ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ
Факултет по Компютърни системи и управление
Катедра „Компютърни системи”

маг. инж. Невен Красимиров Николов

МЕТОДИ ЗА ОПТИМИЗИРАНЕ НА КОМУНИКАЦИЯТА НА ВГРАДЕНИ СИСТЕМИ ЧРЕЗ ИЗПОЛЗВАНЕ НА ОБЛАЧНИ СТРУКТУРИ

А В Т О Р Е Ф Е Р А Т

на дисертация за придобиване на образователна и научна степен
"ДОКТОР"

Област: 5. Технически науки

Професионално направление: 5.3 Комуникационна и компютърна техника
Научна специалност: „Компютърни системи, комплекси и мрежи”

Научен ръководител: проф. д-р инж. Огнян Наков

СОФИЯ, 2018 г.

Дисертационният труд е обсъден и насочен за защита от Катедрения съвет на катедра „Компютърни системи“ към Факултет „Компютърни системи и технологии“ на ТУ-София на редовно заседание, проведено на 19.06.2018г..

Публичната защита на дисертационния труд ще се състои на 08.10.2018г. от 15:00 часа в Конферентната зала на БИЦ на Технически Университет – София на открито заседание на научното жури, определено със заповед № ..... / ..... г. на Ректора на ТУ-София в състав:

1. Проф. д-р Огнян Наков – председател
2. Проф. д-р. Владимир Лазаров
3. Доц. д-р. Милен Петров
4. Доц. д-р. Славчо Манолов
5. Доц. д-р. Надежда Кучмова

Резервни:
1. Проф. д-р Даниела Гоцева
2. Проф. д-р Веселин Павлов

Рецензенти:

1. Проф. д-р Владимир Лазаров
2. Доц. д-р Надежда Кучмова

Материалите по защитата са на разположение на интересуващите се в канцеларията на Факултет „Компютърни системи и технологии“ на ТУ-София, блок №1, кабинет № 1431.

Дисертантът е задочен докторант към катедра „Компютърни системи“ на факултет „Компютърни системи и технологии“. Изследванията по дисертационната разработка са направени от автора.

Автор: маг. инж. Невен Красимиров Николов
Заглавие: „Методи за оптимизиране на комуникацията на вградени системи чрез използване на облачни структури“
Тираж: 30 броя
Отпечатано в ИПК на Технически университет – София

I. ОБЩА ХАРАКТЕРИСТИКА НА ДИСЕРТАЦИОННИЯ ТРУД

Актуалност на проблема

След анализ на областта се вижда, че Интернет на Нещата – IoT е нова област, в която борят на свързаните устройства нараства непрекъснато, в следствие на което трябва да се предложат нови модели за комуникация и синхронизация, както и надграждане на съществуващите. Новите модели предлагат нов и подобрен подход за комуникация между вградени системи и облачни структури, както и обновяване на управляващият софтуер – firmware.

Цел на дисертационния труд, основни задачи и методи за изследване

Дисертационният труд е посветен на задълбочено изследване на IoT протоколи за комуникация и пренос на данни към Сървър (Облачна структура), предложен е вариант за комуникация между IoT вградени системи и Облачни структури със TCP socket комуникация и пренасяне на данни в JSON формат, както и обновяване на управляващият софтуер на IoT устройства.

Във връзка с основната цел са формулирани следните задачи:

1. Да се изследват основните протоколи MQTT и TCP sockets използвани за комуникация и синхронизация на вградени системи към облачни структури.
2. Да се реализира Облачна структура – сървър, който да приема и предава данни от вградени системи - IoT устройства. Данните трябва да бъдат визуализирани на потребителя.
3. Да се реализира и изследва мобилно приложение, което управлява IoT вградена система
4. Да се реализира и изследва обновяването на софтуера на вградена система - FOTA от Облачна структура.
5 Да се предложи, реализира и изследва протокол за комуникация между IoT вградени системи и Облачни структури със TCP socket протокол за пренос на данни в JSON формат

Научна новост

Научната новост се състои в използване, комбиниране и предлагане на нов метод за комуникация между вградени системи и облачни структури.

Практическа приложимост

Практическата приложимост е голяма, поради факта че броят на интернет свързаните устройства нараства непрекъснато. Обема на информацията непрекъснато се увеличава и са нужни нови решения за свързаност между вградени системи и облачни структури.

Апробация

В процеса на изследване е реализирана опитна постановка, в която са наравени редица изследвания на протоколите за комуникация и пренос на данни, както и обновяването на управляващият софтуер. Изследвана е работата на облачната структура.

Публикации

Направени са шест научни публикации, от които четири са на международни конференции, една е в списание, три са на английски език.

Структура и обем на дисертационния труд

Дисертационният труд е в обем от *(145)* страници, като включва увод, *(4)* глави за решаване на формулираните основни задачи, списък на основните приноси, списък на публикациите по дисертацията и използвана литература. Цитирани са общо *(110)* литературни източници, като *(110)* са на латиница и *(0)* на кирилица, а останалите са интернет адреси. Работата включва общо *(103)* фигури и *(20)* таблици. Номерата на фигурите и таблиците в авторефератa съответстват на тези в дисертационния труд.

II. СЪДЪРЖАНИЕ НА ДИСЕРТАЦИОННИЯ ТРУД

ГЛАВА 1. Същност на вградените системи. Интернет на нещата IoT. Анализ на съществуващи решения и технологий

1.0 Същност на вградените системи

Internet of Things – IoT [27] [70] [97]е концепция за връзката на всякакви устройства към интернет и към други свързани устройства. IoT представлява огромна мрежа от свързани устройства – те всички използват и споделят данни [37] [68] [98]. Утройствата свързани в интернет могат да бъдат всякакви – умна кухня, автономни автомобили, фитнес устройства който измерват сърдечния пулс, или номера на стъпките изминати за деня. Можем да добавим към IoT термини като IoT – Industrial Internet of Things, Internet of Service, Internet of Everything и разбирасе Industry 4.0. Към IoT можем да прибавим изчисленията в облака – Cloud computing, анализ на големи данни – Big Data Analytics, като можем да добавим, че IoT има свой разнообразен технологичен стек .

1.1 Анализ на съществуващи решения и технологий при комуникация между вградени системи

Комуникационните възможности при вградените системи се характеризират с голямо разнообразие. Те се определят от вида на вградената система и нейната специфичност, приложение и идея. Съществуват различни интерфейси и среди за комуникация между вградени системи. Избор на комуникационна среда се прави в зависимост от средата в която ще се намира вградената система. Примерно в индустриална среда, трябва да се използва по специфична преносна среда, примерно industrial ehternet, profinet, powerlink, Modbus/TCP, RS485 и други комуникационни среди. Тук основната идея е да се осигури контрол в реално време, както и да може мрежата да бъде предпазена от всякакви видове радиосмущения и интерференций. Примерно за домашна среда не е необходимо да се използват преносни среди както в индустриална зона. За домашна мрежа може да се използва Wi Fi безжична връзка за комуникация. Съществува LoraWan стандарт разработен за интернет на нещата IoT, който може да се дизползва както за индустриална, така и за домашна среда. Съществуват и други комуникационни среди като Bluetooth, ZegBee и т.н. Но до тук са изброени външни комуникационни среди за пренос на данни, вътрешните са във вградената система. Вградената система може да бъде изградена от един или няколко микроконтролера (микропроцесора), който комуникират помежду си, комуникират с радио модули или други интегрални схеми, като серийни флаш памети, дисплей и др.

1.3 Комуникация между вградени системи. Физически среди

WiFi връзката често е очевиден избор за много разработчици, особено като се има предвид широкото разпространение на WiFi в домашната среда в локалните мрежи [101] . WiFi има по-широка съществуваща инфраструктура, както и предлагане на бърз трансфер на данни и възможност за обработка на големи количества данни. По настоящем най-често срещаният WiFi стандарт, използван в домовете и много фирми, е 802.11n, който предлага сериозна производителност в диапазона от стотици мегабита в секунда, което е предостатъчен избор за пренос на данни, но може да е прекалено енергоемко за много приложения на IoT вградени системи.

1.4 Използвани протоколи за комуникация на IoT вградени системи

Съществуват различни видове IoT протоколи за комуникация и пренос на данни между вградени системи и облачни структури и сървъри [3] [41] [100] [102] [104] [106].

