Mechanics Transport Communications

Academic journal http://www.mtc-aj.com

ISSN 1312-3823 (print)
ISSN 2367-6620 (online)
volume 17, issue 1, 2019
article № 1749

DESIGN AND REALIZATION OF COMMUNICATION BETWEEN IOT CLOUD STRUCTURE, ANDROID APPLICATION AND IOT DEVICE USING TCP SOCKETS

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

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

Key words: *Firmware, Esp8266, Dht11, IoT device, STM32L4 discovery IoT, JSON, TCP sockets, Android, Epoll Server, IoT Cloud*

Abstract: *This article introduces the communication between IoT cloud, Android mobile application and IoT embedded systems. We present a way of transmitting temperature and humidity of the IoT cloud using TCP sockets. A JSON type data format is used between the cloud structure and the devices. Embedded systems STM32L4 discovery and Esp8266 are transmited data to IoT cloud. Mobile application realizes the representation of temperature and humidity readings from a sensor. Presented is architecture and software implementation of the experimental stage.*

1. INTRODUCTION

The Internet of Things IoT is embedded systems that are connected to the Internet. Their purpose may be different, for example for industrial purposes in factories, domestic purposes in the home, in the automotive industry, etc. IoT embedded systems have sensors and actuators. The actuators are used to control various processes, most often used to trigger their relays. Sensors can be temperature, humidity, light, radiation, and so on. , and managed sites to be robots, cars, pumps, cameras, and more. As Industry 4.0 is coming up, it is increasingly spoken about global devices or a local network, with the idea of devices sharing data. Data can be sent and processed by a centralized and non-centralized system. In this case IoT Cloud [4] [6] is used, and its software is adapted to process data from IoT embedded systems. IoT Cloud can be anywhere in the world, which is why IoT devices. IoT device data must be transferred to other IoT devices or IoT Cloud, so internet connectivity is used. They can be encrypted depending on security, especially when managing important objects, need to build a private or local network or use encrypted protocols using certificates such as SSL or TLS. The protocols used on IoT devices [2] [3] can be MQTT, CoAP, Rest Full, TCP sockets, and more. TCP sockets [1] are used to implement the experiment. The software implementation of the experimental setup is presented.

2. EXPERIMENTAL SETUP

For purpose of this research are used IoT embedded systems, which have a communication module - WiFi for Internet connection. Embedded systems are shown on

figures - Nodemcu esp8266 Fig.1 and STM32L4 discovery IoT nodes fig.2 . For measurement the temperature and humidity are used sensor DHT11. For actuators are used LED diode, which are controlled by GPIO pin of IoT embedded systems.

Figure.1 Experimental setup of Node MCU esp8266 built-in system

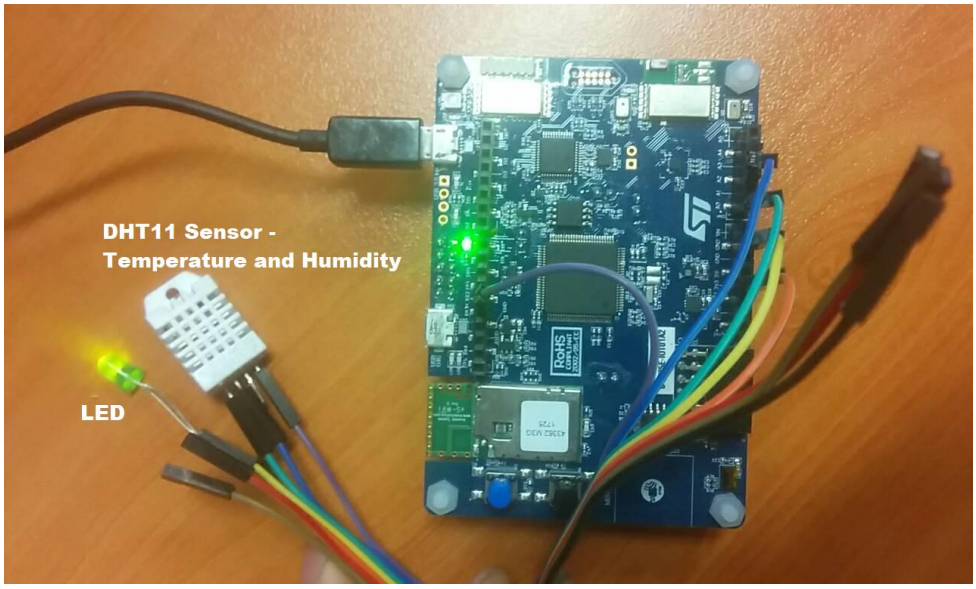

Figure.2 Experimental setup of STM32L4 discovery IoT node

3. ARCHITECTURE

The architecture are shown on fig. 3. The IoT embedded devices used are Node MCU ESP8266 and STM32F429 IoT Node. There are connected to internet through WiFi network to Global Internet. IoT cloud [5] [14] is connectet to Global Internet and he is realized with

server Dell Power Edge R510, Cent OS 7 operating system and C++ Epoll server Application. Sensor and LED diode connection are showed.

