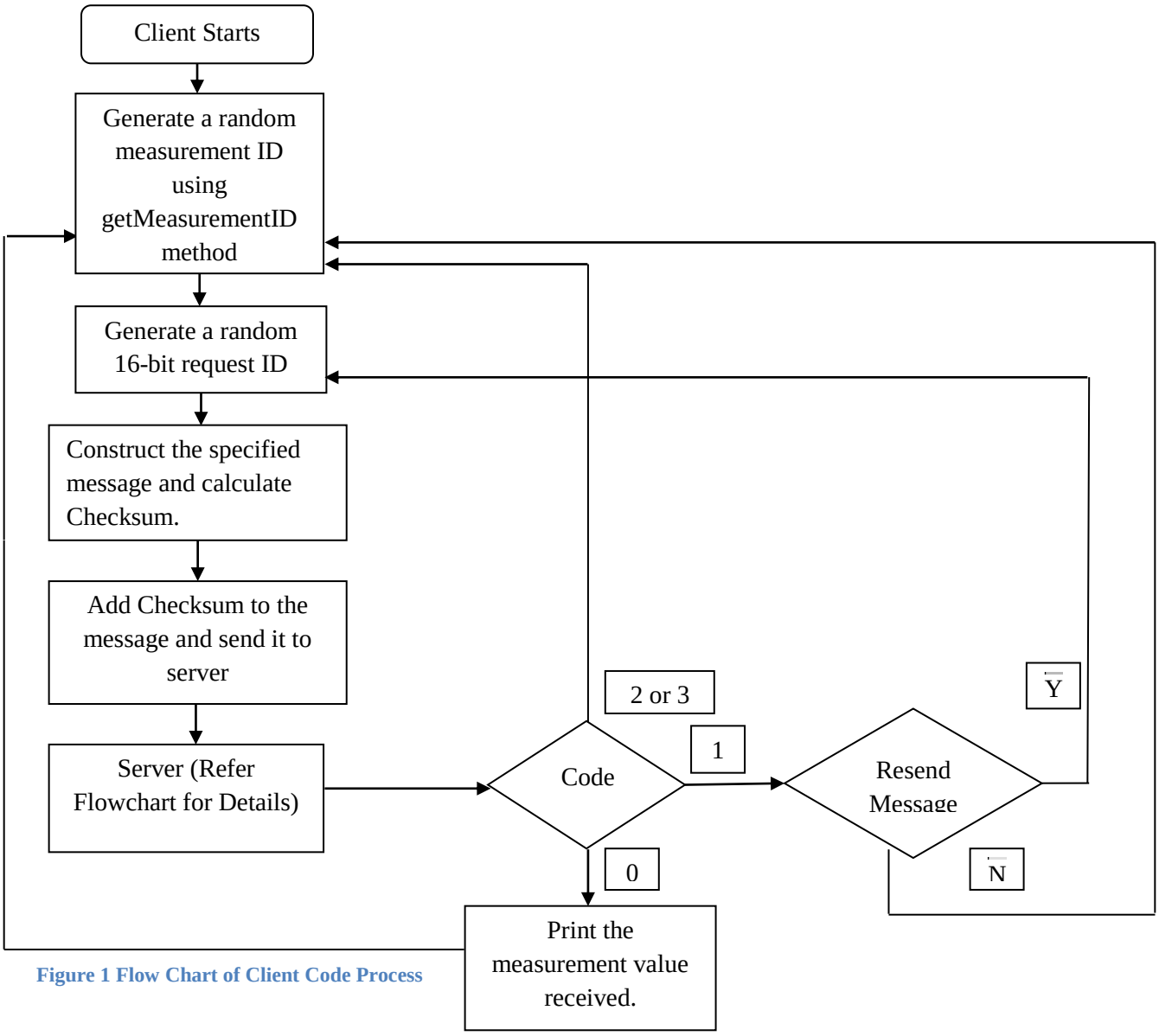

Figure 1 Flow Chart of Client Code Process

The Client side of the project is responsible for the generation of random request ID and measurement ID. The client first opens a socket and makes byte array for receiving the message from the server. Once the random request ID and the measurement ID are generated by the user, they are inserted into the message format specified in the project. This message is currently without checksum. So, the function calculateChecksum() is called in order to find it. The returned value of this function is then appended in the request message and the final message is ready to be sent to the server. The functionality of each function is explained later in the report. This message is sent to the server and a timer is set with initial value of 1 sec. If the response from the server is not received within this time frame the timeout interval is doubled and the process is repeated for 4 times (Try-Catch mechanism is used here).

