Mechanics Transport Communications
ISSN 1312-3823 (print)
ISSN 2367-6620 (online)
volume 17, issue 1, 2019
Academic journal http://www.mtc-aj.com article № 1752

CREATING ARCHITECTURE AND SOFTWARE OF EMBEDDED SYSTEMS WITH CONSTRAINED RESOURCES AND THEIR COMMUNICATION TO THE IOT CLOUD

Neven Nikolov, Ognyan Nakov
n.nikolov@tu-sofia.bg

Department of Computer System and Technology,
Technical University of Sofia, Sofia 1000,
BULGARIA

Key words: *Embedded systems, IoT Cloud, Microcontroller, Software, MQTT, Esp32, PIC18f47K40, net core MVC*

Abstract: *This article described creating architecture, software and communication of IoT devices with constrained resources to IoT Cloud. Also is described soft-ware realization and architecture of IoT Cloud. Used physical environment is WiFi network, which is wide widely used in home environment. In article is shown way for connection the constrained microcontrollers to centralized Cloud, sending and receiving data between them.*

1. INTRODUCTION

Nowadays, the number of Internet related devices - IoT - is growing steadily. They are most often connected to a centralized cloud system that provides data storage, processing, and management through dedicated algorithms. Cloud computing [1] [2] [3] provides storage of data through databases, processing of received data, as well as remotely merging IoT devices on the part of the client. The IoT devices themselves can be anything like both use and use, and owned resources. IoT devices can be comprised of limited resources microcontrollers, such as RAM, permanent flash memory and, of course, CPU processors. With the use of micro controllers with sufficiently large resources, any methods and algorithms can be applied to securely and securely transmit data between the IoT embedded system and Cloud Structure. When using constrained resource microcontrollers [4], it is sometimes impossible to implement a TCP stack or high-level data protocol. It is possible to use protocols with low or no protection, but this only reduces the security of the system. The use of a microcontroller with limited resources is justified by their low cost, or the microcontrollers themselves are designed to perform responsible operation such as phase or frequency drive control or real time pulse intercept. To solve the real-time cloud-based system management synchronization problem, it is necessary to connect the microcontroller with limited resources to another microcontroller that has enough RAM to implement a TCP stack and to fit a TLS certificate for encrypting data to and from the Cloud.

2. REVIEW

The tested IoT embedded system is used to control real-time fan speeds and other consumers such as a heater and a water pump. The communication between the IoT embedded system and the cloud structure can be accomplished by using different physical environments and protocols [6]. Physical data transfer environments are determined by the environment in which the devices will be located. Physical environments can be by using a wired connection, such as Ehternet, as well as a wireless connection to WiFI, LoraWan, ZigBee, 2G, 3G, and so on. This article discusses cases in which a WiFI wireless connection is used, which physical environment is quite widespread. Cloud structure communication uses the built-in ESP32 wrover system, which has a 32-bit dual-core Tensilica Xtensa LX6 processor running at 240Mhz and a 520kb SRAM capable of using TLS and SSL encryption certificates. The ESP32 has WiFi radio module and 4Mb Flash memory. The PIC18f47K40 microcontroller, which has an 8-bit RISK architecture processor running at 64Mhz, is used to control the readers and read the sensors. Real-time fan speed control uses a 8-bit PIC18F14K22 micro controller, with only 512 bytes of RAM. The three microcontrollers are compared in Table 1. For communication between the three microcontrollers, a UART data bus is used.

The MQTT protocol [5], which operates on the Publisher / Sub-scriber principle, is used to build communication between Cloud Structure and IoT. Each device can be subscribed to a topic. The server part uses the .net core MVC technology with MariaDB database.

Table 1. Comparison between ESP32 wrover and PIC18F47K40.

Microcontroller	CPU	Flash Memory	RAM	WiFi
ESP32 wrover	8 bit, Risk, 64Mhz	4Mb	520kbytes	yes
PIC18F47K40	32 bit, 240Mhz, Dual Core	128Kb	3728 bytes	no
PIC18F14K22	8 bit, Risk, 64Mhz	16Kb	5120 bytes	no

3. ARCHITECTURE

The microcontrollers communicate with each other in accordance with principles described in this chapter. A UART bus was used to communicate between them. ESP32 takes care of establishing communication with the Cloud and direct data exchange with PIC18F47K40. Both devices play the role of Master and Slave, Master being PIC18F47K40 and Slave ESP32. A diagram of the microcontroller connection in the IoT device is shown in Figure 1. The construction of the communication is described in Chapter 4.