1.4.1 Протокол MQTT

MQTT е разработен от Анди Станфорд-Кларк (IBM) и Arlen Nipper (Eurotech, сега Cirrus Link) през 1999 г. за мониторинг на петролопровод през пустинята. Целите са да има протокол, който да е ефективен по отношение на честотната лента и да използва малко енергия от батериите, тъй като устройствата са свързани чрез сателитна връзка , а това е било изключително скъпо за това време. MQTT е проектиран за мрежи с ниска честотна лента, висока латентност в края на 90-те и началото на 2000-те години и нейната поддръжка е за свързване на хиляди клиенти към един сървър. Тази характеристика е важна за устройства, които имат много ограничени възможности за обработка и ограничена памет, като например сензори, мобилни устройства, устройства за наблюдение. Тъй като осигуряват общ интерфейс за всичко, новите сензори или устройства могат да се интегрират много лесно към системата.

Протоколът използва архитектура publish/subscribe [38], различна от HTTP с парадигмата за request/response. Publish/subscribe се управлява от събития и позволява съобщенията да бъдат придвижвани към клиентите. Централната комуникационна точка е брокерът на MQTT, който отговаря за изпращането на всички съобщения между изпращачите и правилните приемници. Всеки клиент, който публикува съобщение до брокера, включва тема в съобщението. Темата е информацията за маршрута на брокера. Всеки клиент, който иска да получава съобщения, се абонира за определена тема и брокерът доставя на клиента всички съобщения със съответната тема. Поради това клиентите не трябва да се познават, те само комуникират по темата. Тази архитектура позволява много мащабируеми решения без зависимости между производителите на данни и потребителите на данни Фиг.1.2

1.4.6 TCP socket

TCP socket са low-latency (ниско закъснение), real-time (в реално време), full duplex (двупосочна), long-running (постоянна), единична връзка (TCP) между клиент и сървър [39]. TCP sockets осигуряват огромна полза за уеб приложенията, задвижвани от събития в реално време и са разпространени за актуализиране на данни в реално време и синхронизация, чат на живо, видеоконференции, VOIP, контрол на IoT вградени системи и мониторинг. Те се използват за различни приложения като игри, социални мрежи , мрежи за сътрудничество, логистика, финанси, IoT устройства, автомобилната и индустриалната автоматизация фиг.1.7 . Повечето големи уеб браузъри в момента поддържат използването им.

1.5 Комуникация между вградени системи и облачни структури

IoT Cloud е платформа, която служи за обработка и съхраняване на информация от вградени системи [35] [105]. Това е така наречената скалируема система, която може да предостави малка и голяма изчислителна мощ, в зависимост от нуждата. Примерно при малък брой устройства не е нужно да се използва по –голяма изчислителна мощност предвидена, понеже цената която ще се плати е голяма и неоправдана. За това се използва толкова изчислителна мощ, колкото е нужно. Когато броя на устройствата се увеличи , ще трябва да нарастне изчислителната мощ на облака, за да може да поеме по-голям товар. Тогава говорим за скалируемост, тоест при по-голям брой IoT устройства имаме респективно по-голямо натоварване на облака. Облачните структури са с големи капацитети и могат да ни предоставят по-голяма изчислителна мощност. Тук говорим за IaaS (Infrastructure of a service) или отдаване на сървърна мощност под наем. IoT cloud притежава и SaaS (Software as a Service) или софтуер като услуга, където той предоставя софтуер или готова система , към която трябва да се интегрира кода на приложението. IBM разделя изчисленията в облака на шест различни категорий:

1.6 Обновяване на управляващия софтуер

Обновяването на управляващия софтуер на вградените системи е голямо предимство за потребители, производители и консуматори на IoT устройства. Доста често в практиката се случва да се открият бъгове в софтуера или пък да се добави повече функционалност или пък да се промени алгоритъма на работа на дадено устройство. При такива ситуации е особено наложително опцията за обновяване да бъде налична. Обновяването на микроконтролери и флаш памети на вградени системи може да се осъшестви по няколко начина. Примерно чрез шини като UART, USB, I2c, SPI, примерно чрез поставяне на MicroSD карта свързана към SPI шината или през USB флаш памет. При такъв начин на обновяване на устойството е необходимо мониторна програма-bootloader да поддържа драйвери за устройството или пътя от който ще се осъществява обновяването на устройството. Bootloader е firmware на микроконтролера на IoT устройството, който не се обновява поради възможността то да бъде повредено или т.н. термин – брикнато (brick). Неговата работа е да позволи да се обнови правилно управляващият софтуер на вградената система. Но при вградените системи свързани в интернет – IoT устойствата имат достъп до мрежата и поради този факт те могат да бъдат обновявани отдалечено. Това подпомага в ситуации, където устройствата са далеч, те се намират на трудно достъпни места, на места където има радиация и човек не може да ги достъпи. Обновяването през интернет е така наречения термин FOTA – Firmware Update Over the Air или обновяване през въздуха, когато става дума за безжична връзка или понятие като RFU – Remote Firmware Update касаещо всякакъв вид отдалечено обновяване. Обновяването може да се осъществи през всякакъв физически интерфейс, независимо от физическата среда, била тя Ehternet, WiFi, LoraWan и т.н. Могат да бъдат използвани всякакъв вид протоколи за пренос на Firmware на вградената система. На фиг. 2.3 е показана основна архитектура на вградена система, която обновява своят фирмуер – firmware [4].

Както е показано на фиг.2.5 микроконтролера flash паметта (non-volatile memory) е дефинирана в два сектора – bootloader и bootloadable (приложението или потребителската програма). Bootloader е частта от паметта, която съхранява кода, който е отговорен за операцията за зареждане на устройството, проверява дали хостът има актуализирано изображение на firmware за секцията, която може да се зарежда с bootloader, получава актуализираното изображение на firmware от хоста чрез UART интерфейс и записва актуализираното изображение в зареждащата се част от паметта. Bootloadable секцията от паметта е действителният код на приложението, който дефинира функционалността на системата. За да се осъществи OTA обновяване на firmware , паметта трябва да бъде разделена на bootloader и Bootloadable секций. Тук чрез комуникационният интерфейс се извлича новото firmware изображение от хоста. В случай на актуализации на firmware - FOTA по безжичен комуникационен интерфейс се използват интерфейси като WiFi, ZigBee, Bluetooth,

LoraWan и други за получаване на актуално изображение на firmware по въздуха. Архитектурата на OTA bootloader може да бъде различна в зависимост от приложението и изискванията относно крайните решения. Съществуват два типа на OTA bootloader архитектури – OTA bootloader с фиксиран стек и с обновяем стек.

Изводи от I глава

• Интернет на Нещата – IoT е нова област, която се развива бурно през последните години. Това довежда до създаването на нова ера на автоматизация и централизирано управление на различни области в живота – умни домове, умни градове, свързани автомобили, индустрия 4.0 за автоматизирано производство и други области. Глобалната мрежа – Интернет е достатъчно развита да бъде основен канал за пренос на данните от милиони даже милиярди устройства, който изпращат непрекъснато параметри от сензори или управляват фабрики и заводи в реално време.

• Поради непрекъснато нарастващият брой на свързани в интернет устройства трябва да бъде предложен по-добър вариант за пренос и синхронизация на данни между вградени системи и облачни структури.

• Съществуват различни физически среди като WiFi, loraWan, ZigBee и т,н. Съществуват различни протоколи , който са подходящи за определени ситуации. Важно е при проектирането на IoT система да се подберат подходящите като се вземат в предвид следните ограничителни фактори, като захранването, преноса на данни, ограничените ресурси на IoT вградените системи – CPU, RAM и Flash памет.

• Съществуват различни облачни структури, който предлагат скалируемост на броя на IoT устройствата – Microsoft Azure, Ibm Watson, Aws IoT cloud. Те си имат свойте предимства, но и недостатъци – примерно тяхната цена и защита на личните данни.

• Обновяването на управляващият софтуер на IoT вградени системи е удобно в ситуации в който устройствата са пуснати на пазара и не е нужно да се носят в сервизни центрове .

