CS631 ASSINGMENT-I PDF REPORT
0n detailing the architecture of client /master interaction using Modbus Protocol,Message Format,and for each task what I have done in the Client program

Gargi Sarkar
Roll no-21111263

September 2021

Contents

1 Introduction

Modbus is a data communication protocol ,which has now became a de facto standard communication protocol for its simplicity as well as robustness. It was published by Modicon(now Schnider Electronic) in 1979. It is basically a Master(client) Slave(server) protocol. The Master polls devices operating as Slave and the Slaves, being polled send information in protocol format (Application Specific Data Units). After receiving the information the master processes them and uses it for control actions.Though it was originally implemented as an application level protocol to transfer data over a serial layer, its now got extended and included implementation over wireless communication and TCP/IP networks.

In critical infrastructure Modbus is widely acceptable as its easy to implement and maintain , royalty free and does not ask for countable restrictions. In this scenario the Master is typically a PLC,PAC,RTU,DCS and the Slaves are often the field devices.
Here in this report I am going to describe the basics of the Modbus Protocol in the beginning , then I will describe how Modbus TCP works, and in the end, the final report consisting the details of the task I have been asked to perform.[1]&[2]

2 The Request-Response Cycle

Basically in this Master-Slave protocol, the Master only can initiate the interaction, which gives a Master a power of domination to control the Slave behavior, which is helpful for few particular cases and the Slave device cannot volunteer any information ,it has to wait to be called.

In Modbus , the request made by the Master is a layered set of data. the 1st layer is the application data unit(ADU), which defines what type of Modbus Protocol is going to be used. There are three ADUs; ASCII, RTU and TCP/IP. the choice of ADU depends on desired physical network, number of devices in the network, number of masters in the network. Within each ADU there is protocol data unit (PDU) which is the core of the protocol. Each PDU consists of a function code and associated data. the function code is nothing but the command sent to the Slave. Except PDU, ADU is consisting address and error checksum, where address is mainly the address of the devices it is intending for and the checksum information allow the recipient to detect transmission error.

3 Types of Registers

In Critical Infrastructure it is important to the devices to have values defined as input and outputs. For this purpose the data values in Modbus are divided into four ranges namely, Coils,Discrete Inputs,Holding register,Input Registers. A Slave can define up to 65536 elements in each range. The coils and registers each having a read only table and a read write table.
Each Coil(discrete output) is 1 bit , Boolean type , assigned data address between 0000 to 270E(hexadecimal) and read-write type, and having the register number ranging from 1-9999
each discrete input is 1 bit, Boolean type, read only type, assigned to the data address between 0000 to 270E, and having the register number ranging from 10001 to 19999.
Each Input Register is one word=16 bits,read only type, and assigned to data address between 0000 to 270E and having the register numbers ranging from 30001-39999
Each Holding Register is 1 word=16 bits, read and write type, assigned to data address between 0000 to 270E and having the register number ranging from 400001-49999

The coil/register numbers can be thought of a location names since they do not appear in the actual messages. The data addresses are used in the messages. For example , the first Holding Register, 40001, has the data address 0000. The difference between these two values is called the offset. Each table has a different offset 1, 10001,30001,40001[3]

4 Function Codes

The second byte send by the Master is the Function Code. The number tells the slave which table to access and whether to read from or write to the table. For example 01 denotes to read Coil ,05 denotes to write single Coils, 16 denotes write multiple Holding Registers. [4]

5 Modbus Error Codes

In a normal response, the server repeats the exact function codes as described as above. When there is an error, it replies with the requested function code plus 128 . For example if the requested function code is 1 , it will reply sending function code as 121. the error could be many types:

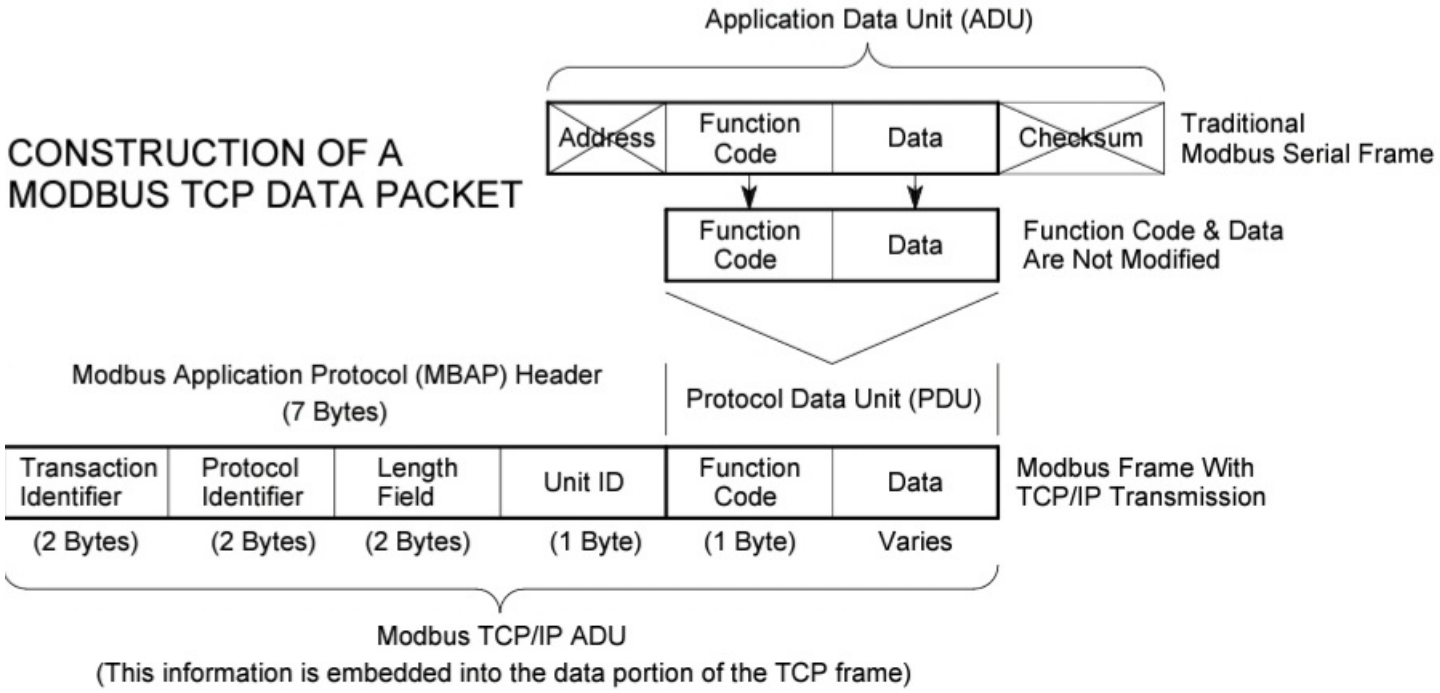
- The data address could be outside the acceptable range,i.e, the register does not exist.
- The value contained in the requested function is not acceptable to the salve.

- There could be slave devices failure when performing the requested action
- Slave could be busy and engaged processing a long duration command. the master have to try again later time in this case.
- The gateway could be misconfigured or overloaded

Done with enough introduction! Lets talk about Modbus TCP.

6 Modbus TCP

Modbus TCP/IP is simply the Modbus RTU Protocol with a TCP interface that runs on Ethernet . Communication over Modbus TCP can happen between different types of devices,for example a Modbus TCP/IP client and server devices connected to a TCP network ,or the interconnection devices like a bridge, router or gateway for interconnection between the TCP/IP network and serial line sub network which permit connections of Modbus serial line client and server end devices. In practice Modbus TCP embeds a standard Modbus Data Frame, without Modbus checksum as shown in the below figure.



The Modbus commands and user data are encapsulated into the data container of a TCP/IP without being modified in any way. As the standard Ethernet TCP/IP having link layer checksum method(Which provides data integrity) , here its not needed to use error check (i.e the checksum).
Lets talk about the above figure. Its clear that the Function Code and data are absorbed in the original form. Thus , a Modbus TCP/IP ADU takes the form of 7 byte header, containing Transaction Identifier , Protocol Identifier, Length Field , Unit Identifier and the PDU. The fields in Modbus application protocol (MBAP) as follows:

- Transaction /Invocation Identifier: It is 2 bytes long and used for transaction pairing when multiple messages are sent along the same TCP con-

nection by a client without waiting for a prior response.

- Protocol Identifier: Its also 2 bytes long. Its always having the value 0 for Modbus services.
- Length: Its also 2 bytes long which indicates about the byte counts of the remaining field.
- Unit Identifier: It is 1 byte long. It identifies a remote server on non TCP/IP network. It basically ranges from 00 to FF . In typical Slave application Unit Identification gets ignored and just echoed back in the response.

The Modbus TCP ADU , embedding into data field of TCP/IP frame is basically sent via port 502.Modbus masters and server listen and receive Modbus data through port 502.