The PIC18F14K22 microcontroller reads the data output from PIC18F47K40 to ESP32. The format of the protocol also includes the parameters needed for the revolutions of the electric motor. The microcontroller is responsible for the phase control of the motor.The construction and description of the protocol are described later in chapter IV. The communication protocol used between ESP32 and Cloud Structure is MQTT.

For the Cloud structure [7] [8] , a DELL PowerEdge R510 server was used with the CentOs 7 operating system, where MQTT Broker Mosquitto was used to receive and transmit data from the devices and to the server that subscribed to all the threads on all the devices it supports. The technology used is the .net core MVC, which uses the C # programming language. The database used is MariaDB. The architecture that uses the .NET core server is the MVC, which has three layers - Model View Controller described in Chapter 4. A cloud structure summary scheme is shown in Fig. 2.

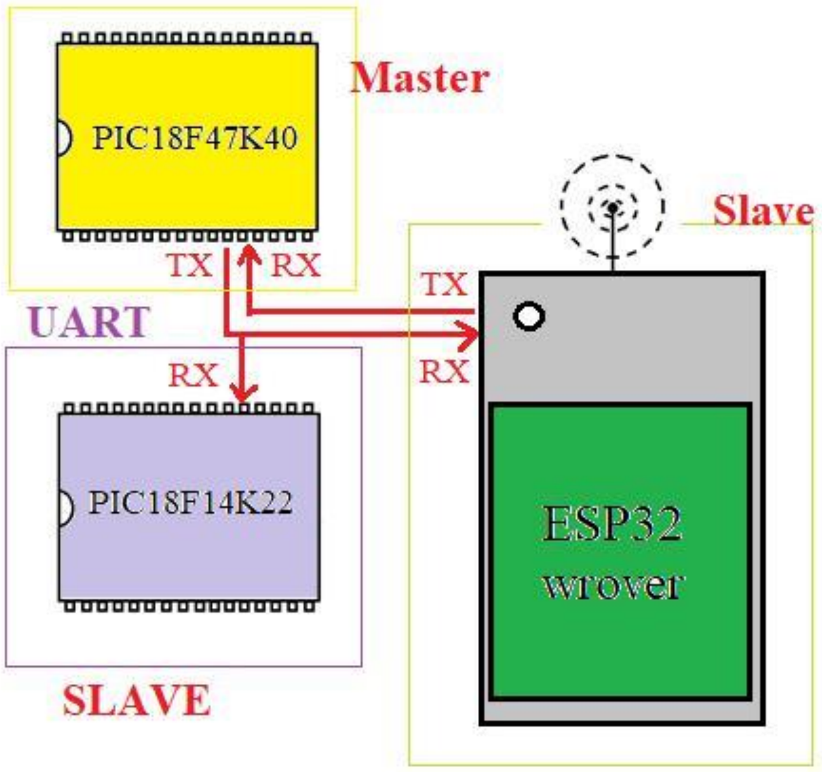


Fig. 1. Communication between ESP32, PIC18F14K22 and PIC18F47K40 using UART bus in IoT Embedded Device