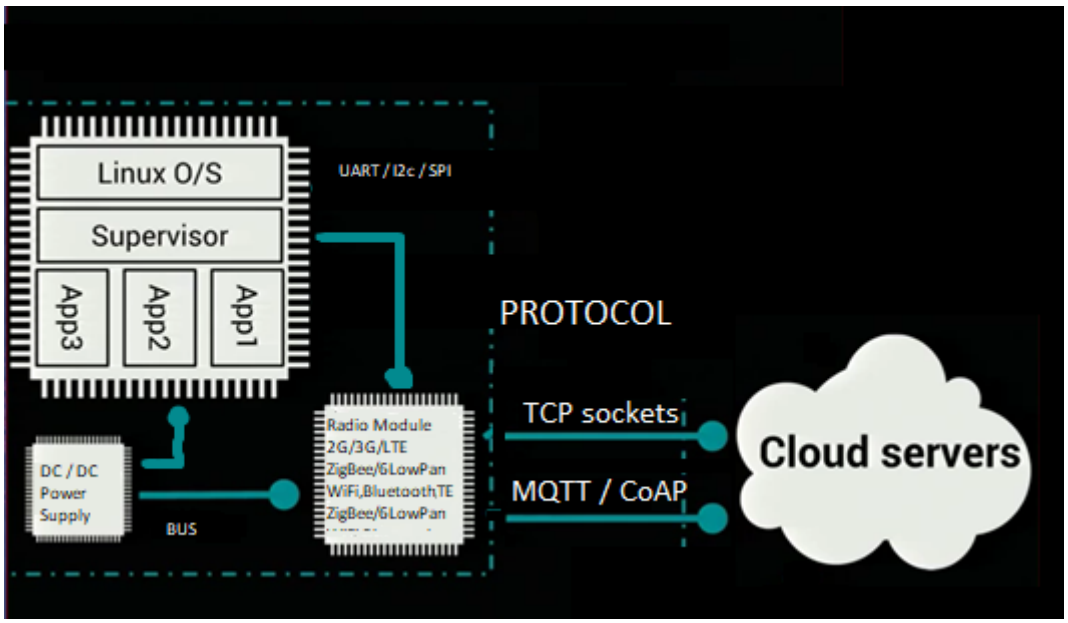
ГЛАВА 2. Алгоритми за оптимизация на комуникацията и обновяване на софтуера на вградени системи. Теоретична постановка

Глава II е посветена на изграждането на архитектурата и теоретичната постановка на изследването. Разглеждат се подробно протоколите за комуникация между вградени системи и облачни структури, предлага се нов модел за представяне и предаване на данни . Разглеждат се съществуващи протоколи за синхронизация и комуникация и се предлагат подобрения с цел оптимизация на комуникацията между вградени системи и облачни структури.

Комуникацията е от основно значение за интернет на нещата. Мрежовите технологии позволяват устройствата на IoT да комуникират с други устройства, както и с приложения и услуги, които се изпълняват в облака [71] [108]. Интернет се основава на стандартизирани протоколи, за да се гарантира, че комуникацията между хетерогенни устройства може да се осъществи надеждно. Стандартните протоколи определят правилата и форматите, които устройствата използват за създаване и управление на мрежи, както и за предаване на данни в тези мрежи.

2.2 Комуникация между вградени системи и облачни структури

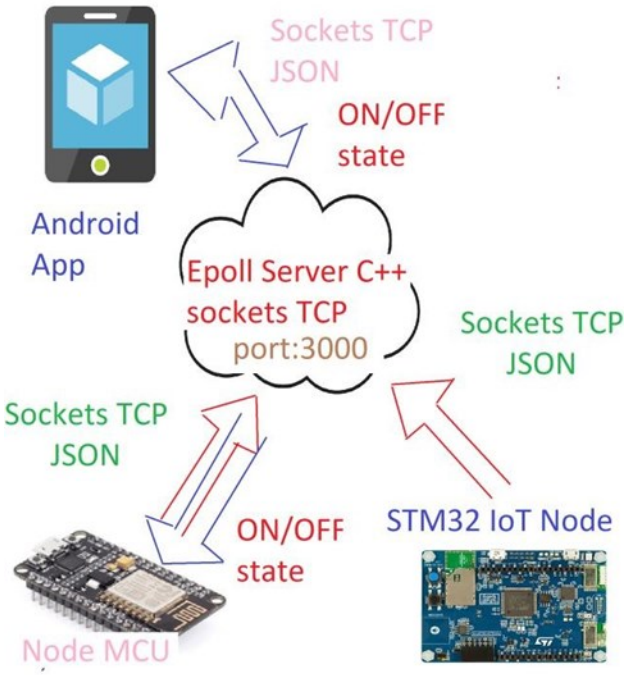
На фиг. 3.1 е представена обобщена схема на IoT система изградена от микроконтролер (процесор) , Cloud сървър [19] [42] [44] [62] [89], Радио модул за комуникация, захранващ модул на вградената система, както и Network Operator в случая за физическа преносна среда на данните. Микроконтролера, радио модула и захранващия блок са минималната конфигурация на IoT вградената система. Микроконтролера комуникира чрез интерфейси като UART, I2c, SPI с радио модула. Радио модула може да бъде всякакъв тип, в зависимост от физическата среда – 2G/3G/LTE, ZigBee, 6LowPan, WiFi, Bluetooth. Комуникацията с радио модула може да се осъществи чрез AT команди



Фиг.3.1. Обобщена схема на IoT вградена система свързана към Cloud server

2.3 Архитектура на IoT система за обмяна на данни

Архитектурата на системата е представена на фиг.3.2 и фиг.3.3. Тя включва Облака [12] [64] [75] [94] [95], вградените системи и мобилното приложение, което сканира температурата и влажността, както и включва и изключва светодиода. На фиг.3.3 е представена комуникацията чрез протокола MQTT, а на фиг.3.2 custom протокол чрез комуникация със сокети [72]. И в двата случая формата на протокола е един и същ, използва се JSON (Javascript Object Notation) човеко четим тип формат за пренос на данни M2M [109].



Фиг. 3.2. Архитектура Custom протокол със използване на TCP sockets

На фиг.3.2 е показана обобщена схема на свързаните устройства към облака. Показани са мобилното устройство, което е Android и вградените системи – NodeMcu и STM32 IoT Node.

Облака е реализиран, като е използван Epoll Server, който работи като процес в операционната система Centos 7. Използван е Epoll Server написан на езика C++, поради неговото бързодействие и време за реактция. Сървъра слуша на порт 3000, като вградените системи се свързват към него, а той отваря TCP socket комуникация. Всяка вградена система която изпраща данни се пращат към сървъра, а той им връща обратно резултат след обработка. Вградените системи трябва да слушат за върнатия резултат. Примерно вградената система NodeMcu Esp8266 при получаване на отговор за активиране на светодиода, трябва да задейства във ниво лог.1 gpio пин към който е свързан светодиода. STM32 IoT Node единствено изпраща данни за температурата и влажността.

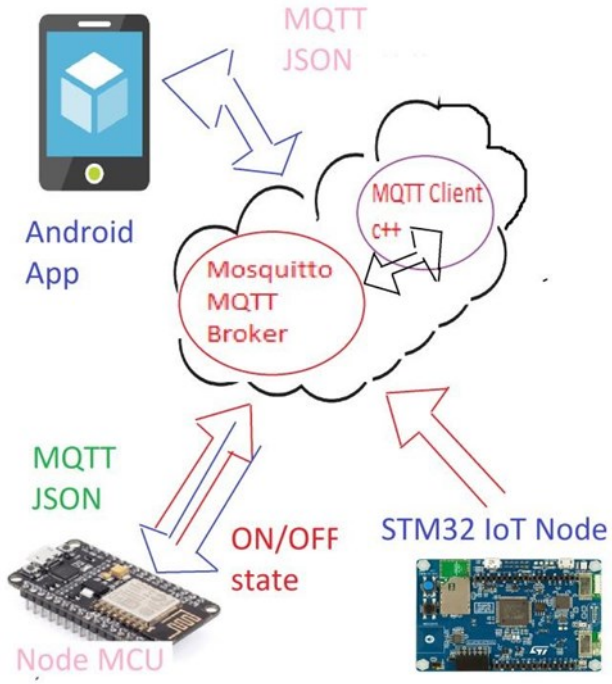
Android приложението след като се свърже към сървъра, получава съобщение изпратено от вградените системи. Приложението е написано на езика Java, като средата за разработка е Android Studio. След като се свърже със сървъра и се отвори TCP socket комуникация, android приложението получава протокола в тип JSON, десериализира го и визуализира температурата, влажността и състоянието на светодиода.