At this stage, we have a response message for the server. The integrity check function is called to check if the checksum of the server message is equal to the checksum which is received from the server. If main receives false from integrityCheck() method a random request ID is generated and inserted into request message and sent to server again and the above process happens.

If the integrityCheck() returns true, all the spaces and new line feed are removed from this message and the code which implies what operation has been executed at the server is found out. According to the value of the code, corresponding action is taken. If the code in the response message is 1, the user gets a chance if he/she wants to resend the message.

- ## Functions Used:

a) calculateChecksum(string):

This function has been created to find the checksum of the request message.

- Parameter for this function: String Type
- Return type: Integer

The request message is passed to the function as input and all the blank spaces, carriage feed(\r) and new line feed in this message are removed using the string function called **replaceAll (char to be replaced, char to be inserted in the place of replaced char).** Then the entire string message is converted into character array using the function **toCharArray().** As mentioned in the project, if the length of this array is odd, the length of the array named characterValue (which stores the ASCII value of character) is incremented by 1 and 0 is added as the last element. Then using for loop, value of first two elements in the characterValue array is stored in integer named mostSignificant and leastSignificant. These integer are further converted to the string using **toString()** function. It is important to make sure that the leastSignificant byte has the length of 8. If no, a while loop is used to make satisfy this condition. Lastly, the entire string which has the most significant and least significant bytes is converted in integer array using **Integer.parseInt()** function. Using the iteration code provided in the project description the sum of the entire array is figured and returned to the main. Find the flow char below.