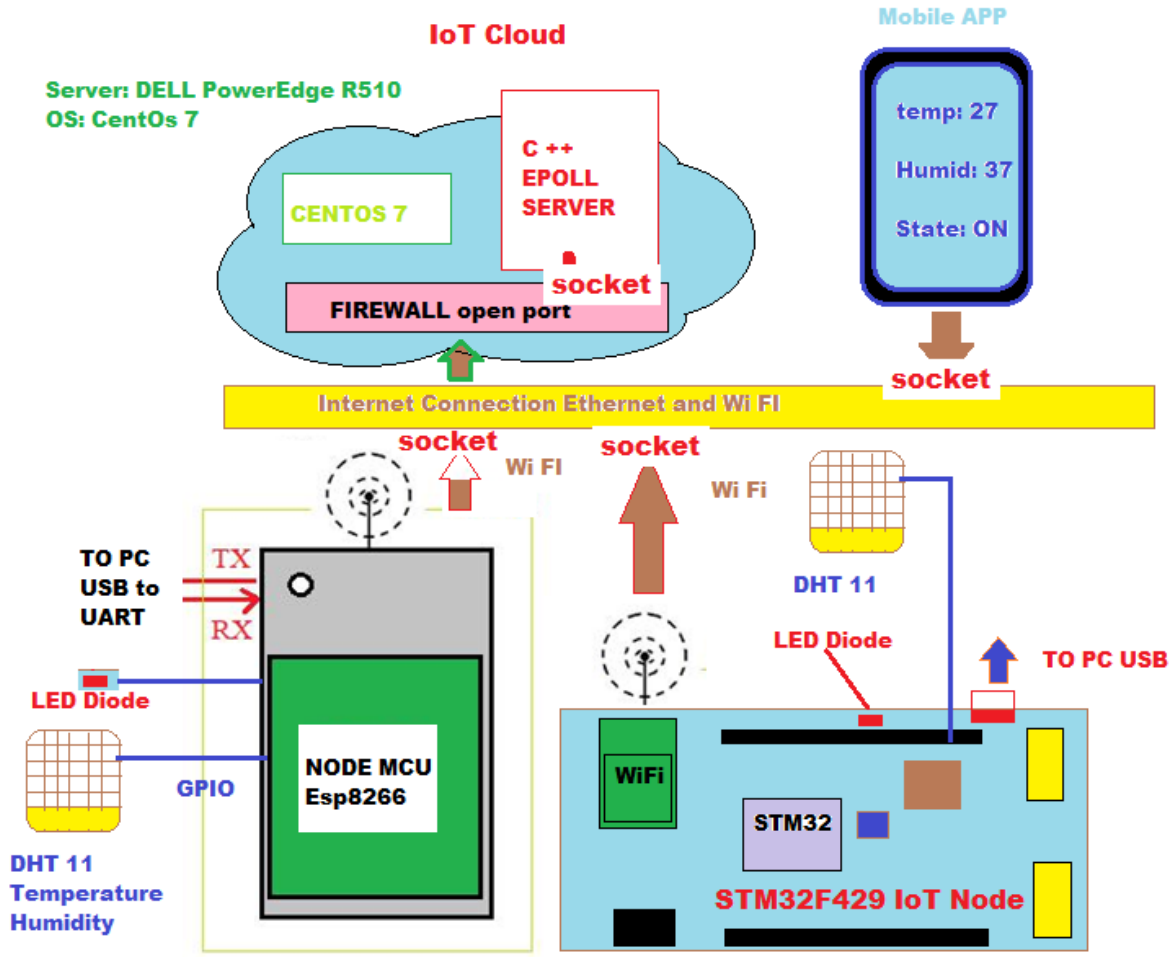

Figure 3. Architecture of Connected IoT Embedded systems to IoT Cloud

4. SOFTWARE REALIZATION

The C ++ language was used to implement Epoll Server, using the cmake tool to compile the program code. The Epoll Server application is wrote in Linux and used POSIX [8] [9]. The cmake tool is open source, multi-layered and designed to build, test, and package software. Used to manage the software compilation process using standard configuration files that are independent of the platform and compiler and generate so-called make files. First IoT devices must be connected to the Epoll server first. Once the IoT devices send a message it goes through IoT cloud processing and sends a message as a response to the IoT embedded system. If IoT embedded system is transmitting or reading information needs to connect, open a socket and send a message with the current temperature, humidity and status of the LED diode. The Epoll server must immediately respond with the corresponding result for the IoT embedded system, for example if the mobile application is connected to the IoT cloud and the LED diode is activated through it, then the server will pass the corresponding parameter to the IOT embedded system. This will activate the LED on the IoT embedded system. The Epoll server uses the so-called Ring Buffer to store the messages. The ring_buffer variable is used to store the messages that transcend the IoT devices and the mobile application, and DEFAULT_RING_BUFFER_SIZE is used to determine the queue size of the buffer, in this case 1024 :

```
auto ring_buffer =                                                    (1)
 new processor::RingBuffer<event_data>(DEFAULT_RING_BUFFER_SIZE);
```