Формата на протокола изпратен от вградените системи към Epoll Server C++ (1) е следният:

{"deviceID":1,"type":"IoTdevice","humidity":42, (1)

"name":"ESPDH22","state":"on","temp":28}

като полето "deviceID" е уникален номер на всяко устройство, "humidity" е влажността прочетена от температурния датчик DHT22, "name" е името на вградената система, "state" е състоянието на светодиода – включен/изключен, "temp" е температурата отчетена от температурния датчик, а "type" показва типа на устройството. В случая типа е "IoTdevice".



Фиг. 3.3. Архитектура с използване на MQTT протокол

Формата на протокола изпратен от мобилното приложение към сървъра е следният (2) :

{ "deviceID" : "2", (2)
"name" : "Android",

"state" : "on",
"type" : "mobile" }

като полето "name" е Android а "type" е mobile. В случая е изпратено състояние "on" за включване на светодиода на вградената система.

На фиг.3.3 е показана обобщена схема на архитектурата , при която се използва MQTT протокола. При този вариант се използва брокер Mosquitto MQTT, като неговата работа е да свързва всички устройства, който работят на принципа Publish/Subscriber. Използва се порт 1883, на който отговаря брокера. В облака е реализиран MQTT клиент, който е абониран за всички устройства и получава данните от вградените системи и мобилното приложение. Мобилното приложение изпраща данните в тип JSON, като формата им е описаният по-горе. В случая за Subscriber е MQTT клиента в облака, като той получава данните от обилното приложение. Вградената система ESP8266 е абонирана за данните, който ще получи от MQTT клиента. Така се изпраща командата за включване на светодиода на вградената система. Данните за температурата и влажността от вградената система се публикуват към клиента във Облака, след което той ги публикува към мобилното приложение.

2.4 Архитектура за обновяване на управляващия софтуер на IoT вградени системи.

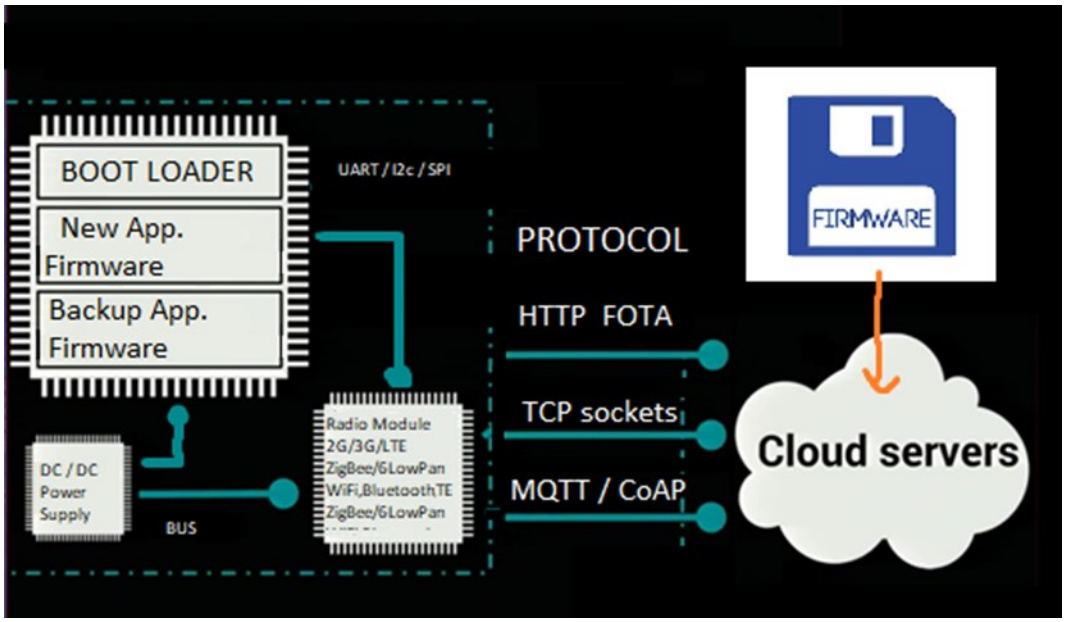
С нарастването на IoT в днешно време броят на свързаните устройства нараства експоненциално, както и свързаният софтуер, който ги управлява. Няма съмнение, че фърмуерът над въздуха (FOTA) е изключително желана - ако не се изисква - функция за всеки вграден проект / продукт както за домашно, така и за търговско. Да можеш да предоставиш дистанционна актуализация на фърмуера очевидно е много полезно. Възможността тук е да се подобри функционалността на продукта, оперативните функции и да се дадат корекции за конкретни проблеми. Актуализирането на фърмуера OTA може да премахне необходимостта от поставяне на продукт в сервизен център за ремонт. Въпреки че не всеки проблем може да бъде разрешен с актуализация на фърмуера, ако има такъв за конкретен проблем, той може да спести много време и пари.

2.4.1 Обща концепция за обновяване на програмната памет на вградени системи

За обновяване на управляващия софтуер (firmware) на вградената система, съществуват няколко варианта [90]. Обновяването на софтуера се обуславя чрез заместване на старият софтуер (firmware) разположен във flash паметта на микроконтролера или външната памет. Вградената система ESP8266 използва SPI flash памет от серия W25QXX, която може да бъде с размер 512kb, 4mb, 8mb и т.н. На енергонезависимата памет - flash се разполагат няколко независими блока от програмен код и данни. Независимо дали се използва операционна система примерно Lunux, RTOS или RiotOS се използва специална програма наречена „boot loader“ [92]. Следващите блокове заемащи място във flash паметта са два блока в програмната памет. Първият включва софтуера на вградената система, а вторият блок е с големина като първия. Вторият блок се използва за съхраняване на новия управляващ софтуер, който се получава от сървъра (Cloud). Съществува вариант, при който програмната памет не е достатъчна да побере два блока, от който единия да съхранява старият софтуер. Когато софтуера открие че сървъра е изпратил заявка за обновяване на firmware на вградената система, тя трябва да изтегли новият firmware или във RAM паметта, или да замени старият софтуер директно или да съхрани новият във втория регион. На фиг 4.6 е показан вариант при който се пази старият софтуер в регион „Standby Copy“, а новият се намира в „Running Copy“. Когато се копира от сървъра (Cloud) новият софтуер, той се съхранвява във вторият регион „Standby Copy“, след което регион „boot loader“ променя адреса на указателя от който трябва да се зареди новият софтуер.

2.4.2 Обща концепция на обновяване на програмната памет от облак

IoT Вградените системи е необходимо да комуникират помежду си или да изпращат определени данни към сървър. Данните могат да бъдат определен тип. Те се генерират от сензорите свързани към вградената система и трябва да бъдат изпратени за обработка към централен компютър. За тази цел се използват сървъри или cloud системи, като тяхната роля е да поддържат комуникацията между вградените системи и обработката на данни. Най често сървърната част взима решение на базата на постъпилите данни и праща готов отговор към вградената система за изпълнение. Но използването на сървъри в днешно време се заменя със Cloud платформи, поради факта че притежават преимущества пред тях. Преимуществата могат да бъдат по-ниски разходи, по – съвършен модел, по-добра сигурност. Друго основно преимущество е ,че системата може да скалира, т.е. да се наложи да се свържат много вградени системи към сървъра, а това винаги е свързано със повече налични ресурси като изчислителна мощ и оперативна памет. За тази цел Cloud базираните системи са по-гъвкави и при по- голямо натоварване могат да бъдат освободени повече ресурси. Други предимства са че системата е с ниска цена, притежава непрекъсваемост на електрозахранването и не е необходимо изграждането и от страна на клиента. Разгледаните технологий в статията са Cloud базирани системи като AWS IoT (Amazon Web Service IoT), Azure IoT , както и IBM Watson IoT. Комуникацията на вградените системи със облачните структури се осъществява чрез използване на отвърдени протоколи създадени за интернет на нещата. Такива протоколи са MQTT, CoaP, XMPP, AMQP, REST, TCP socket и т.н. Може да се използват и други протоколи, разработени само за дадена система, с цел по-голяма сигурност или ниска консумация на енергия. Целта на статията е да представи начините за синхронизация на вградените системи свързани в интернет - IoT (Internet of things) с облачни структури. Разлеждат се облачните структури и начините за комуникация с IoT устройства.



