Networks and Protocols
Project

Overview

A Java application consisting of a client and a server using Java's UDP sockets. The reliability of the communication have been ensured by the application using UDP's unreliable data transfer services. The protocol use integrity check, timeout and retransmission as needed for purpose.

The server serves temperature measurement values (in degrees Fahrenheit) to the client. The client send's 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. The measurement IDs and the corresponding measurement values are in the file `data.txt`. The file contains 100 lines, and each line represents a separate measurement ID/measurement value pair. The first number in each line is the measurement ID, which is a 16-bit unsigned integer. The second number is the corresponding temperature measurement value, which is a fixed-point decimal number with two fractional digits. The server uses the content of data file when serving the client's request.

Message Structure

The messages sent between the client and the server is text-oriented and consist of human-readable character sequences. Thus, they are assembled as `String` objects and then converted to byte sequences before sending them out through the UDP sockets.

An example request message is shown below.

```
<request>
	<id>7542</id>
	<measurement>2851</measurement>
</request>
1472
```

The whole request message (except for the checksum) is enclosed within the `<request>` opening and `</request>` closing tag pairs. Inside, there is two more message elements: the request ID, enclosed within the `<id>` and `</id>` tag pairs, and the measurement ID, enclosed within the `<measurement>` and `</measurement>` tags. The request ID is a 16-bit unsigned integer randomly generated by the client, and its role is to identify the request/response message pairs. The measurement ID represents the ID of the temperature measurements, and it is one of the measurement IDs provided in the `data.txt` file. The message elements is included in the request message in this order. The last message element is the checksum, which is a 16-bit unsigned integer, and its role is to provide bit error detection capability. The checksum is calculated over all characters in the request message (except for the checksum), starting with the "<" character of the opening `<request>` tag, and ending with the ">" character of the closing `</request>` tag. The details of the checksum calculation will be provided later in a separate section.

When processing the request and response messages, white space (new lines, spaces, tabs) is ignored. For example, this is also a valid request message:

```
<request><id>


7542      </id>
       <measurement>       2851



       </measurement></request>36023
```

The server's response message in case of no errors is assembled according to this figure:

```
<response>
       <id>7542</id>
       <code>0</code>
       <measurement>2851</measurement>
       <value>98.30</value>
</response>
748
```

The whole response message (except for the checksum) is enclosed within the `<response>` opening and `</response>` closing tag pairs. If there are no errors (normal response), there are four more message elements inside (in this order): the request ID, enclosed within the `<id>` and `</id>` tag pairs, the response code, enclosed within the `<code>` and `</code>` tags, the measurement ID, enclosed within the `<measurement>` and `</measurement>` tags, and finally, the measurement value, enclosed within the `<value>` and `</value>` tags. The request ID is the same as the one received from the client with the request message; the server simply echo back the request ID. The response code indicate the outcome of the received request, taking up values according to the table below.

Response code	Meaning
0	OK. The response has been created according to the request.
1	Error: integrity check failure. The request has one or more bit errors.
2	Error: malformed request. The syntax of the request message is not correct.
3	Error: non-existent measurement. The measurement with the requested measurement ID does not exist.

If there are no errors (normal response), the value of the response code is 0. The measurement ID is the same as the request's measurement ID, and the measurement value is the temperature value that corresponds to the request's measurement ID based on the content of the `data.txt` file. The checksum also be included at the end of the message, calculated over the whole message (except the checksum element), starting with the "<" character of the opening `<response>` tag, and ending with the ">" character of the closing `</response>` tag.

If some error has occurred, the response message only contain the request ID and the response code elements, and the response code indicate the problem encountered based on the response code table above. The checksum also be included at the end. Here is an example error response message:

```
<response>
       <id>9999</id>
       <code>3</code>
</response>
4675
```

Protocol Operation

First, need to set up the client and the server UDP socket parameters (IP addresses, port numbers etc.) so that the two sides could communicate with each other. Then, the client application work according to the following description.