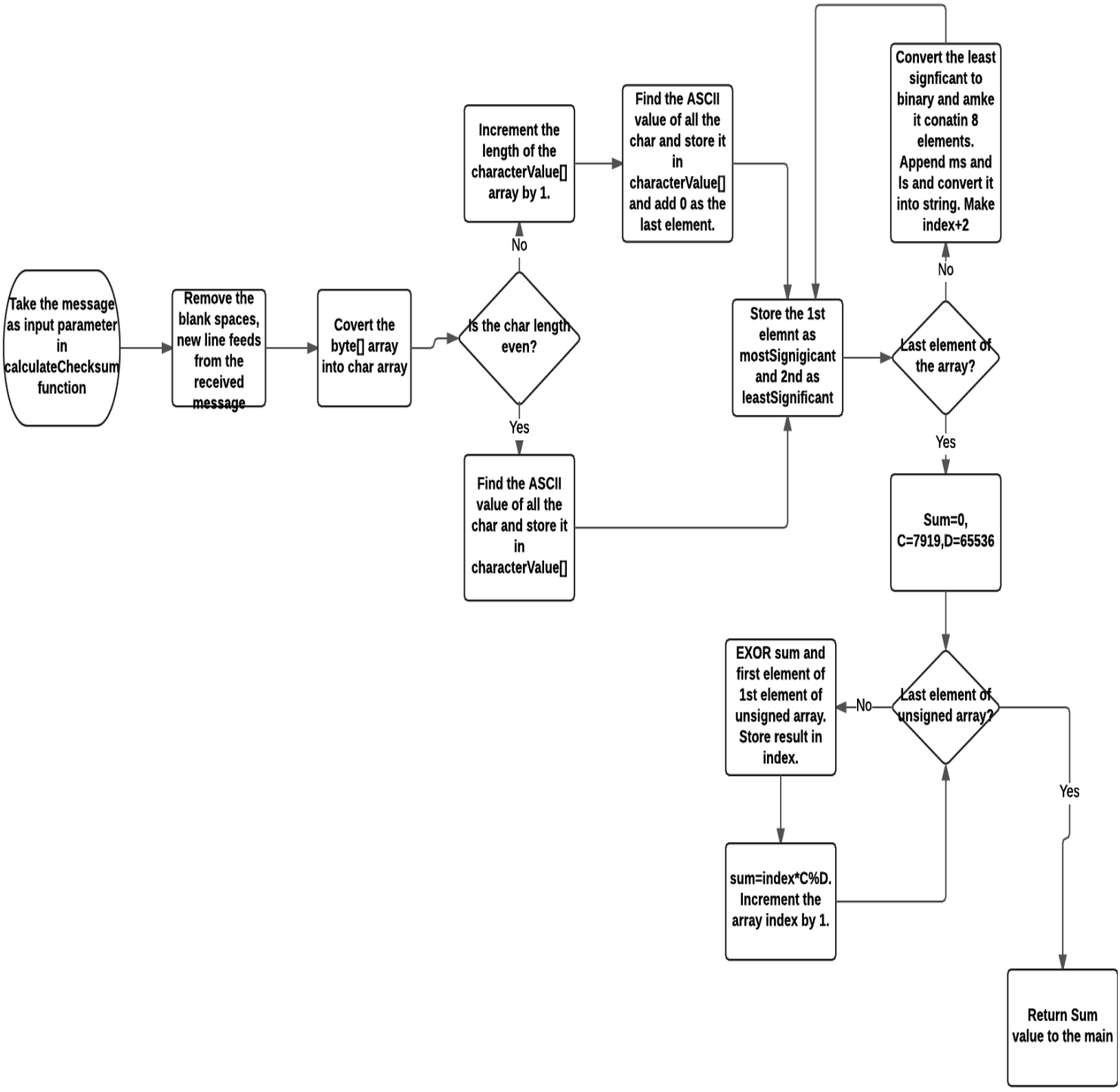

Figure 2 Flow Chart for Calculating Checksum

b) integrityCheck(string):

This function has been created in order to find if the checksum in the response message and the checksum of response message are same.

- Parameter for this function: String type
- Return type: Boolean.

Here, the response message received from the servers is taken as input parameter. **replaceAll()** is used in order to remove all unwanted character (e.g. spaces). Then the index of checksum which was appended by the server is found using the **lastIndexOf(char)** function. A char sequence of the this appended checksum is made using the function **subSequence(start index, last index)**. This char sequence is then converted into string and integer in a single line using **toString()** and **Integer.parseInt()**. The appended checksum value is stored in variable named checksum. Further the string without checksum is passed to the function calculateChecksum() and return value from this function is stored in variable named messageChecksum. If the checksum value and message checksum value is equal true is returned to the main else return false.

c) getMeasurementID():

This function has been created in order to read the data file and store it into arrays named id[] and value[]. Further a random number is generated between 1 to 100 and is passed as index to the id[] array. Thus a random measurement id can be obtained. This function also uses the **split()** function in order to break the line read into 2 parts where the space occurs.

- Parameter for this function: none
- Return type: integer

• URL Diagram:

Client
- iD: int[ ] – static - value: double[ ] – static - FILE_LOCATION: String - static, final
- getMeasurementID(): int - static - integrityCheck(requestMessage: String): boolean - static - calculateChecksum(messageWithoutChecksum: String): int - static

Figure 3: Client Class URL Diagram

Let's see a short explanation of this URL diagram. Three variables are constructed to store measurement IDs, measurement values and location of the file. The getMeasurementID() method returns the randomly generated measurement ID by using random index in the iD[] array. The integrityCheck() method checks whether the response checksum is equal to the calculated

checksum of response message, if they are equal it returns true otherwise false. The calculateChecksum() method is used to calculate the checksum of the message which is later appended to the request message and then send it to the server.

- ## Problems Faced:

a) Deciding the size of the byte array while sending and receiving packets:

This was one issue where we spent some time to figure out an approach which will take care of all the conditions. Then we decided to use a while loop and start traversing the array from the end and stop at the index where first valid char is found and made a copy of all the valid characters.