Фиг.4.8. Архитектура на свързана IoT вградена система към Cloud Server - обновяване на Firmware

Съществуват различни варианти за връзка иа обмен на данни с обновяващия сървър (Cloud). Основната идея е вградената система периодично да прави запитване към сървъра или сървъра да уведомява за съществуващо обновяване на firmware на вградената система. При наличен ъпдейт и връзка със сървъра, вградената система може да премине към обновяване. Нужно е да се осигури достатъчно надежно транспорно ниво, с цел да бъде записан новият софтуер от сървъра на вградената система. Избора на физическата среда за комуникация трябва да бъде основен момент, който трябва да се определен от предназначението на вградената система. Най сигурен вариант е Ethernet или обновяване по Lan мрежата. Но в ситуации където няма изградена кабелна мрежа се налага да се използва безжична среда за пренос на данни. Безжични среди за пренос на данни могат да бъдат Bluetooth, ZigBee, WiFi, LoraWAN и други. За изграждане на

преноса на данни е необходимо използването на протололи за обмяна на данните между тях. Примерно може да се използва протокол на транспорно ниво TCP, MQTT, CoAP, XMPP, JMS, AMQP и REST. На фиг.4.8 е показана опростена схема за връзка на вградена система (IoT) и сървър (Cloud). Вградената система ESP8266 притежава вграден WiFi модул, който ще служи за връзка със мрежата. За сървърна технология могат да бъдат използвани Apache, Nginx, Node JS и други.

Изводи от II глава и насоки за продължаване на изследването

В глава II са описани принципите и моделите на IoT протоколите, устройствата и Облака към който трябва да се свържат устройствата. Описани са в детайли как да се реализира MQTT клиент, описана е работата в детайли на MQTT протокола, както и начина за пренос на данни на ниво концепция. Същото е направено за TCP socket комуникацията , като е описана концепцията за клиент сървър. Описани са основни неща в детайли необходими за комуникацията на IoT вградената система към Облачна структура. За обновяването на софтуера е предложен принципен подход, по който ще се осъществява обновяването на flash паметта (програмната памет) на IoT вградената система от Облачна структура.

След описаното в глава II, следваща стъпка е преминаването към реализацията, която е описана в глава III. В глава II са описани основни насоки за изграждането на опитната постановка, в която трябва да се включат различни устройства, симулиращи работата и синхронизацията със Облачната структура.

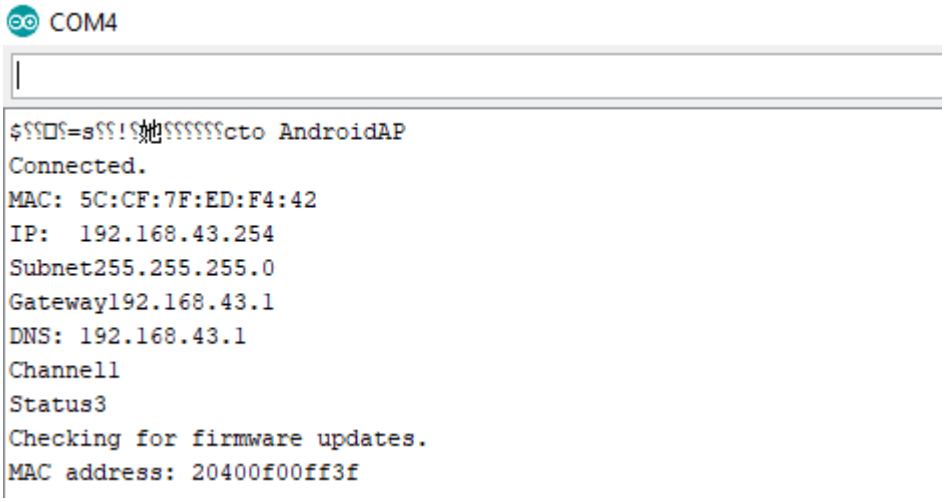
ГЛАВА 3. Реализация на опитна постановка

3.2 Изграждане, интеграция и програмиране на IoT вградени системи

Иползваните вградени системи са NodeMcu фиг.4.9 , STM32 IoT Node фиг. 5.1 и Raspberry Pi 2.0 фиг.5.2[18]. Двете платки са оборудвани с необходимата елементна база, която позволява да се извърши комуникация към безжична мрежа – WiFi. Притежават радио модул, който извършва досъпа на физическо ниво към мрежата. За реализацията на софтуера се използват средите като Arduino IDE, Iar Embedded Workbench, който ще бъдат обяснени. Програмирането при вградените системи е реализирано със отделен чип, който реализира хардуерен програматор – дебъгер в реално време.

Вградената система STM32 IoT Node е показана на фиг. 5.1. Тя е предназначена да се разработват приложения, който да осъществяват директна комуникация към Облака [46]. Вградената система позволява да се използват приложения използващи ниско енергиина комуникация.

Програмирането на вградената система [91] Node Mcu esp8266 се извършва през средата за програмиране Arduino IDE фиг. 5.3. На фигурата е показана конфигурацията на вградената система. Избрана е скорост на UART порта, през който ще се извършва дебъгване и комуникация с вградената система – 115200kbps, CPU frequency 80mhz и име на платката NodeMCU 0.9 (ESP-12 Module). За обратна връзка и съобщения от вградената система се използва "serial monitor" в менюто „Tools“ фиг.5.4. Съобщенията се изпращат през UART Rx и Tx линията на микроконтролера esp8266, като това се осъществява чрез функцията Serial.print(). Използван е COM порт 4 на компютъра, където е свързана вградената система.



Фиг. 5.4. Съобщения генерирани от функцията Serial.print() за състояние на вградената система

Към вградената система Node Mcu esp8266 е свързан сензор за температура и влажност DHT 22, както и светодиод . Схема на свързване е показана на фиг.5.5 . Идеята е сензора да се чете от микроконтролера а светодиода да се управлява отдалечено през Облака, Web и мобилно приложение. Сензора за температура и влажност е свързан на пин D2 на вградената система, а DHTTYPE оказва че типа на използваният сензор е DHT 22.

Вградената система се конфигурира чрез следните редове (4):

```
//#define DHTTYPE DHT11   // DHT 11                                    (4)
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
DHT dht(D2, DHTTYPE);
```

Функцията dht се използва за инициализация на сензора за температура и влажност. LED диода е свързан на пин D5 и се управлява от функцията digitalWrite (D5, HIGH), която в случая вдига въс високо ниво съответния пин D5 и светва LED диода .

3.3 Технологии, нужни при изграждането на Облачна струтура

Вградените системи трябва да комуникират и обменят пряко данни помежду си [43] [73] . Това за да се осъществи те трябва да бъдат свързани към общ сървър или група от сървъри в зависимост от техния брой или натоварване. Важно е да се подберат подходящи технологии, който трябва да са сигурни, надежни и бързи с цел да обработят данните и трафика от вградените системи. Важна е архитектурата на системата, понеже след като се добавят нови и повече на брой IoT устройства е нужно тя да може да скалира.

За целите на дизетационния труд за реализиране на Private Cloud са използвани технологиите Epoll Server написан на C++ , MQTT брокер Mosquitto, MQTT client написан на C++ и технологията на IBM Node Red използващ Node js .

3.3.1 Реализация на Epoll Server

За реализацията на Epoll Server е използван езика C++, като за компилация на програмния код се използва инструмента cmake [48] [49][50]. Инструмента cmake е с отворен код, нногоплатформен и предназначен за изграждане, тестване и пакетиране на софтуер. Използва се за управление на процеса на компилиране на софтуер, като се използват обикновенни

конфигурационни файлове, който са независими от платформата и компилатора и генерират така наречените „make files“.

Epoll сървъра работи на принципа устройствата да се свържат към Облака и след като изпратят съобщение то преминава през обработка и се изпраща пак съобщение като отговор към IoT вградената система. Вградената система за да предаде или прочете информация е необходимо да се свърже , да отвори сокет и да изпрати съобщение със текущата температура, влажност и състояние на LED диода. Epoll сървъра трябва веднага да отговори със съответният резултат за вградената система, примерно ако мобилното приложение се е свързало към облака и LED диода е активиран през него, тогава сървъра ще предаде съответния параметър към IoT вградената система. Така ще се задейства LED диода от страна на вградената система.