1. Select one of the available measurement IDs to receive the corresponding temperature measurement value from the server.
2. Generate a random request ID, assemble the request message as a sequence of characters (as a `String` object), including the integrity check field, convert it into a byte array and send it to the server. also start a timer at this point with initial timeout value of 1 second.
3. Wait for the response from the server. If no response arrives and the timer expires, resend the request, and the timeout interval is doubled at each timeout event. After the 4th timeout event, the transmitter declare communication failure and print an error message on the screen. If the response arrives after a number of timeout events and retransmissions, the timeout value and the timeout counter is reset to their initial values.
4. Calculate the integrity check value for the response message and compare it with the integrity check value supplied in the response message. If they are not equal, go back to step 2 and repeat sending the request message.
5. Check the value of the response code. If no errors occurred (i.e. if it is 0), then display the measurement value received from the server on the screen. If some errors occurred, display an error message on the screen indicating the problem that has occurred (e.g. malformed request message). If the error was that the request message's integrity check failed (response code 1), ask the user if he/she would like to resend the request and go back to step 2 and resend the request if needed.
6. Go back to step 1 and do the whole request/response sequence in a loop until the application is terminated.

The server application go through the following steps:

1. Wait for requests on the specified port.
2. Upon reception of a request message, verify the integrity of the message by calculating the integrity check value and comparing it with the integrity check value provided in the request message. If they are different, send back an error response with response code 1.
3. If the integrity check passes, convert the received message into a character sequence (a `String` object) and check the syntax of the request message. This include 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 also be checked, e.g. that the number between the `<measurement>` and the `</measurement>` tags is a 16-bit unsigned integer.   White

space in the message are ignored during this step. If the syntax check fails, send back an error response with response code 2.

4. Look up the measurement value that corresponds to the requested measurement ID, as defined in the `data.txt` file. If there is no measurement value for the requested measurement ID, send back an error response with response code 3.
5. Assemble the response message with response code 0 including the requested measurement value and the integrity check value, convert it into a byte array and send the response back to the client.
6. Go back to step 1 and wait for another request message.

Integrity Check

The integrity check for a message consisting of a sequence of characters (not considering the integrity check) is calculated as follows. First, the character sequence is converted to a sequence of 16-bit words (unsigned integers). The ASCII code for the 1st character is the most significant byte of the 1st 16-bit word, the ASCII code for the 2nd character is the least significant byte of the 1st 16-bit word, and so on, each 16-bit word contain 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 be set to zero. Then, assuming that the N so obtained 16-bit words are $\mathrm{x}[0], \mathrm{x}[1], \ldots, \mathrm{x}[\mathrm{N}-1]$, the 16-bit integrity check value will be given by the following algorithm:

```
1. $\mathrm{S} = 0$
2. for $\mathrm{i} = 0, 1, 2, \ldots, \mathrm{N}-1$ do
     2a. $\mathrm{index} = \mathrm{S}\ \mathrm{XOR}\ \mathrm{x}[\mathrm{i}]$
     2b. $\mathrm{S} = (\mathrm{C} \cdot \mathrm{index})\ \mathrm{MOD}\ \mathrm{D}$
```

where S and index are 16-bit unsigned integers, XOR stands for the bitwise exclusive or operation, MOD stands for the modulo operation, $\mathrm{C} = 7919$ and $\mathrm{D} = 65536$. Will need to make sure that the integer multiplication and the modulo operations use enough precision to avoid overflow. Once the algorithm completes, the variable S will contain the 16-bit checksum value as an unsigned integer. This checksum value is converted to a character sequence (e.g. the checksum value 1234 is converted to `“1234”`), and it is appended to the message to be sent. The last character of the checksum is the last character of the message.

At the receiver, the same process is repeated, and the checksum value is re-calculated. Then, the calculated checksum value is compared with the received checksum value, and if they are not equal, Assume that there are some bit errors in the received message (i.e. integrity check failure).