Server Side

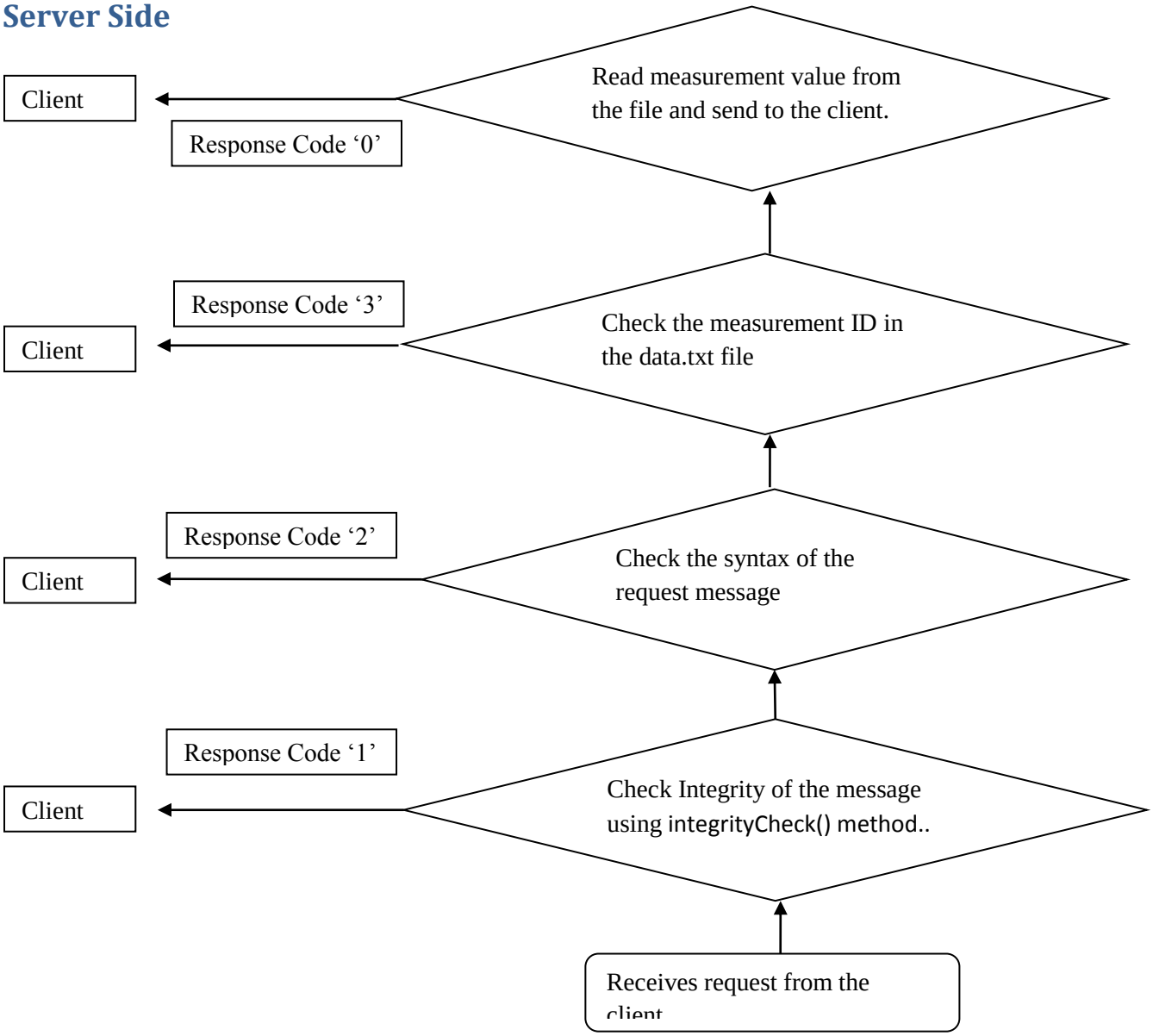

Figure 4: Flow Diagram for Server Side.

The server is responsible to carry out a number of operations. Let's have a deeper insight into the functionality of server. Firstly, the server socket is opened and byte array of size 200 is made to store the received message. Then a while loop is used to find the empty spaces in the byte array named receiveMessage[]. The while loop starts from the last character of the array and goes on till it finds the first valid element. The loop is terminated as soon as a valid entry is found and the copy of this array is made using function **copyOf()**. Further the integrity check is done using the function integrityCheck(). If integrity check fails then send valid response. If integrity check is successful perform the syntax check. If syntax check fails send valid message to client. If syntax check is successful, read the data file and find if the measurement ID (from **checkMeasurmentIDMatch()**) is valid. If yes, send the corresponding temperature value

(obtained from **getMeasurementValue()**) to the client with specified message format and code (0 in this case). Let's have an in depth look into various functions used:

- ## Functions Used:

a) integrityCheck(byte, int):

This function has been created in order to find if the checksum appended in the response message and the checksum of response message are same.

- Parameter for this function: byte type, integer type
- Return type: Boolean.

Here, the response message received from the client is taken as input parameter along with byte array length. All the blank spaces in this array are removed using while loop and array copy is made as explained earlier. **replaceAll()** is used in order to remove all unwanted character (e.g. spaces). Then the index of checksum which was appended by the server is found using the **lastIndexOf(char)** function. A char sequence of the this appended checksum is found using the function **subSequence(start index, last index).** This char sequence is then converted into string and integer in a single line using **toString()** and **Integer.parseInt().** The appended checksum value is stored in variable named checksum. Further the string without checksum is passed to the function calculateChecksum() (whose function same as the function explained in Client part) and returned value from this function is stored in variable named messageChecksum. If the checksum value and message checksum value is equal true is returned to the main else return false.

b) getMeasurementValue(int):

This is a very simple function. It returns the measurement value of the measurement ID passed.