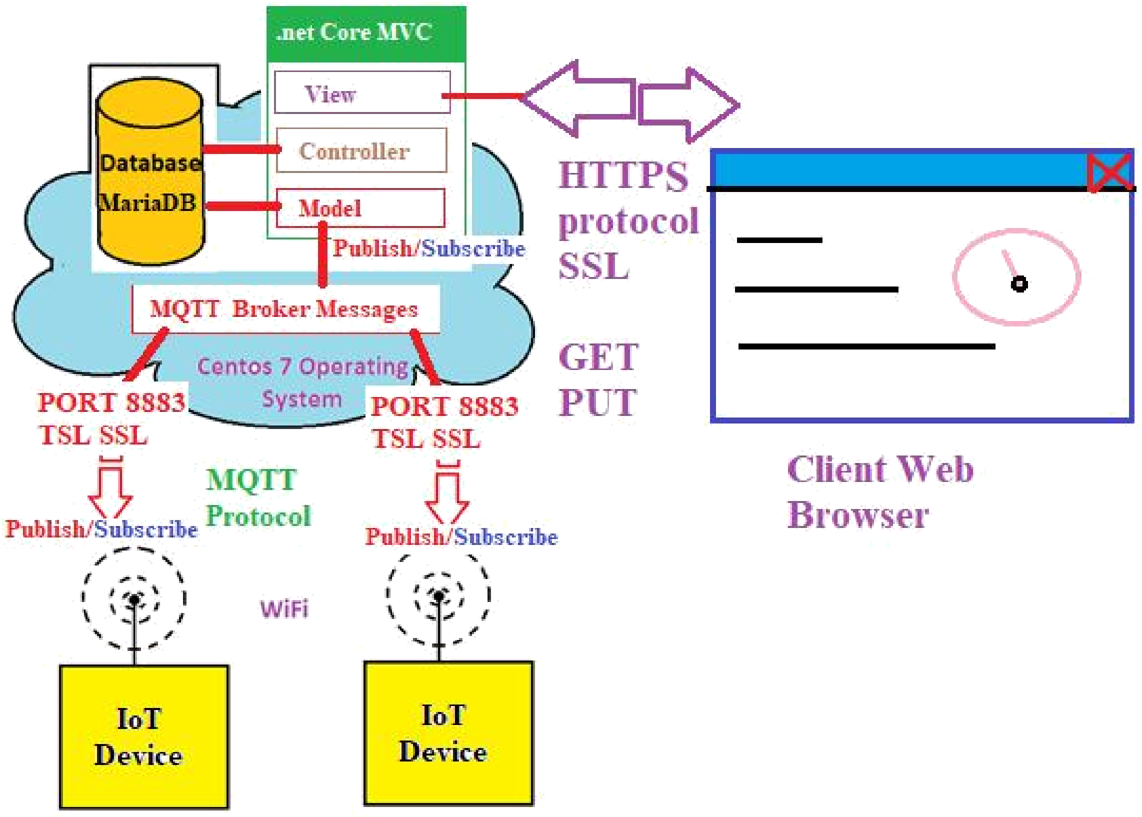


Fig. 2. Communication between Cloud structure and IoT Embedded Devices

4. SOFTWARE REALIZATION

The low-level programming languages C and C ++ were used for the software implementation of the IoT device. The IoT Cloud software implementation uses the .net core MVC technology, the programming language being C #. The ESP32 program implementation includes multi-threaded programming, reading and writing data over the UART interface, logical part, and receiving and transmitting data to IoT Cloud. The programming language used is C ++, with Arduino IDE being used for development. For communication between ESP32 and IoT Cloud, the MQTT protocol was used, using the Pub-SubClient library. Three threads were used to send data via UART bus, receive data over the UART bus, and receive IoT Cloud data:

```
returnValue = pthread_create(&ReceiveUARTthread, NULL,                    (1)
ReceiveUART, NULL);
returnValue = pthread_create(&TransmitUARTthread, NULL, TransmitUART,
NULL);
returnValue = pthread_create(&LogicThread, NULL,Logic, NULL);
```

To send data to IoT Cloud, use the "client.publish (mqtt_topic_SEND, JSON-messageBuffer)" feature in the void loop () function. The void loop () function plays an endless loop in the main thread. For the communication of ESP32 to IoT Cloud, the MQTT protocols are used, defining data acquisition and data submission topics, Table 2. Fragments of the program code are shown below:

```
const char* mqtt_topic_SEND        = "Tx000001";                         (2)
const char* mqtt_topic_RECEIVE     = " Rx000001";
void loop() {
client.subscribe(mqtt_topic_RECEIVE);
if (client.publish(mqtt_topic_SEND, JSONmessageBuffer) == true) {
#if DEBUG_MODE == true
Serial.println("Success sending message");
#else
MQTTbrokerMess =  MQTTBrokerSend_Success;
```

Table 2. MQTT protocol Receive and Transmit topics.