The server consists of the main () function, device class (), process_messages () and event_loop (), as well as local variables. The Device () class defines constructor, destructor, local private variables, get () and set () methods to access the private variables (6):

```
private:                                                    (2)
int ddeviceID;  string ttype;
string nname;  float hhumidity;
string sstate;   float ttemp;
```

The private variables hhumidity, sstate, ttemp serve to store the temperature, humidity and condition sent by the IoT embedded system. The variables ddeviceID, ttype, nname serve to identify the type of device. The get () and set () methods are used to access private member variables :

```
int get_deviceID(){                                         (3)
return ddeviceID;
}
void set_deviceID(const int deviceID){
ddeviceID = deviceID;
}
```

The function process_messages is used to send messages from the server to the IoT embedded system. It defines the following variables required for server:

```
string bufferS;                                             (4)
Json::Value root,AndroidRoot;
Json::Reader reader;
string stateSensor = "off" ;
```

The root variable is used to compare the data to be sent to the IoT embedded system :

```
if(root["type"] == "IoTdevice"){                            (5)
string formatS;
sensor.set_deviceID(atoi(root["deviceID"].toStyledString().c_str()));
formatS = root["type"].toStyledString();
sensor.set_type(formatS.substr( 1 ,(formatS.length() -3 )));
```

Here it is seen that the content of the root variable "type" == "IoTdevice" is checked, and when its value is IoTdevice, then it is processed by taking the value from the field of the JSON file. In this case the deviceID or the unique device number connected to the server. You will find this number on the Epoll server console in the row:

```
printf("\n %d ",sensor.get_deviceID() );                    (6)
```

In the case where root ["type"] is sent from a mobile application, the following code rows are processed:

```
if(root["type"] == "IoTdevice"){                            (7)
string formatS;
sensor.set_deviceID(atoi(root["deviceID"].toStyledString().c_str()));
formatS = root["type"].toStyledString();
sensor.set_type(formatS.substr( 1 ,(formatS.length() -3 )));
```

The process_messages function runs continuously in the while() cycle, sending socket data to the device, whether it is a IoT embedded system or a Android mobile application. The void event_loop function (int epfd, int sfd, processor :: RingBuffer <event_data> * ring_buffer) is used to receive messages sent from embedded systems.

4.2. IoT IoT Embedded System Software Realization

The IoT embedded systems used for the experiment are Node Mcu esp8266 and STM32 IoT Node. Programming languages used for programming is low-level C and C ++. IoT embedded systems connect to the Epoll server via TCP sockets. It exchanges data with data being packaged in JSON format [12]. The format of the protocol is as follows :

{ "deviceID" : 1, (8)
"type" : "IoTdevice",
"humidity" : 42,
"name" : "ESPDH22",
"state" : "on",
"temp" : 28 }

The "deviceID", "type", "humidity", "name", "state" and "temp" fields are used to form the data transfer protocol between the IoT embedded system and the IoT Cloud structure. As you see, values are assigned to them. An example deviceID is used for a unique device number, each having its own number, which in this case is a unit. The type field shows what the device type is, in this case IoT device, but it can also be mobile - for a mobile application. The name field is the name of the embedded system, in this case ESPDH22 is the esp8266 and the DH22 is the type of the sensor. The state field can be on or off, and when the IoT is on, the device must turn on the LED diode. The humidity and temp fields show the humidity and temperature readings from the embedded system. Table 1 shows the fields in summary form.

Table 1. JSON fields of the IoT Embedded System Node MCU Esp8266.

Field name	Example value	Field type	meaning
"deviceID"	2	Integer	Unique number
"type"	"IoTdevice"	string	Type IoT device
"humidity"	15	integer	Humidity
"name"	" STM32 IoT node "	string	Name IoT device
"state"	"on"	string	Condition
"temp"	22	integer	Temperature

The program implementation of the Node Mcu esp8266 embedded system is implemented through the Arduino IDE development environment. A clientTCP [12] [13] object of the WiFiClient type that is used to open a socket to Cloud is created:

WiFiClient clientTCP; (9)
if (!clientTCP.connect("192.168.0.192", 3000)) {
Serial.println("Connection to echo server failed");
while (true) {}
}
clientTCP.print(jsonS)

with the code being used to use port 3000, which are passed as a parameter to clientTCP.connect ("192.168.0.192", 3000). The clientTCP.print (jsonS) function sends the JSON protocol contained in the jsonS variable. Receiving a Cloud Protocol to the IoT Embedded System is performed by the following code :

while (true) { (10)
while (clientTCP.available() == 0) {}
temp[i] = clientTCP.read();

where the clientTCP.available () function monitors esp8266's WiFi radio modular arrivals package, and after the packet arrives, the cycle condition changes, the clientTCP.read () function reads the data. The serialization and desealizing of the JSON protocol is done through the root object JsonObject, performing the so-called parse:

jsonT = temp; (11)
JsonObject& rootT = jsonBufferT.parseObject(jsonT);
const char* sensor = rootT["sensor"];

const char* tempC = rootT["temp"];

const char* state = rootT["state"];

by filling in the variable sensors, tempC, State of the fields of the JSON protocol.

The program realization of IoT embedded system STM32 IoT Node is realized with development environment IAR Embedded Workbench IDE. Used programming language is C. For sending data to the cloud are used following line:

WIFI_SendData(Socket, responce, sizeof(responce), &Datalen, WIFI_WRITE_TIMEOUT); (12)

with the variable response being used for the serialized JSON protocol that packs parameters for the temperature, humidity and condition of the LED diode. Using Cloud Data:

WIFI_ReceiveData(Socket, RxData, sizeof(RxData), &Datalen, WIFI_READ_TIMEOUT); (13)

4.3. Android Mobile device Software Realization

The Android mobile application serves to control the IoT embedded system Node Mcu esp8266. It connects to the Epoll server on port 3000 via TCP sockets [10] [11]. The mobile application controls the LED and displays parameters such as temperature, humidity and LED status. The graphical part of the application is shown in Fig. 9. Controls in the UI interface are used as text fields and buttons. The text boxes are temp, humidity and state, and connect, disconnect, on LED and off LED buttons. The technology used to implement the mobile app is Android Studio 3.0, and the programming language is Java. The form of the JSON communication protocol for the mobile application to the Cloud is the following lines:

{ "deviceID" : "3", (14)

"name" : "Android",

"state" : "on",

"type" : "mobile" }

as deviceID has the unique IoT device number, name is the name of the device, and the type is the type of device. The state field is used to turn on and off the LED on the built-in system. Table 2 shows the fields in summary form.

Table 2. JSON fields of Android mobile application for communication to the IoT Cloud.

Field name	Example value	Field type	meaning
"deviceID"	2	Integer	Unique number
"type"	"IoTdevice"	string	Type IoT device
"name"	" STM32 IoT node "	string	Name IoT device
"state"	"on"	string	Condition

The form of the IoT Cloud protocol to the mobile application is the same as sent by the IoT device, Table 1. The architecture of the mobile application is as follows: it has the Client.java class and the MainActivity.java class. The class describing the client has got () and set () methods to access the result of the server, constructor, doInBackground () method and variables. The class variables are private :

private String dstAddress; (15)

private JSONObject sendData;

private int dstPort;

private static String response = "{\n" + "\t\"humidity\" : 0,\n" + "\t\"sensor\" : \"0 \",\n" + "\t\"state\" : \"off\",\n" +"\t\"temp\" : 0\n" + "}\n";

The sendData variable is used to send data to the JSON-type server, dstAddress is the name of the address to which the mobile application connects, and the response is of the String type, which is the type of return returned by the server. The doInBackground method

starts a thread that takes care of the implementation of the code in the body of the method. Methods and buffers are found in the body of the method to open the connection to the server:

```
socket = new Socket(dstAddress, dstPort);                                    (16)
ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(1024);
byte[] buffer = new byte[1024];
```

the Socket method opens a socket to the server by a given address and port, ByteArrayOutputStream is a 1024-byte server input buffer:

```
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);           (17)
out.println(sendData.toString());
out.flush();
```

here PrintWriter, out and println (sendData.toString ()) are used to send data to the server:

```
response = "";                                                               (18)
while ((bytesRead = inputStream.read(buffer)) != -1) {
byteArrayOutputStream.write(buffer, 0, bytesRead);
response += byteArrayOutputStream.toString("UTF-8");
```

here response is the variable that takes the result of byteArrayOutputStream, converted to a String type. The parameter of the while () is the inputStream.read (buffer) method! = -1, which reads the received bytes from the server.