- Parameter for this function: integer type
- Return type: integer

c) syntaxCheck(string):

This function is used to check if the message received from the client is in correct format. After every successive if loop the deeper if loop is checked and true is retuned after all the conditions are satisfied. It makes use of **substring()** and **indexOf()** method in order to divide and check.

- Parameter for this function: string type
- Return type: Boolean.

Along with the format of the string this function also checks if the measurement ID and request ID are 16 bits unsigned number. A try catch loop is also used here, in order to make sure that the measurement ID is not converted to a character during transmission. If this happens, ParseInt() method will throw a Number format exceptions and code will catch the exception by returning false. Let's have a look at the flow chart.

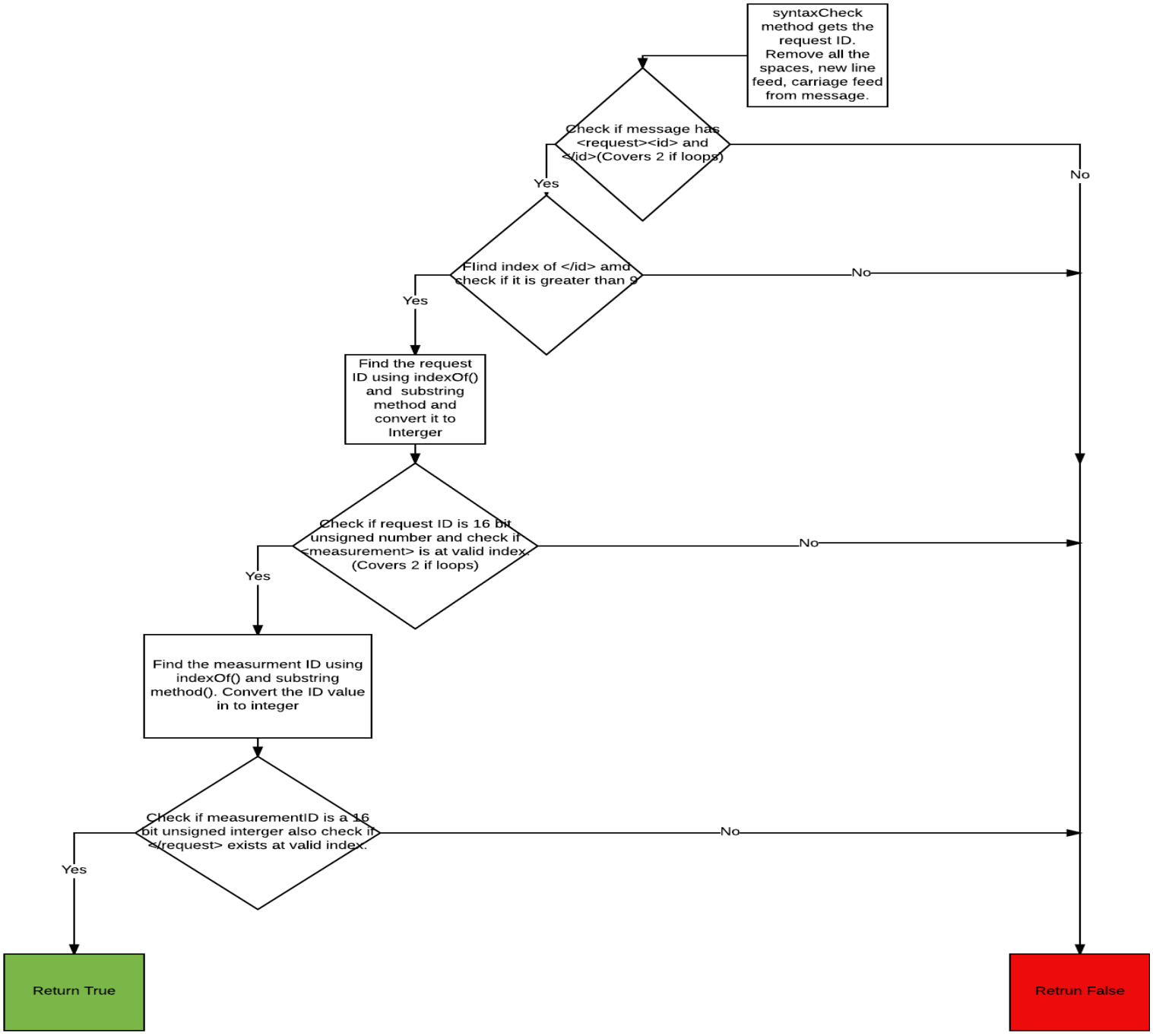

Figure 5: Flow chart for the syntax code.

d) checkMeasurementIDMatch(int):

This function has been created to satisfy the valid measurement ID condition in the project.