Variable String	Device	Value Topic	Type Message	Serial number IoT device
mqtt_topic_SEND	1	Tx000001	Tx – Transmit	000001
mqtt_topic_RECEIVE		Rx000001	Rx - Transmit	000001
mqtt_topic_SEND	2	Tx000002	Tx – Transmit	000002
mqtt_topic_RECEIVE		Rx000002	Rx - Transmit	000002

Receiving IoT Cloud data is performed using the callback function:

```
void callback(char* topic, byte* buffer, unsigned int length) {          (3)
char temp[100] ;
String s;
for (int i = 0; i < length; i++) {
Serial.print((char) buffer [i]);
temp[i] = (char) buffer [i] ;
                    }
temp[length] = '\0';
          s = temp;
}
```

The UART protocol send thread is built from a non-continuous "for (;;)" cycle, and the function that writes on the serial port is "Serial.write". The delay function (1000), which delays 100ms, is used to send and receive data from the PIC18F47K40 microcontroller:

```
void *TransmitUART(void *threadid) {                                  (4)
unsigned short Counter = 0;
for(;;){
Serial.write( RingBufferTransmitUART[Counter] );
delay(1000); // delay for 100ms
if( Counter < ( RBbufferSIZE -1 ) ){
Counter ++;
}else{
Counter = 0;
}
}
}
```

The UART protocol receive thread is also built by an endless "for (;;)" cycle, the special one being that it has a byte-byte counter and traces of received byte with a integer value of " InputByteStream == 255 ". When a byte of 255 variable "InputByteStreamCount" is reset and the process starts scanning again all received bytes:

```
void *ReceiveUART(void *threadid) {                                   (5)
for(;;){
if(Serial.available()>0) {        //Checks is there any data in buffer
InputByteStream =  Serial.read();
If ( InputByteStream == 255 ){
InputByteStreamCount = 0;
}
switch(InputByteStreamCount){
case 1:{
ONoFF = InputByteStream;
}break;
/. . . . /
case 11:{
STATUsMes  = InputByteStream;
}break;
}
InputByteStreamCount ++;
}
vTaskDelay(10);
```

IoT Cloud's software implementation utilizes the .Net Core 2.2 technology that uses the MVC Model View Controler. The language used is C #, and the logical part of receiving and receiving data from the broker is in the "Program.cs" class where the main function is located. MQTT broker Mosquitto works with .Net Core 2.2 on the Centos 7 operating system. Functions “MqttClient”, “MqttMsgPublishReceived”, “Subscribe” and anothers serve for receive and transmit MQTT messages[9]. The following code shows how to perform the .NET Core 2.2 with MQTT broker:

```
public static void Main(string[] args) {                              (6)
Console.WriteLine("Waiting...");
Console.WriteLine("Starting Client");
client = new MqttClient("localhost");
```

```
var clientId = Guid.NewGuid().ToString();
client.Connect(clientId, "admin", "admin");
string[] topic = { Topic };byte[] qosLevels =
{MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE };
client.Subscribe(topic, qosLevels);
client.MqttMsgPublishReceived = Client_MqttMsgPublishReceived;
/. . . . /
```

The Connect function serves as a link to the MQTT broker. In the "Models" layer, there are classes such as "UserModel.cs", "InitializeData-base.cs", "DatabaseManager", "ErrorViewModel.cs", "JSONparce.cs", "DeviceID.cs" and "DeviceProtocolMessageSTACK.cs". The "JSONparce.cs" class contains data processing fields between ESP32 and PIC18F47K40:

```
public class JSONparce        {                                              (7)
    //          ONoFF
    public decimal  A { get; set; }
    //          TemperaturePump
    public decimal  B { get; set; }
    //          STATUs
       public decimal  C { get; set; }
    //          PwmFan
       public decimal  D { get; set; }
  /. . . . /
```

Table 3 shows MQTT communication in message format in JSON type format.