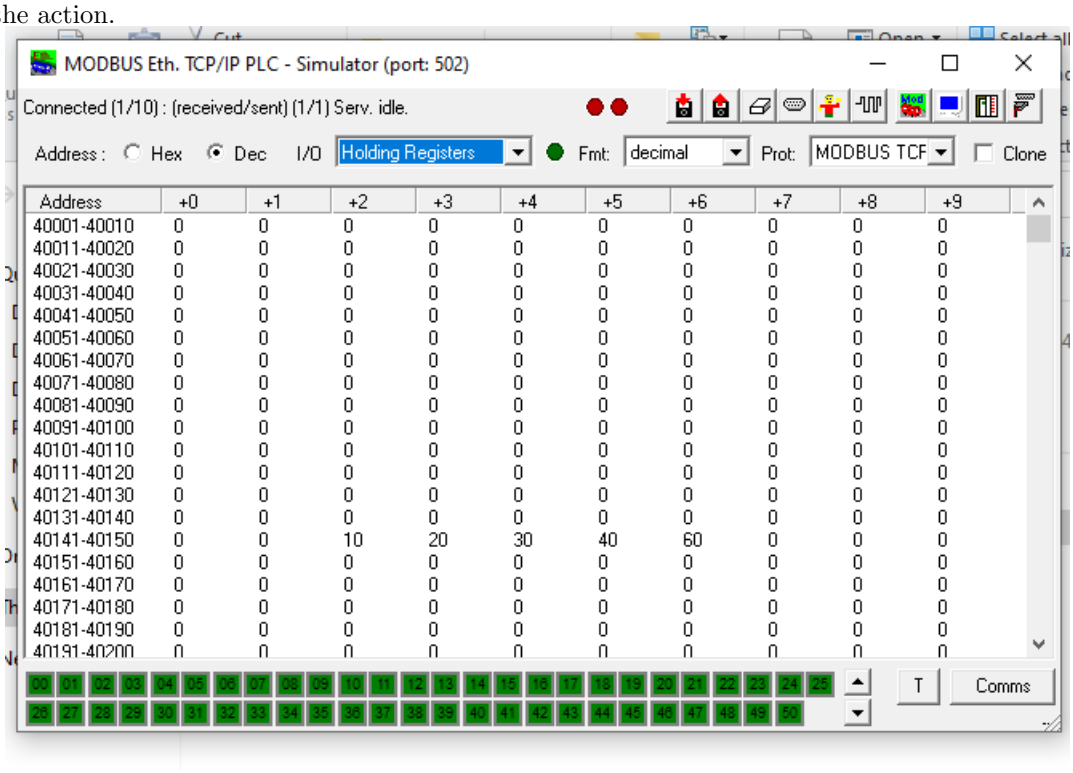
In traditional serial Modbus a Client can send only one request at a time and have to wait for an answer before sending the second request. In Modbus TCP devices devices may send several request to the same server without waiting for the response using the help of the transaction Identifier. Its basically a sequence of number which gets incremented in each request. The maximum number of transaction is ranging from 1 to 16 based on devices. The Unit Identifier was introduced for having communication between Modbus TCP/IP Ethernet devices and traditional Modbus serial devices by using a gateway to route traffic from one to other. All Modbus TCP/IP connections are point to point communication path between devices . It requires a source address, a destination address and a connection id in each direction. A TCP connection is needed to be established before message can be sent via Modbus tcp/ip. The master initiates the connection. [5]

7 Description of performed activities for the given assignment

To observe practically how Modbus TCP communication protocol works , as directed I have created two virtual machine, one to assign as platform to the master(client) other to the slave(server). For the server I chose windows 10 machine and installed the ModRSSim simulator there. For clients I chose ubuntu machine. In order to assign different virtual machines different NAT addresses so that they appear on different portion of the network, and they can see one another but not be seen from the host , I created a custom NAT Networks. After doing this , to contact with the master simulator I wrote a client program in python ,the details python client is attached with this directory.[6]&[7]