- Parameter for this function: integer type
- Return type: Boolean.

Initially the method reads the data file using BufferReader and FileReader and further it stores it into 2 arrays named id[] and value[] which happened in client as well. It also checks if the measurement ID passed to this function is equal to the any of the measurement ID stored in id[]. If yes, return true else false.

• URL Diagram:

Server
- iD: int[ ] – static - value: double[ ] - static - FILE_LOCATION: String - static, final
- getMeasurementValue(measurementID: int): double – static - checkMeasurementIDMatch (measurementID: int): boolean - static - syntaxCheck(aRequestMessage: String): boolean - static - integrityCheck(requestMessage: byte[], packetLength: int): boolean - static - calculateChecksum(messageWithoutChecksum: String): int – static

Figure 6: Server Class URL Diagram

Let's have a look at the URL of the server class. This class contains three variables which are used to access and store the data file and it also contains 5 methods to perform the functionality as mentioned in the project.

The variable iD[] array contains all the measurement IDs and value[] array contains all the measurement values from the data files. The FILE_LOCATION variable is used to store the location of the data file in the computer. The method getMeasurementValue() is used to return the measurement value from the data stored in the arrays. It takes a measurement ID as an input, then checks the index of the ID in iD[] array and it finally return the corresponding index value from the value[] array. The method checkMeasurementIDMatch() returns true if it finds the measurement ID in the iD[] array otherwise returns false. The syntaxCheck() method compares the request message format with specified message, if the received message at server does not matches the specified format it return false, otherwise it further checks whether the measurement ID and request ID are 16 bit unsigned number. The integrityCheck() compares the calculated checksum and appended checksum in the request message are same. The calculateChecksum() method here is similar to the client side calculateChecksum() method.

- **Problems Faced:**

a) **Problems in Syntax check function when the measurement ID is converted to a character.**

Various hurdles came across while handling the syntax check function. This is one of them and we handled this using try catch loop.

b) **Problem in figuring out the start index of the checksum appended in the client message.**

We used lastIndexOf(“>”) function in order to find the index of the checksum. What if there is a syntax error and > is disappeared? We handled this case by using a while loop which starts reading character from last and stops when you get a letter or symbol. It also returns false if there is no digit at the last place when it reads for the first time.

Sample Outputs

- **Perfect transmission:**

Output Screen:

```
The measurement value corresponding to measurement ID 26857 is: 75.01
The measurement value corresponding to measurement ID 62709 is: 93.98
The measurement value corresponding to measurement ID 58039 is: 76.30
The measurement value corresponding to measurement ID 14216 is: 103.60
The measurement value corresponding to measurement ID 20330 is: 82.82
The measurement value corresponding to measurement ID 58206 is: 93.24
The measurement value corresponding to measurement ID 38751 is: 81.49
The measurement value corresponding to measurement ID 47533 is: 85.20
The measurement value corresponding to measurement ID 17731 is: 70.08
The measurement value corresponding to measurement ID 8701 is: 71.69
The measurement value corresponding to measurement ID 35268 is: 105.40
The measurement value corresponding to measurement ID 36926 is: 93.04
The measurement value corresponding to measurement ID 3805 is: 95.90
The measurement value corresponding to measurement ID 18926 is: 101.99
The measurement value corresponding to measurement ID 63631 is: 61.35
The measurement value corresponding to measurement ID 58292 is: 69.28
The measurement value corresponding to measurement ID 20489 is: 81.19
The measurement value corresponding to measurement ID 18926 is: 101.99
```

- ## Timeout:

When port numbers are mismatched or server is not running.

Output Screen:

Connection Failure!!

- ## Syntax error:

We introduced various syntax error in the message i.e removing a single character, a symbol or new line feed or entire word or combination of multiple changes and desired output was displayed. First time output is shown below.

Output Screen:

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

- ## Integrity Check:

We made changes in the calculating the checksum at the server side by chances some values for e.g. C or D and got the desired output. First time output is shown below.

Output Screen:

Integrity check has failed at server end, would you like to resend enter: y/n
y