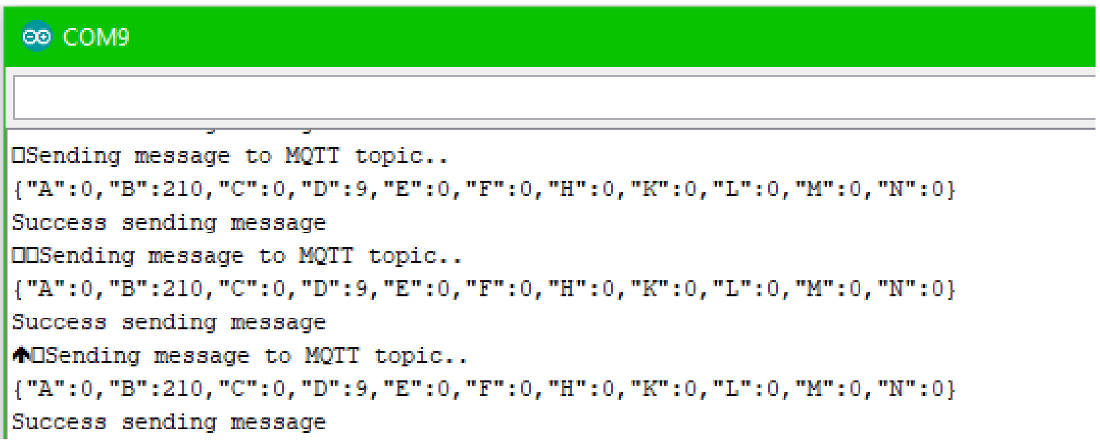
Table 3. MQTT protocol messages field definition between ESP32 and PIC18F47K40

Nimber Type	Name	Description	Example
1 Integer	A	On/ Off Consumers	1 - Start Pump 0 - Stop Pump
2 Integer	B	Temperature Pump	34*C
3 Integer	C	Power IoT device	600 W
4 Integer	D	PWM Fan	60%

The Controller layer contains the "HomeController.cs" class that communicates directly with the layers like Model and View. Layout View contains files such as "Index.cshtml" and "IoTdeviceLandingView.cshtml", which contain the FrontEnd visual part of the system in front of the user. There are fields to visualize the on or off state of the consumers, monitor parameters such as real-time temperature and switch on and off consumers.

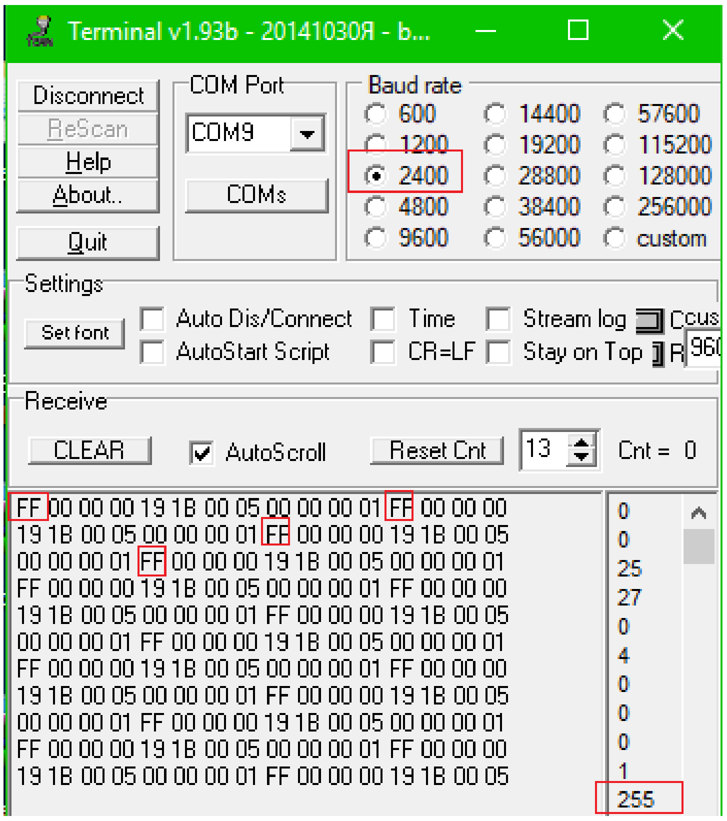
5. EXPERIMENTAL RESULTS

A USB / UART converter is used to listen to the communication between ESP32, PIC18F47K40 and PIC18F14K22. The "Terminal" program is used, whereby port 9 of the test computer is open to a 2400 baut rate data rate. In Fig. 5, the red color is enclosed by the transmission speed and value tag 255, which distinguishes each new data session from PIC18F47K40 to ESP32 and PIC18F14K22. Figure 4 shows the message sent from ESP32 to IoT Cloud, using the JSON format sent in the MQTT message. In fig. 6 are shown received message in .Net Core Server Application in console of CentOS 7.


