**Documentation MQTT**

The language of choice for our team to use as a base to code the MQTT protocol in Julia was the implementation in embedded C. The biggest initial issue the team faced was figuring out exactly what the protocol was and where to begin.

The way the program is intended to work is based on modules. By using modules we allow the program to have new elements to be easily added and altered rather then risking the need to change the entire program to implement or change a single feature of the protocol.

The most useful thing we used to test the code was the inclusion of the load.jl file. This allowed us to run the Julia program in command line which we found to be much more helpful. To run our project in command line the user must have Julia set as an environment variable. Once this is done the user should open up command line and cd to the location of the folder that contains all the project code and type Julia to start Julia in this location.

After Julia starts the first thing to do is to type in include(“load.jl”). this will load everything necessary in order to run the program smoothly. The next step is to set up the client variable. To do this type client=MQTTClient() and then hit enter. This sets the variable client to be MQTTClient() which can be found in the mqttData file. It sets all the variables necessary to be able to tell the server that it’s a client.

The next part is wherethe user can see the client starting to interact with the broker. Next the user will type in MQTTConnect(client) into the command line. This will call the function MQTTConnect and pass the client into the function as a parameter. MQTTConnect is a function in the file mqttClient.jl. Two parameters are required but only one is passed when calling the function because the other is already assigned a value when the function is called.

The first thing the function does is check that the client is connected. This is done using an if statement. Next a try block is then entered in the try block the program first sets a number of variables in order to set up the keep alive function so that the connection will stay open. The first major step in the program is to serialize the connect packet . the program gets the length of the serialized connect packet first. This is done by calling the function serializeConnect and assigning it to a variable(len). Three variables are needed to be passed into the serialize connect. Once in the serializeConnect the program first sets the variable ip = 1 as the count cant start at 0. The program then needs the connect length which is achieved by passing in options as a parameter. There must then be a check to make sure the packet length is smaller then the buffer. The header then needs to be set by passing “connect” as the message type to the function mqttheader. The next few steps are to get the length of the serialize. For this we need to know the size of the write buffer, the encode packet length, the version of mqtt. Getting each value and incrementing the variable ip. Next the flags need to be set. These depend on the reserved flags and other factors like options and passwords. Next the program will add the length of the flags and the options to the variable ip. There is also a check for username and password to add to the length. All the writebuf functions used in serialize connect can be found in the file tools.jl. None of these are complex. They are designed to take in variables and return a int.

The next part of the connect function is to send the now serialized connect packet. This function is in the mqttsend.jl file. Three variables(Timer, client, and length) are sent into the function. A while loop is used to to send the packet as long as the timer is not expired and the variable sent is <= the length of the packet. Next the program checks of the packet has been successfully sent or not. The program now uses waitfor in order to receive the response from the broker. If the expected response is received the program can continue. This is checked using function cycle.

Cycle calls the function readPacketTemp. To determine the packet type. If the packet type matches what is expected(CONNACK). The function waitfor expects cycle to return the same packet type as wait for passed on as a parameter then the program continues on to deserialize the CONNACK. The program first checks if the header is a CONNACK. Next it must be checked if the packet length is correct or not. This is done using function decodePacketLength. Next session and rc are set and returned. Finally, the program checks the client is connected and returns rc.

The next function is subscribe. To run this in command first create a string (s = “test”). Next the function is called by typing MQTTSubscribe(client, s, FireAndForget). First the program checks the connection and enters a try block. The subscribe needs to be serialized similar to how connect was. All the parameters passed in are passed to functions to find their length and add them to the length(ip – gets returned). the next step is to send the packet. This is done using the same function that was used in the connect packet. The program must now wait for the response from the sever to make sure Subscribe was successfully received. A suback should be received (the same as connack for connect). The program now must deserialize the suback. This is done using the deserializesubsck. In the deserialize the program first decodes the packet length

Once the connection is fully established we then moved onto the subscribe packet. We initially ran into an issue with the subscribe packet that we didn’t understand. After trying to run the code in a number of different ways it was discovered that there was no function created for the handler to pass into the subscribe packet. This was found to be the issue as when an empty function was written and then called into passed into the subscribe function then their appeared to be no issues(apart from the fact that we didn’t know what we needed to make the function do), but at least it meant we then knew what the issue was. Once all the variable are successfully passed into the Subscribe function the first thing the function does is check to see if the client is still connected. If the client is not connected then the will return an MQTTCLIENT\_FAILURE. As mentioned before if any part of the protocol does not happen in the intended order or with the corrected expected values then the request will fail next the client must serialize the subscribe and get the length of the subscribe so that it can inform the broker when it sends the subscribe packet how large the packet is. This is important as the broker must allocate resources in order to be able to handle the subscribe packet correctly and efficiently. Along with length the client must also send the broker the connection itself and a timer. The reason for the timer is to ensure the broker knows how long the client is going to wait before the connection will time out. The client will now wait patiently for the broker to respond to the subscribe packet with a SUBACK packet. The client also expects to receive back the client and the timer that it sent. Using waitfor allows the client to wait for the response before allowing the program to proceed as it cannot successfully carry out the next step without successfully completing the previous step in the chain. It is expected by the client to receive a SUBACK packet in response to the SUBSCRIBE packet being successfully sent. When the client receives the response from the broker the client must check that the response is what was expected(SUBACK). To do this the client must deserialize the response and check 2 things. First it will check the QoS of the response it receives. If the QoS = 0x80 then rc is set to MQTTCLIENT\_FAILURE, this represent a failed topic subscription. If QoS is set to anything that is not 0x80 then this means the subscription was a success and the rc is then set to MQTTCLIENT\_SUCCESS. In order to display any issues that might occur in this part of the function we have surrounded it with a try catch statement. This way any problem that may be unexpectedly encountered we can use a print statement in the catch to say exactly where the error occurred and print what the error was.

To date we have successfully managed to get the program to send the connect packet and receive and verify the CONNACK packet.

The next step in the assignment is to check create the UNSUBSCRIBE packet. This is important for the client as there may be a loss of interest in a subscribed topic. There are only 2 parameters required to be passed into the UNSUBSCRIBE packet, the connection and a string containing the topicfilter. The function first checks to make sure the connection is still valid. We then use a try catch to attempt to first serialize the UNSUBSCRIBE packet and then send the packet. Async is then used to wait from the broker. The program will then try to deserialize the response so we can then check that it is the response we expect. If the response is incorrect then an error message will be printed by the catch. The error message will be non specific so in order to narrow more accurately determine any source of error we have put in a number of print statements to find out exactly how far into the function the program gets before an error is caught.

DISCONNECT

Next on the list is the Disconnect Packet. Once again using the print statements it was determined that there were 2 errors present in the packet. However initially it was not fully determined why the errors was occurring. There was also another issue with a function that the disconnect was calling. The disconnect was calling serializeDisconnect which at the time did not exist. This issue was easy to fix. The prints tell us that there is an undefined variable error in the attempt to serialize the disconnect. It also tells us that rc is not defined in the function. Using prints tells us that all the variables do in fact hold values. This makes the errors even more confusing for us to try and identify as the variables cannot be undefined if they hold values. After working on the disconnect for a while the next error was that the timer as not being assigned its value properly. The disconnect is fully working now. However there is still one issue. The rc is not set properly in the disconnect so it has been hardcoded for now in the disconnect