Epoll сървъра използва така наречения Ring Buffer, като в него се съхраняват съобщенията. Променливата **ring_buffer** се използва за съхраняване на съобщенията престигащи от IoT устройствата и мобилното приложение, а **DEFAULT_RING_BUFFER_SIZE** се използва за определяне на размера на опашката на буфера, в случая 1024 (5) :

```
auto ring_buffer = new processor::RingBuffer<event_data>(DEFAULT_RING_BUFFER_SIZE);          (5)
```

Сървъра се състой от main() функция, клас Device(), функция process_messages() и event_loop(), както и локални променливи. Класът Device() дефинира конструктор, деструктор, локални private променливи, get() и set() методи за достъп до private променливите (6):

```
private:                                                                                    (6)
   int    ddeviceID;
   string ttype;
   string nname;
   float  hhumidity;
   string sstate;
   float  ttemp;
```

Private променливите **hhumidity** , **sstate** , **ttemp** служат за съхраняване на температурата, влажността и състоянието изпратени от вградената система. Променливите **ddeviceID, ttype, nname** служат за идентификация на вида на устройството.

Методите get () и set() служат за достъпване на private член променливи (7):

```
int get_deviceID(){                                                                         (7)
     return ddeviceID;
 }
 void set_deviceID(const int deviceID){
    ddeviceID = deviceID;
 }
```

Функцията int process_messages(processor::RingBuffer<event_data>* ring_buffer) се използва за изпращане на съобщения от сървъра към вградената система. В нея се дефинират следните променливи необходими за работата на сървъра (8):

```
string bufferS;                                                    (8)
Json::Value root,AndroidRoot;
Json::Reader reader;
string stateSensor = "off" ;
```

Root променливата се използва са сравняване на данните, който ще се изпращат към вградената система (9):

```
if(root["type"] == "IoTdevice"){                                   (9)
    string formatS;
    sensor.set_deviceID(atoi(root["deviceID"].toStyledString().c_str()));
    formatS = root["type"].toStyledString();
    sensor.set_type(formatS.substr( 1 ,(formatS.length() -3 )));
```

От тук се вижда, че се проверява съдържанието на променливата **root["type"] == "IoTdevice",** като когато нейната стойност е **IoTdevice** тогава се обработва, като се взима стойността от полето на JSON файла, в случая се взима deviceID или уникалният номер на устройството свързано към сървъра. Този номер ще се извете в конзолата на Epoll сървъра чрез следния ред (10):

```
printf("\n %d ",sensor.get_deviceID() );                            (10)
```

При варианта когато **root["type"]** е изпратен от мобилно приложение се обработва следните редове код (11):

```
if(root["type"] == "IoTdevice"){                                   (11)
    string formatS;
    sensor.set_deviceID(atoi(root["deviceID"].toStyledString().c_str()));
    formatS = root["type"].toStyledString();
    sensor.set_type(formatS.substr( 1 ,(formatS.length() -3 )));
```

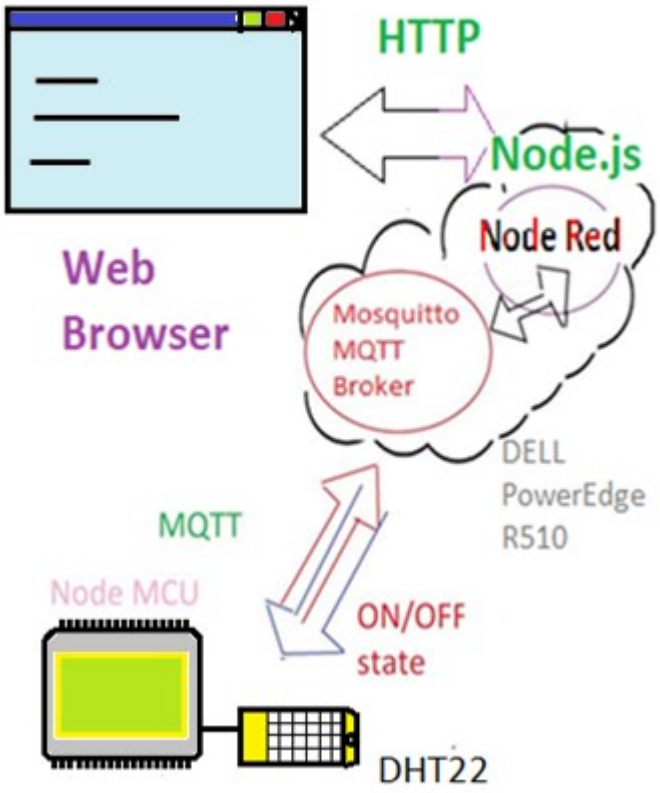
Функцията process_messages работи непрекъснато в цикъл while(), като тя изпраща данни през сокет към устройството, било то вградена IoT система или мобилно приложение. Функцията void event_loop(int epfd,int sfd, processor::RingBuffer<event_data>* ring_buffer) се използва за приемане на съобщения, изпратени от вградените системи.

3.3.3 **Реализиране на Облачна структура с Node Red IBM**

Съществуват различни разновидности на IoT Облак [96][28][42][99].В зависимост от предназначението и вида на използването му, може да бъде частен и публичен облак. Публичния е удобен в случай, когато нямаме конфеденциални данни и системата е скалируема . Примерно броят на свързаните устойства се увеличава непрекъснато, в следствие на което расте натоварването на сървъра и е нужно все повече изчислителна можност. Това при частен облак не е подходящо решение, понеже са нужни системни администратори и техници да поддържат облачната структура, да се добавят повече и повече сървъри. Но относно публичния недостатъците са, че има оязвимост относно защита на личните данни, както и някой платформено зависими особености.

За изграждането на частен облак е използван сървър Dell Poweredge R510, като е инсталирана Centos 7 операционна система [63] [76]. За сървърна технология е използван Node Red, който е инструмент за разработка, базиран на потоци и разработен от IBM, като идеята е да свърже заедно хардуерни устройства, APIs и Online services, който са част от Internet of Things [77] [61].

Node red е редактор работещ в баузъра и удобен за създаването на JavaScript функции. Приложението може да бъде съхранявано или споделяно, както и използвано отново. За да работи Node Red е нужно да се инсталира сървърната технология Node.js – runtime. Проекта описан в Node Red се съхранява, като се използва JSON тип формат, както за комуникация към сървъра се използва и TLS криптирана връзка [58].



Фиг. 5.7. Архитектура на Облачната структура изградена с Node Red

Относно вградената система, която се свързва към Облака реализиран с Node Red се използва вградената система NodeMcu Esp8266, към която е свързана сензор за температура и влажност DHT 22 [59]. Архитектурата на Облака е показана на фиг. 5.7 а самата вградената система на фиг.5.8.

За изграждане на Облака са използвани следните технологии – Node Red, Node.js, Mosquitto MQTT broker инсталирани на Centos 7 операционна система върху сървър Dell Poweredge R510. За изграждане на backend часта се използва Web базираната среда на Node Red [57]. Тя се извиква като се стартира сървъра чрез командата в терминала :

> node-red

след което системата съобщава - Server now running at http://127.0.0.1:1880 . Чрез въвеждане в Web браузъра а IP адреса и порта на процеса на Node Red, се зарежда графичната среда за разработка фиг. 5.9.

Вградената система NodeMcu Esp8266 трябва да се свърже към MQTT Mosquitto Broker, както и Node Red. За тази цел в полето търсачка за функционални модули в Node Red се избират „mqtt input nodes“ за температурата и влажността, като те се настройват да се свържат към MQTT Mosquitto broker. За тази цел се задават следните настройки показани на фиг.6.0 , localhost:1883 което е адреса и порта на MQTT broker server и име на Topic - temperature.

Относно софтуерната реализация на вградената система Esp8266 е използвана MQTT библиотека , като вградената система се свързва към MQTT Broker server. Вградената система

изпраща периодично през 10 секунди към брокера стойностите отчетени от сензора DHT22 за температура и влажност. Тя следи непрекъснато за команди от MQTT broker, в случая за активиране на светодиода през Web browser от страна на потребителя. Esp8266 работи като Publisher/Subscriber.

3.4 Реализиране предаване на данни през TCP sockets

Вградената система се свързва към Epoll сървъра чрез TCP sockets. Чрез тях тя обменя данни, като данните са пакетирани в JSON формат [55]. Формата на протокола е следният (26):

```
{ "deviceID" :  1,                                                    (26)
  "type"          :  "IoTdevice",
  "humidity"   :  42,
  "name"        :  "ESPDH22",
  "state"         :   "on",
   "temp"        :    28 }
```

Полетата "deviceID" , "type", "humidity" , "name", "state" и "temp" се използват за формиране на протокола за пренос на данни между IoT вградената система и Облачната структура. Както се вижда на тях се присвояват стойности . Примерно deviceID се използва за уникален номер на устройствата, като всяко си има свой номер, който в случая е единица. Полето type оказва какъв е типа на устройството, като в случая е IoTdevice, но може да бъде и mobile – за мобилно приложение. Полето name оказва името на вградената система, в случая това е ESPDH22 което е esp8266 а DH22 е типа на сензора. Полето за състояние state може да бъде on или off, като когато е on IoT устройството трябва да включи LED диода. Полетата humidity и temp показват влажността и температурата отчетени от вградената система. На таблица 5 са показани полетата в обобщен вид.