COM9

```
□Sending message to MQTT topic..
{"A":0,"B":210,"C":0,"D":9,"E":0,"F":0,"H":0,"K":0,"L":0,"M":0,"N":0}
Success sending message
□□Sending message to MQTT topic..
{"A":0,"B":210,"C":0,"D":9,"E":0,"F":0,"H":0,"K":0,"L":0,"M":0,"N":0}
Success sending message
🡅□Sending message to MQTT topic..
{"A":0,"B":210,"C":0,"D":9,"E":0,"F":0,"H":0,"K":0,"L":0,"M":0,"N":0}
Success sending message
```

Fig. 4. Communication between Cloud structure and ESP32 – sending data in JSON format to IoT Cloud.


Terminal v1.93b - 20141030Я - b...

COM Port: COM9

Baud rate: [✓ 2400]

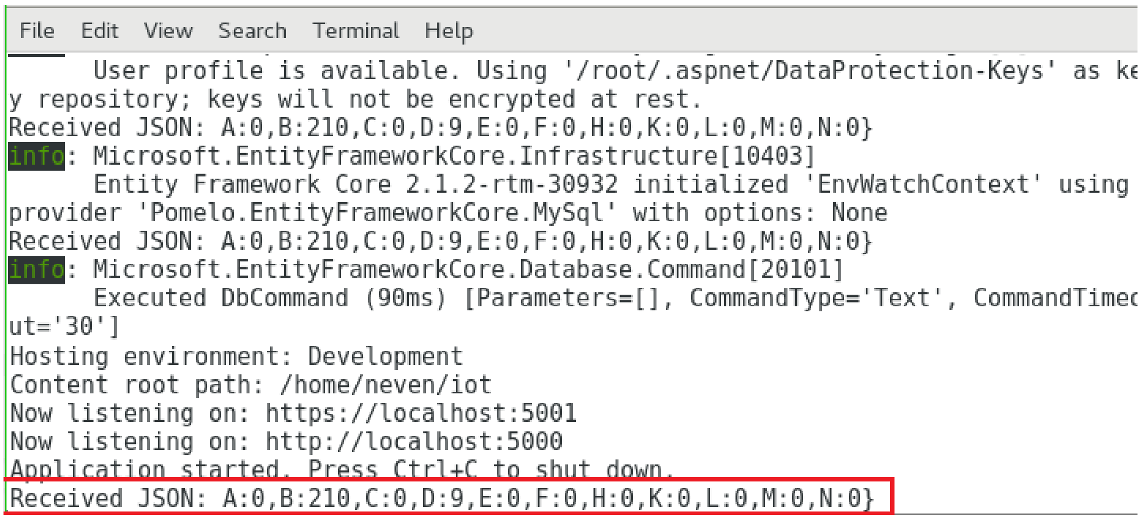
Settings: Auto Dis/Connect ☐ Time ☐ Stream log ☐ AutoStart Script ☐ CR=LF ☐ Stay on Top ☐

Receive: AutoScroll ☑ 13 Cnt = 0

```
FF 00 00 00 19 1B 00 05 00 00 00 01 FF 00 00 00
19 1B 00 05 00 00 00 01 FF 00 00 00 19 1B 00 05
00 00 00 01 FF 00 00 00 19 1B 00 05 00 00 00 01
FF 00 00 00 19 1B 00 05 00 00 00 01 FF 00 00 00
19 1B 00 05 00 00 00 01 FF 00 00 00 19 1B 00 05
00 00 00 01 FF 00 00 00 19 1B 00 05 00 00 00 01
FF 00 00 00 19 1B 00 05 00 00 00 01 FF 00 00 00
19 1B 00 05 00 00 00 01 FF 00 00 00 19 1B 00 05
00 00 00 01 FF 00 00 00 19 1B 00 05 00 00 00 01
FF 00 00 00 19 1B 00 05 00 00 00 01 FF 00 00 00
19 1B 00 05 00 00 00 01 FF 00 00 00 19 1B 00 05
```

```
0
0
25
27
0
4
0
0
0
1
255
```