The MainActivity.java class is the private void updateUI () method, whose main job is to pack the data into a JSON object, visualize the UI interface by updating the text boxes - humidity, temperature and LED status. The serialization of JSON data takes place in the following way :

```
object2.put("deviceID","2");                                                 (19)
object2.put("type","mobile");
object2.put("name","Android");
object2.put("state","on");
```

in which case object2 is a JSON type variable, and the "deviceID", "type" "name" "state" fields are used to describe the communication protocol between the Epoll server and the mobile application. The text, temperature, humidity, and state text fields The LED diodes are updated via a Handler that allows MessageQueue messages to be sent and processed on the thread processing the MainActivity graphical environment, since it is not possible to directly access the text fields and update them from another thread :

```
private Handler handlerr = new Handler() {                                   (20)
@Override
public void handleMessage(Message msg) {
Bundle b = msg.getData();
String string = b.getString("Answer");
try {
JSONObject root = new JSONObject(string);
temper.setText(root.getString("temp"));
hum.setText(root.getString("humidity"));
onof.setText(root.getString("state"));
```

with the variable msg in the handleMessage method from which the message is taken using the variable b and is converged from the String type. To receive a message handler you need to add the following code rows in the updateUI () method :

```
b.putString("Answer", myClient.GetResponce());                               (21)
msg.setData(b);
handlerr.sendMessage(msg);
```

A thread is run to process the updateUI () method using the Runnable thread class :

```
private Runnable separateThread = new Runnable() {                    (22)
@Override
public void run() {
try {
updateUI();
```

5. EXPERIMENTAL RESULTS

Embedded systems are connected to the Cloud via a WiFi network. The mobile application connects to the mobile network's 3G network and is connected to the Internet. The IoT Cloud (Server) is connected to the Internet with a real and static IP address.

5.1. Test the work of IoT Cloud

For experiment are used embedded systems -NodeMCU esp8266 and STM32 IoT. They are connected to the IoT Cloud – Epoll Server, which connect to the server and are managed by an Android mobile app. In Fig. 4 shows the messages that are answered by the devices connected to the Cloud. The temperature and humidity values, as well as the state of the LED diode - whether or not on and the contents of the JSON package, see Figure 5.