Вградената система STM32 IoT Node използва същият протокол за комуникация към Epoll сървъра, като разликата е че deviceID е друга стойност , както и name е различен. На таблица 6 е показан формата на протокола за комуникация.

ТАБЛИЦА 5
ПОЛЕТА НА JSON ФОРМАТА НА IOT ВГРАДЕНАТА СИСТЕМА NODE MCU ESP8266

Име на полето	Примерна стойност	Тип на полето	значение
"deviceID"	1	Integer	Уникален номер
"type"	"IoTdevice"	string	Тип IoT устройство
"humidity"	42	integer	Влажност
"name"	"ESPDH22"	string	Име IoT устройство
"state"	"on"	string	Състояние
"temp"	28	integer	Температура

ТАБЛИЦА 6
ПОЛЕТА НА JSON ФОРМАТА НА IOT ВГРАДЕНАТА СИСТЕМА STM32 IOT NODE

Име на полето	Примерна стойност	Тип на полето	значение
"deviceID"	2	Integer	Уникален номер
"type"	"IoTdevice"	string	Тип IoT устройство

"humidity"	15	integer	Влажност
"name"	" STM32 IoT node "	string	Име IoT устройство
"state"	"on"	string	Състояние
"temp"	22	integer	Температура

Програмната реализация на Node Mcu esp8266 вградена система се реализира чрез средата за разработка Arduino IDE. Създава се обект clientTCP от тип WiFiClient, който се използва за отваряне на сокет към Облака (27):

```
WiFiClient         clientTCP;                                                    (27)
if (!clientTCP.connect("78.83.114.192", 3000)) {
    Serial.println("Connection to echo server failed");
    while (true) {}
  }
  clientTCP.print(jsonS)
```

като в кода се вижда, че се използва порт 3000 и IP адрес 78.83.114.192, който са предадени като параметър към функция **clientTCP.connect("78.83.114.192", 3000).** Функцията **clientTCP.print(jsonS)** изпраща JSON протокола, който се съдържа в променливата jsonS. Получаването на протокол от Облака към IoT вградената система се осъществява чрез следният код (28):

```
while (true) {                                                                  (28)
    while (clientTCP.available() == 0) {}
   temp[i] = clientTCP.read();
```

където функцията **clientTCP.available()** следи за пристигнали пакети от WiFi радио модулна на esp8266, а след като пристигне пакет се променя условието на цикъла, като функцията **clientTCP.read()** прочита данните.

Сериализирането и десереализирането на JSON протокола се извършва чрез обекта rootT от тип JsonObject, като се извършва така нареченото парсване – parse (29) :

```
jsonT = temp;                                                                   (29)
JsonObject& rootT = jsonBufferT.parseObject(jsonT);
const char* sensor = rootT["sensor"];
const char* tempC = rootT["temp"];
const char* state  = rootT["state"];
```

като се попълват променливите **sensor, tempC , state** от полетата на JSON протокола.

Програмната реализация на IoT вградената система STM32 IoT Node се реализира чрез средата за разработка IAR Embedded Workbench IDE. Програмният език е C. Използва се функцията за изпращане на данни към Облака (30) [54]:

```
WIFI_SendData(Socket, responce, sizeof(responce), &Datalen, WIFI_WRITE_TIMEOUT);   (30)
```

като променливата response се използва за сериализираният JSON протокол, в който са пакетирани параметри за температурата, влажността и състоянието на LED диода. Използва се функцията за получаване на данни от Облака (31):

```
WIFI_ReceiveData(Socket, RxData, sizeof(RxData), &Datalen, WIFI_READ_TIMEOUT);   (31)
```

3.5 Реализиране предаване на данни през MQTT

Вградената система се свързва към MQTT брокера Mosquitto и предоставя информацията до клиентите абонирани за дадената тема. Данните са пакетирани в JSON формат. Формата на протокола е следният (32):

```
{ "deviceID" :  1,                                                    (32)
   "type"            :  "IoTdevice",
  "humidity"   :  42,
  "name"        :  "ESPDH22",
  "state"         :   "on",
  "temp"        :    28 }
```

Полетата "deviceID" , "type", "humidity" , "name", "state" и "temp" се използват за формиране на протокола за пренос на данни между IoT вградената система и Облачната структура. Както се вижда на тях се присвояват стойности . Примерно deviceID се използва за уникален номер на устройствата, като всяко си има свой номер, който в случая е единица. Полето type оказва какъв е типа на устройството, като в случая е IoTdevice, но може да бъде и mobile – за мобилно приложение. Полето name оказва името на вградената система, в случая това е ESPDH22 което е esp8266 а DH22 е типа на сензора. Полето за състояние state може да бъде on или off, като когато е on IoT устройството трябва да включи LED диода. Полетата humidity и temp показват влажността и температурата отчетени от вградената система. На таблица 7 са показани полетата в обобщен вид.

ТАБЛИЦА 7
ПОЛЕТА НА JSON ФОРМАТА НА IOT ВГРАДЕНАТА СИСТЕМА NODE MCU ESP8266

Име на полето	Примерна стойност	Тип на полето	значение
"deviceID"	1	Integer	Уникален номер
"type"	"IoTdevice"	string	Тип IoT устройство
"humidity"	42	integer	Влажност
"name"	"ESPDH22"	string	Име IoT устройство
"state"	"on"	string	Състояние
"temp"	28	integer	Температура

Програмната реализация на Node Mcu esp8266 вградена система се реализира чрез средата за разработка Arduino IDE [56]. Създава се обект clientTCP от тип WiFiClient, който се използва за отваряне на сокет към Облака (33) , връзка към брокера (34), абониране за тема (35) и изпращане на съобщението (36) :

```
PubSubClient clientMQTT(mqtt_serverS, 1883, wifiClientS);
  (33)


if (clientMQTT.connect(clientIDS, mqtt_usernameS, mqtt_passwordS)) {
  (34)
   Serial.println("Connected to MQTT Broker!");
 }
clientMQTT.subscribe("test");                                          (35)
clientMQTT.publish(mqtt_topicS, string2char(jsonS));                   (36)
```

3.6 Реализация на мобилно приложение за контрол на IoT устройства

Андроид мобилното приложение служи за управление на светодиода на вградената система Node Mcu esp8266 [53]. То се свързва към Epoll сървъра на порт 3000 чрез TCP web sockets. Мобилното приложение управлява LED диода и визуализира параметри като температура, влажност и състояние на светодиода. Графичната част на приложението е представена на фиг. 6.2. Използвани са контроли в UI интерфейса като текстови полета и бутони. Текстовите полета са temp, humidity и state, а бутоните connect, disconnect, on LED и off LED. Използваната технология за реализацията на мобилното приложение е Android Studio 3.0, а програмния език за програмиране е Java.

Формата на JSON протокола за комуникация мобилното приложение към Облака е следният (37):

```
{   "deviceID" : "3",                                                    (37)
    "name" : "Android",
    "state" : "on",
    "type" : "mobile" }
```

като deviceID оказва уникалния номер на IoT устройството, name е името на устойството а type е типа на устройството. Полето state се използва за включване и изключване на LED диода на вградената система. На таблица 8 са показани полетата в обобщен вид.

ТАБЛИЦА 8
ПОЛЕТА НА JSON ФОРМАТА НА АНДРОИД МОБИЛНО ПРИЛОЖЕНИЕ ЗА КОМУНИКАЦИЯ КЪМ ОБЛАКА

Име на полето	Примерна стойност	Тип на полето	значение
"deviceID"	2	Integer	Уникален номер
"type"	"IoTdevice"	string	Тип IoT устройство
"name"	" STM32 IoT node "	string	Име IoT устройство
"state"	"on"	string	Състояние

Формата на протокола от Облака към мобилното приложение е същият, който е изпратен от IoT устройството, таблица 8.