Fig. 5. Communication between Cloud structure and IoT Embedded Devices



```
      User profile is available. Using '/root/.aspnet/DataProtection-Keys' as ke
y repository; keys will not be encrypted at rest.
Received JSON: A:0,B:210,C:0,D:9,E:0,F:0,H:0,K:0,L:0,M:0,N:0}
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.1.2-rtm-30932 initialized 'EnvWatchContext' using
provider 'Pomelo.EntityFrameworkCore.MySql' with options: None
Received JSON: A:0,B:210,C:0,D:9,E:0,F:0,H:0,K:0,L:0,M:0,N:0}
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (90ms) [Parameters=[], CommandType='Text', CommandTimec
ut='30']
Hosting environment: Development
Content root path: /home/neven/iot
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
Received JSON: A:0,B:210,C:0,D:9,E:0,F:0,H:0,K:0,L:0,M:0,N:0}
```

Fig. 6. Show received message in .Net Core Server Application in console of CentOS 7

6. CONCLUSION

Connecting a IoT device with limited resources to the IoT Cloud structure is a great challenge because the device needs to make secure and reliable communication in order to function properly. A certain microcontroller is used to manage the consumers in real time. But here the problem of managing it over an Internet connection is conditioned by the fact that it needs to have sufficient resources to build a security level. For this purpose another microcontroller is used, its purpose being to make a reliable and secure connection with the Cloud. This article provides secure and reliable consumer management by building an appropriate architecture.

Author Contributions: The conception in this research was created, (N.K) and (O.N); software (N.K); experimental production, (N.K); conceptualization (O.N); analysis methodology, (N.K) and (O.N); research, (N.K);

Acknowledgments: The paper is published with the support of the project No BG05M2OP001-2.009-0033 "Promotion of Contemporary Research Through Creation of Scientific and Innovative Environment to Encourage Young Researchers in Technical University - Sofia and The National Railway Infrastructure Company in The Field of Engineering Science and Technology Development" within the Intelligent Growth Science and Education Operational Programme co-funded by the European Structural and Investment Funds of the European Union.

Conflicts of Interest: The authors declare no conflict of interest.

REFERENCES

[1] A.Alshehri, R. Sandhu, “Access Control Models for Virtual Object Communication in Cloud-Enabled IoT”, IEEE Computer society, pp:16-25, 2017
[2] J. Weinman, “The Strategic Value of the Cloud”, IEEE Cloud Computing, vol.2, pp 66-70, 2015
[3] H.Truong, S.Dustdar, “Principles for Engineering IoT Cloud Systems”, vol.2, pp.68-76, 2015.
[4] S.Nastic, H. Truong, S. Dustdar, “A programming model for resource-constrained iot cloud edge devices”, Banff, IEEE international Conference on systems, 2017.
[5] Stanford-Clark, H.Linh Truong, “MQTT For Sensor Networks (MQTT-SN)”, Protocol Specification, November 14, 2013, IBM.
[6] Postscapes, “IoT Standards and Protocols”, 2018’

[7] Botta, W. de Donato, V. Persico, A. Pescap´e, “Integration of Cloud Computing and Internet of Things: a Survey”, September 18, 2015
[8] Cloud Standards Customer Concil, “Cloud Customer Architecture for IoT”, 2016
[9] eclipse.org, “C# .Net and WinRT Client” , 2019

СЪЗДАВАНЕ НА АРХИТЕКТУРА И СОФТУЕР НА ВГРАДЕНИ СИСТЕМИ С ОГРАНИЧЕНИ РЕСУРСИ И ТЯХНАТА СВЪРЗАНОСТ КЪМ IOT ОБЛАК

Невен Николов, Огнян Наков
n.nikolov@tu-sofia.bg

Технически университет – София
София 1000, бул. „Климент Охридски“ 8
БЪЛГАРИЯ

Ключови думи: *Вградени системи, IoT облак, микроконтролер, софтуер, MQTT, Esp32, PIC18f47K40, мрежово ядро MVC*

Резюме: *Тази статия описва създаването на архитектура, софтуер и комуникация на IoT устройства с ограничени ресурси към IoT облак. Също така е описана софтуерна реализация и архитектура на IoT облак. Използваната физическа среда е WiFi мрежа, която е широко използвана в домашната среда. В статията е показан начин за свързване на ограничени микроконтролери към централизиран облак, изпращане и получаване на данни между тях.*