**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 where the 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.

It should be noted that throughout this program if either the client or broker receives something unexpected or cannot process something the way MQTT is designed to do then the connection between the client and the broker will terminate.

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. In our case we set it so the connection should stay open indefinitely. 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 an 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 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, readInt gets the size in bytes. ReadByte gets the QoS. Deserialize then returns the packet ID and the QoS. Subscribe then checks the value of the granted QoS. Following this the is the catch to detect errors, and the return of rc and the unlock of mutex.

Next is the disconnect. To run the disconnect type into the command line MQTTDisconnect(client). The first thing disconnect does is set rc and check the client is still connected. Mutex is then locked before entering the try block. The program now needs the length of the serialized disconnect packet. The same idea as the other functions that get the length of a serialized packet. SerializeDisconnect is called and variable len its set to the value of the serialized packet. The program will then set the timer variable before passing the parameters required to send packet. The same as anything else that gets sent down the socket. Send packet keeps sending packet down the stream until the end of the packet(len) is reached. Also still performs the check for the size of the length compared to what is actually sent. If sent exceeds length then an error is thrown. After the disconnect is successfully sent the program will then set is connected = 0, set rc = success, unlock mutex and return rc.

The publish function caused the least amount of issues. The first thing, same as every other function is to check the client is still connected. The program then locks mutex and enters a try block which first checks QoS. The length of the serialized publish packet is then retrieved. Serialize publish works by getting the length of the packet, checking that it ios not bigger then the buffer size, setting the header to publish and then getting the length of the encoded packet, the header and the topic message. They are all added together and returned. Next up is setting the timer and sending the packet. It is sent using the same sendPacket function as Connect, Subscribe etc. The client then waits for a reply from the broker. The publish needs either of 2 conditions listed to be true in order to deserialize the acknowledgement of the publish.

The deserialize of the acknowledge starts with setting the header and decoding the packet length. Next is to get the value from readInt. Finally return the packet type(header) and packetID. Provided there are no errors the program will return rc and unlock mutex.

**Structure:**

Major parts of code structure is currently as follows:

Functions in mqttClient: MQTTConnect

MQTTSubscribe

MQTTUnsubscribe

MQTTPublish

MQTTDisconnect

GetNextPacketID

Waitfor

Keepalive

Cycle

Function in mqttSend: sendPacket

ReadPacketTemp

GetPacketLength

Functions in serialize: getConnectLength

GetDisconnectLength

SerializeConnect

GetPublishLength

SerializePublish

GetSubscribeLength

SerializeSubscribe

SerializeUnsubscribeLength

SerializeAck

SerializeDisconnect

DeserializeConnack

DeserializePublish

DeserializeAck

DeserializeSuback

DeserializeUnsuback

Functions in tools: getpacketlength

EncodePacketLength

DecodePacketLength

ReadString

ReadInt

ReadPayload

ReadByte

Writebuf (X5 all different)

Functions in interface: mqttreadTemp

MqttWrite