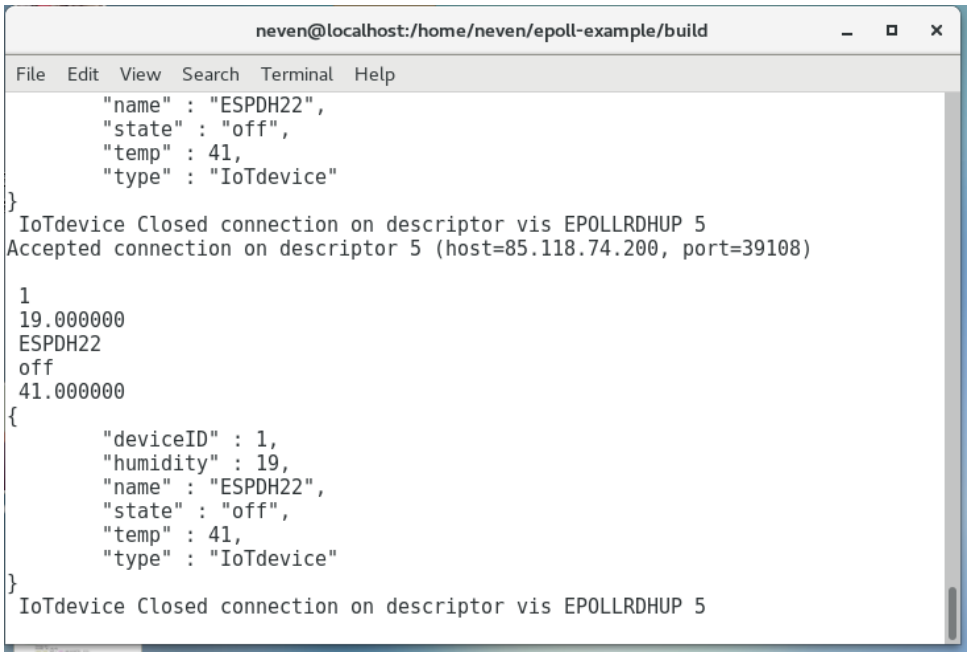

Figure 4. Messages as a response to the Epoll server console

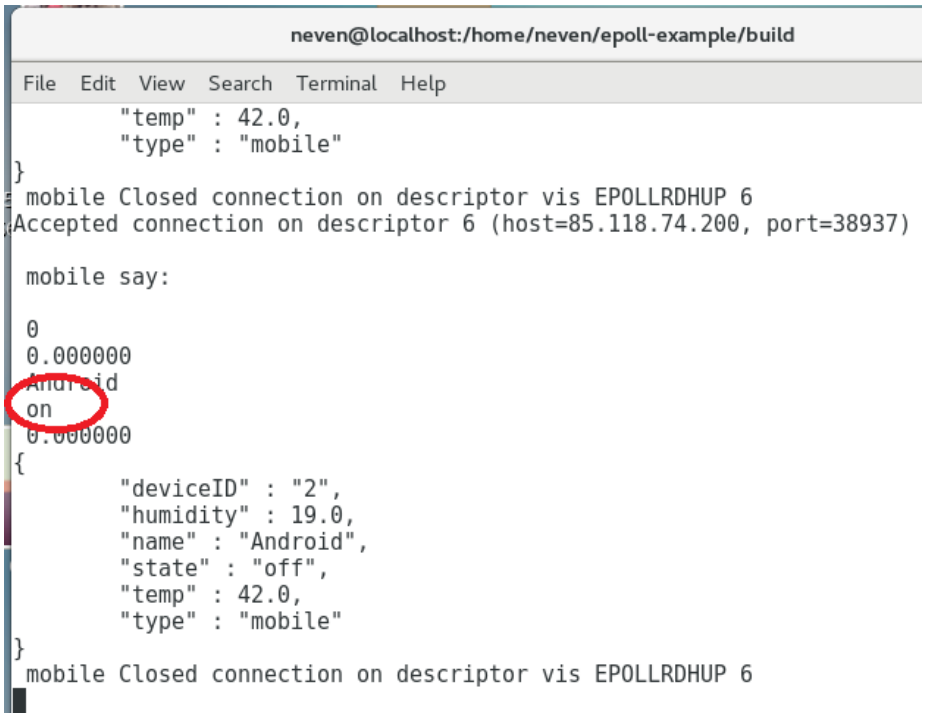

Figure 5. LED response message response

5.2. Test the work of IoT Embedded Systems

The embedded systems NodeMcu Esp8266 and STM32F475 IoT node are used for the experiment. In Fig. 6 shows the connection of NodeMcu Esp8266 to the Internet via WiFi and sending data to the Cloud. The Esp8266 embedded system is connected to USB to computer for debugging. In Fig. 8 shows the connection of the STM32F475 IoT node to the WiFi network. The WiFi Hotspot is an Android mobile phone that acts as a router, with the name of the SSID being AandroidAP. In Fig.7. the NodeMcu Esp8266 received a “on” state for LED diode from IoT Cloud. The IoT device has received a Command from the Cloud to trigger the LED, and the console displays the hold and a command called "on".

Figure 6. Successfully connect NodeMcu Esp8266 to an Internet network and send data to the Cloud

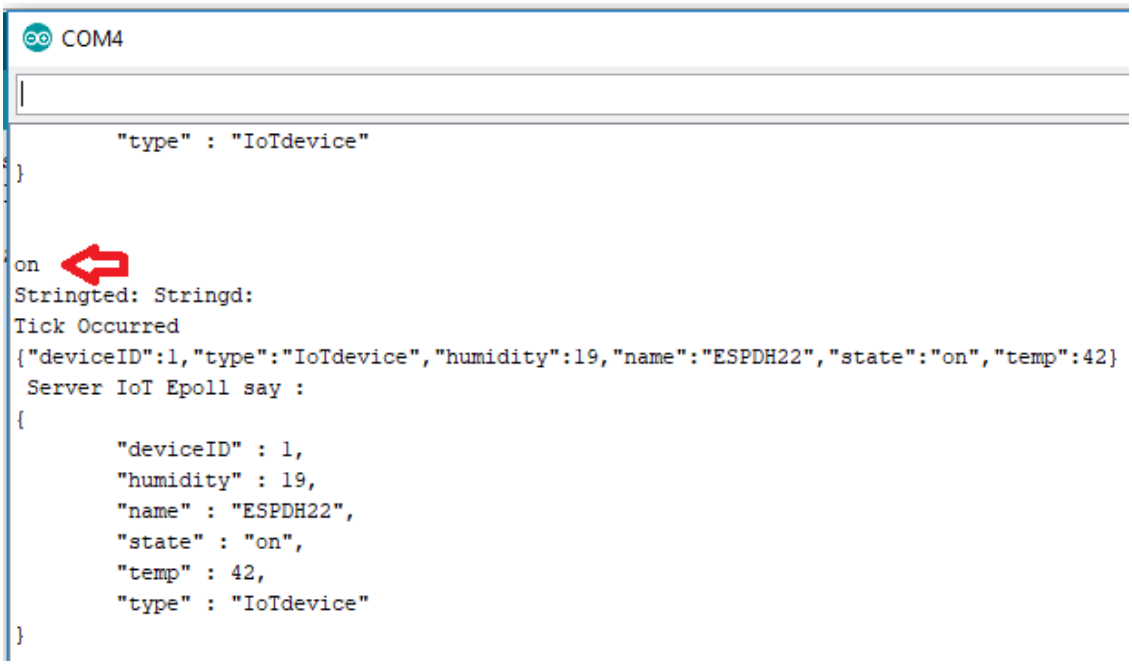

Figure 7. Turn on the LED of the NodeMcu embedded system

Upon feeding the power supply to the STM32F475 IoT node, the embedded system initializes the board, WiFi radio module, and peripherals such as temperature, humidity and GPIO input ports. In Fig. 8 introduces its initialization as well as the successful connection to the IoT Cloud on port 3000 to the Epoll server. The form and data protocol is identical to the NodeMcu Esp8266 embedded system. In Fig. 9 shows the sending of a JSON packet to the Cloud containing parameters for temperature and humidity :