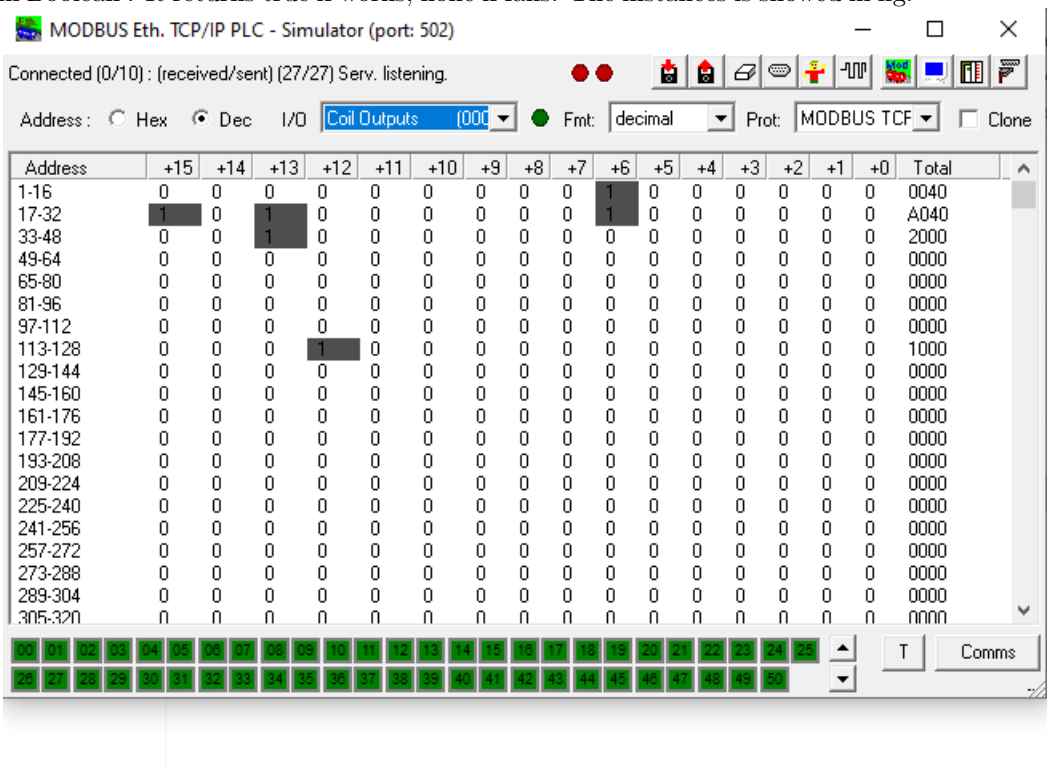
7.1 Writing values in the Holding Register

To write values in the holding register I used the Modbustcp function write_registers which is taking the parameters register address; an integer value ranging from 0 to 65535 and a list;register values to write. It returns the value true if its successful otherwise none. The fig is showing in the observed instance of the action.



7.2 Writing value in Coil

used the ModbusTcp function write_coil where I passed the address of the bit(integer) (ranging from 0 to 65553 as mentioned earlier) and the bit value in Boolean . It returns true if works, none if fails. The instances is showed in fig.



7.3 Reading the values from the Holding Registers

Here I used the ModbusTcp function read_holding_registers and passed the parameter register address(int) ranging from 0 to 65553 and register number(int) i.e the number of registers to read. It returns register list if works , otherwise none.

7.4 Reading values from the Coil

Used the function read_coil where i passed two parameters, bit address(int)ranging from 0 to 65535 and number of bits(int) to read. It returns a list if works, none otherwise

7.5 Displaying the function code for reading and writing for coils

Called the functions for reading and writing the coils. using predefined attribute function_code fetched the function codes from a stored response object.

7.6 Displaying the function code when an error is occurred

When a Modbus Slave recognises a packet and determines an error present there in the request, it will return an exception code reply instead of data reply. The exception reply consists a copy of the function code with a high bit set, i.e, if the function code was 16, then it will return 128+16=144. For the each individual above task I produced some error, then run the program , it returned the as same as describe as above.

7.7 Changing the simulator setting

From the simulator setting if we disable the write option for the coils and register, and then try to write on the register when the simulator is installed in a different machine it was not working but when it is installed in the same machine it was working same as before.

8 Difficulties faced to complete the task

In the beginning days when I started assignment I faced difficulties to understand and perform few things but later by collecting data from google sites, discussing with classmates, and with the guidance of the TAs I successfully manage to complete the task I was given . From having the lower ram capacity to run two virtual machine together ,how to make two virtual machine to see one another, how to find the the error code from the response to complete question no. 6, everything got sorted out!cheers!:)

9 Learning from the Homework

- To complete the assignment , as advised and guided I have followed all the resources , PDFs, links etc to get a understanding how communication protocol like Modbus works , different aspects of it, how it has been improved prior to demands and security day by day.
- As using this Modbus master simulator we can query data from a device, we can use the same to develop a Modbus slave device.
- The Modbus TCP protocol implementation contains multiple vulnerabilities, that could allow an attacker to perform reconnaissance activity or

issue wrong commands. This protocol specification does not include any authentication mechanism for validating communication between the master and slave. For this reason , and unauthenticated remote attacker can easily breach into any slave via a Modbus master.

References

[1] https://en.wikipedia.org/wiki/Modbus.

[2] http://www.modbus.org.

[3] J. McConahey, Using modbus for process control and automation, part 1, Appl Autom A12–A14 2 (2011).

[4] http://www.lammertbies.nl/comm/info/modbus.htm.

[5] http://www.ni.com/white-paper/7675/en.

[6] G. Collins, Pymodbus documentation (2013).

[7] (https://sourceforge.net/projects/modrssim/files/).