{"deviceID" : "2","humidity" : 15.0, (54)
"name" : "STM32 IoT node","state" : "off",
"temp" : 22.0,"type" : "IoTdevice"}

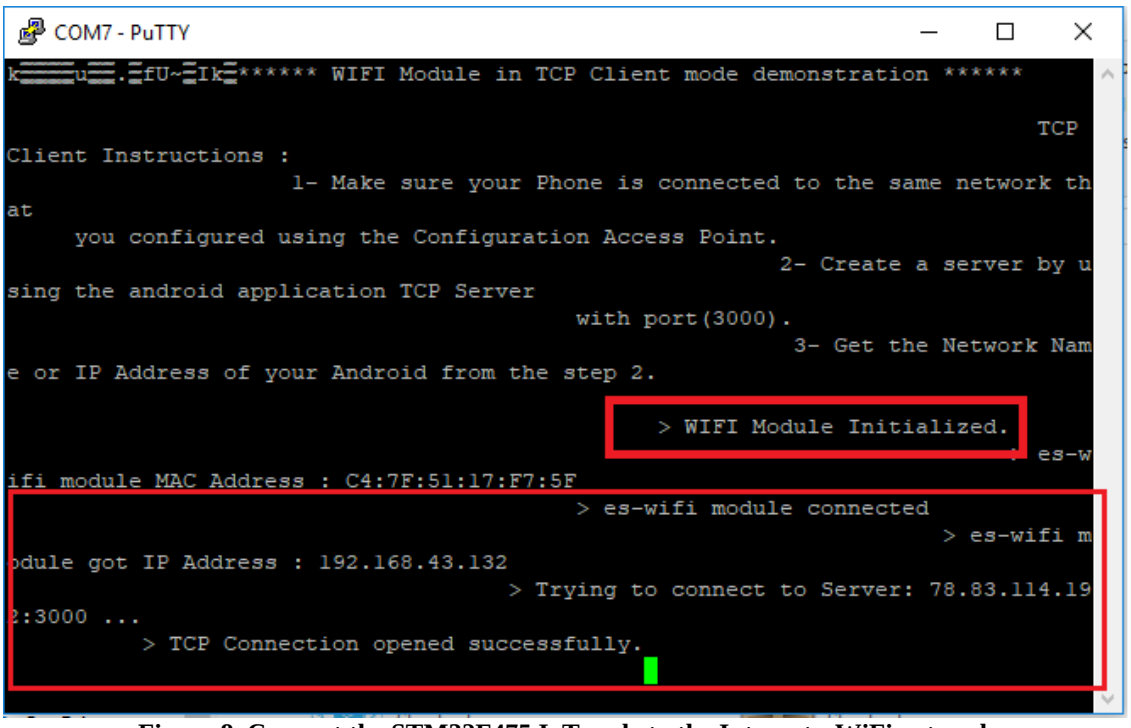

Figure 8. Connect the STM32F475 IoT node to the Internet - WiFi network

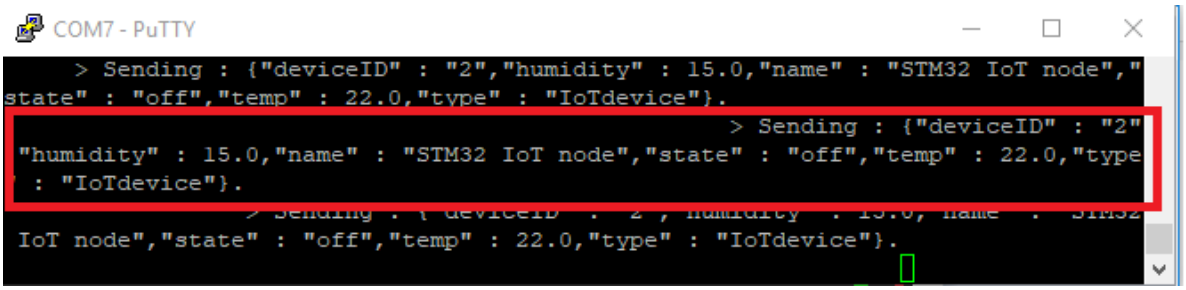

Figure 9. Send the STM32F475 IoT node of a JSON packet to the Cloud

5.3. Test the work of Android Mobile device

The mobile application is built on the Android operating system, which connects to the IoT cloud and reads data like the temperature, humidity and condition of the LED diode fig.10. The LG Nexus 5 phone with the Android operating system, 6 Marshmallow, was used. The "CONNECT" and "DISCONNECT" buttons are used for connection and disconnection by the Epoll server and the On Led" and "Off Led" buttons control whether the LED is on or off. In this case the readings are given in Table 3.

Table 3. Research the Android mobile application and showing the measurement values

IoT Device	Temperature	Humidity	State LED
Esp8266	27	37	off
STM32F475	26	37	on

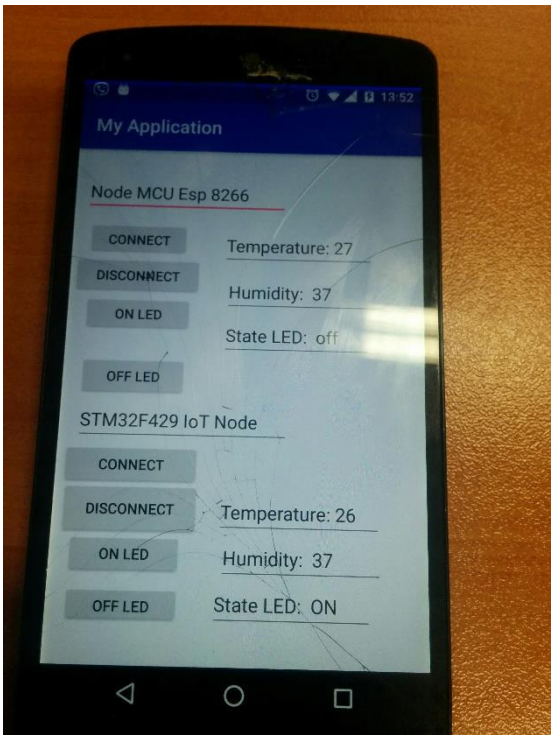

Figure 10. Temperature and humidity displayed on mobile application

6. CONCLUSION

This article uses an experiment that makes communication between Androit mobile application, IoT Cloud and IoT embedded systems. An experimental layout, system-wide architecture and fragments of software implementation are presented. The communication is accomplished by using sockets for communication via port and IP address, with all devices using sockets. The embedded systems are two STM32F429 IoT node and Node MCU

esp8266 with a different mapping. Communication between IoT embedded systems, IoT Cloud and mobile application written on Android.

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] R. Preethika , G. Prabhu, A. Prabhu, AN ANALYSIS OF COMMUNICATION WITH IOT DEVICES USING WEBSOCKETS, 2017J. Weinman, "The Strategic Value of the Cloud", IEEE Cloud Computing, vol.2, pp 66-70, 2015
[2] Frédéric Camps, Connected objects, IoT, M2M, architecture, protocols, 2017
[3] Postscapes, IoT Standards and Protocols, 2018
[4] Botta, W. de Donato, V. Persico, A. Pescap´e, Integration of Cloud Computing and Internet of Things: a Survey, September 18, 2015
[5] Cloud Standards Customer Concil, Cloud Customer Architecture for IoT, 2016
[6] Yu Liu1 , B. Dong , B. Guo , J. Yang and W. Peng, Combination of Cloud Computing and Internet of Things (IOT) in Medical Monitoring Systems, 2015
[7] STMicroelectronics, Wireless connectivity for IoT applications, 2015
[8] Robert Love, Linux System Programming, 2007 O'Reilly Media
[9] Eclipse Paho, MQTT C++ Client for Posix and Windows, 2018
[10] developer.android.com, Developer Guides Android, 2018
[11] STMicroelectronics, Discovery kit for IoT node, multi-channel communication with STM32L4, 2018
[12] techtutorialsx.com, ESP32 Arduino Websocket server: Receiving and parsing JSON content, 2017
[13] hackster.io, Temperature Dashboard Using Arduino UNO, ESP8266 And MQTT, 2016
[14] Botta, W. de Donato, V. Persico, A. Pescap´e, "Integration of Cloud Computing and Internet of Things: a Survey", September 18, 2015

ПРОЕКТИРАНЕ И РЕАЛИЗИРАНЕ НА КОМУНИКАЦИИ МЕЖДУ IOT ОБЛАЧНА СТРУКТУРА, АНДРОИДНО ПРИЛОЖЕНИЕ И IOT УСТРОЙСТВО, ИЗПОЛЗВАЩИ TCP СОКЕТИ

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

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

Ключови думи: *Firmware, Esp8266, Dht11, IoT устройство, STM32L4 откритие IoT, JSON, TCP гнезда, Андроид, Epoll сървър, IoT облак*
Резюме: *Тази статия представя комуникацията между облака IoT, мобилното приложение Android и вградените системи на IoT. Представяме начин за предаване на температура и влажност на IoT облака чрез TCP сокети. Формат на данни тип JSON се използва между структурата на облака и устройствата. Чрез вградените системи STM32L4 discovery и Esp8266 са прехвърлени данни към облака IoT. Мобилното приложение реализира представянето на показанията за температура и влажност от датчик. Представена е архитектура и софтуерна реализация на експерименталния етап.*