Архитектурата на мобилното приложение е следната, то притежава клас Client.java и клас MainActivity.java. Класът описващ клиента притежава get() и set() методи, с който се достъпва резултата от сървъра, конструктор, метод doInBackground() и променливи. Променливите в класа са от тип private (38):

```
private String dstAddress;                                               (38)
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";
```

като променливата sendData се използва за изпращане на данни към сървъра, която е от тип JSON, dstAddress е името на адреса към който се свързва мобилното приложение, а response е от тип String, което е типа на върнатия резултат от сървъра.

ГЛАВА 4. Експериментални резултати от изследването

4.1. Сравнение протоколи за комуникация на IoT вградени системи

Съществуват много стандарти и протоколи за комуникация на IoT устройства. IoT вградените ситеми се използват навсякъде, както в индустрията , уните домове, за военни цели и навсякъде. IoT вградените системи могат да четат информация от сензори, да управляват двигатели, машини, релета и т.н. IoT устройствата трябва да работят във определена среда, те трябва да комуникират помежду си, както събират и изпращат данни към Облака. В зависимост от целта и средата в която ще се използват тези вградени системи, съществуват различни протоколи и физически среди.

Протоколите реализирани на високо ниво за Интернет на Нещата – IoT , предлагат различни функции, който ги правят подходящи за широка дама от приложения. IoT протоколите имат различни функции и предлагат различн възможности [65] [66] [67]. Повечето от тези протоколи са разработени от определени разработчици, който обикновенно популяризират собственият си протокол [69].

ТАБЛИЦА 9
ПРЕДИМСТВА И НЕДОСТАТЪЦИ НА IOT ПРОТОКОЛИТЕ

Протокол	Предимства	Недостатъци
MQTT	- Идеален е за нискоенергиина консумация (атерийно захранване) - Изисква олекотени ресурси за изграждане на приложен стек - Хедъра на съобщенията може да бъде малък, колкото 2 байта. Това го прави много ефективен относно широчина на връзката, както е идеален за ограничени мрежи - Поддържа всякакви съобщения в мрежата (publish / subscribe и request / reply) - MQTT SN поддръжа topicID вместо topic name, UDP, ZigBee, Bluetooth и други wireless протоколи	- Не се поддържа опашка от съобщения, като единствено най последните съобщения се съхраняват в брокера - Не се поддържат header полета като TTL (Time to live), изпратени към потребителя - Няма секция за пропъртита на съобщенията
CoAP	-Има същата концепция като REST модела, но като изключим TCP -Много бърза device-to-device комуникация през UDP	-Има същите слабости като REST , но като изключим качеството на нивото на сървиза -Предлага променливо ниво на сървиза -Поддържа единствено request-reply модел на съобщенията
XMPP	-разширен протокол за съобщения -използва XML текстов формат като native тип -осъществява person-to-person комуникация	-няма съществени недостатъци

	-използва TCP	
AMQP	-поддържа повечето модели за обмяна като publish/subscribe, request-reply и опашка на съобщенията - поддържа всички класове на сървиза -поддържа детайлни хедърни полета като TTL, replyTo и потребителски пропъртита -позволява портабилно кодиране на съобщенията -поддържа както TCP, така и UDP	-Използва доста от захранването, обработката и паметта. За IoT вградени системи изискванията са по високи -изискваните хедърни полета са сравнително дълги
HTTP	-не изисква клиентска библиотека да се поддържа от IoT устройството -опростява архитектурата, ако е позволено да има загуба на данни -осигурява 'lowest common denominator' свързаност, след като повечето устройства използват GET или POST	-Хедърните полета са сравнително дълги -няма подръжка за качеството на сървиза -приложението трябва да се справи с цялата надеждност

ТАБЛИЦА 10
АРХИТЕКТУРА, УПОТРЕБА, ИЗПОЛЗВАНИ РЕСУРСИ И ТРАНСПОРТ

Протокол	Архитектура	Употреба	Използвани ресурси	транспорт
MQTT	Tree	IoT приложения - съобщения	10Ks RAM/Flash	TCP / IP
CoAP	Tree	IoT приложения - файлове	10Ks RAM/Flash	UDP / IP
XMPP	Client Server	за децентрализирани IoT системи	10Ks RAM/Flash	TCP / IP
AMQP	Client Consumer	Комуникация между различни доставчици, Облачни изчисления	В зависимост от големината на данните 100Ks RAM/Flash	TCP / IP
HTTP	Client Server	Energy management, Gateways,home services	10Ks RAM/Flash	TCP / IP

ТАБЛИЦА 11
МОДЕЛ СЪОБЩЕНИЯ, УПОТРЕБА В МОБИЛНИ МРЕЖИ, ЗАХРАНВАНЕ И ЗАЩИТА

Протокол	Съобщения	2G,3G,4G	При слабо захранване	Защита
MQTT	Publisher/Subscriber	Отличен	Добър	Средна SSL
CoAP	Request/Responce	Отличен	Отличен	Средна DTLS
XMPP	Publisher/Subscriber	Отличен	Лош	Висока TLS/SSL
AMQP	Publisher/Consumer	Отличен	Лош	Висока TLS,SASL
HTTP	Request/Responce	Отличен	Лош	Ниска
HTTPS	Request/Responce	Отличен	Лош	Висока TLS/SSL

4.2 Изследване на работата на IoT вградена система

За нуждите на експеримента са използвани вградените системи NodeMcu Esp8266 и STM32F475 IoT node. Тяхната употреба и интеграция е подробно описана в глава III. На фиг. 7.7 е показан опит за свързване на NodeMcu Esp8266 вградена система към интернет - WiFi мрежа. На фиг. 7.8 е показано успешно свързване на NodeMcu Esp8266 към интернет мрежата през WiFi и изпращане на данни към Облака. На фиг. 8.1 е показано свързването на STM32F475 IoT node към WiFi мрежата. За WiFi Hotspot е използван Android мобилен телефон, играещ ролята на рутер, като името на SSID е AndroidAP. На фиг.7.9. е показано получаването на данни от Облака на вградена система NodeMcu Esp8266.

На фиг.8.0 е показано задействане с високо ниво (включване) на LED диода на NodeMcu вградената система. IoT устройството е получило команда от Облака за задействане на LED диода, а на конзолата се вижда притетата и потърдена команда – „on“.

STM32F475 IoT node вградената система след подаване на захранване трябва да инициализира платката, WiFi радио модула и периферията като датчици за температура, влажност и входно-изходни портове GPIO. На фиг. 8.1 е представена нейната инициализация, както и успешното свързване към Облака с IP адрес 78.83.114.192 на порт 3000 към Epoll сървъра. Формата и протокола на данните е идентичен със този на NodeMcu Esp8266 вградена система. На фиг. 8.2 е представено изпращането на JSON пакет (1) към Облака , съдържащ параметри за температурата и влажността (54) :

4.2 Изследване на работата на IoT вградена система

За целта на изследването е използвана IoT вградената система NodeMcu Esp8266, която се свързва към Облака чрез протокол MQTT. Чрез програмата за анализ на мрежа - Wireshark е направен анализ на MQTT пакетите между IoT устройството и Mosquitto брокер. Вградената

система е настроена така, че се изпращат поредица от ASCII стрингове с различна сължина – фиг.8.3, фиг.8.4, фиг.8.5, фиг.8.6, фиг.8.7. На фиг. 8.3 е показана анатомията на иззпратен един символ - „1“ с тема “test” от вградената система. Вижда се съобщението, като полето Data не е с големина 1 байт а цели 9 байта, като причината е че се изпраща и служебна информация като темата на съобщението. Дължината на MQTT пакета съдържащ съобщението - Frame е цели 63 байта. На фиг. 8.4 е показано изпращането на два символа - „12“, на фиг. 8.5 е показано изпращането на три символа - „123“ , на фиг. 8.6 е показано изпращане на седем символа - „1234567“ а на фиг. 8.7 е показано изпращане на 91 символа“ към Облака.

Всички резултати са снети от фиг.8.3, 8.4, 8.5, 8.6 и 8.7, като те са обощени в Таблица 12. На таблицата се виждат всичките стойности на параметрите при определените изпратени съобщения.Вижда се че служебната информация на пакета MQTT – Header е постоянна стойност от 62 байта. На фиг. 8.8 е представена графиката от стойностите на Таблица 12. На нея се вижда, че е линейна графиката на зависимостта от Data Payload и Packet Frame .

ТАБЛИЦА 12
ИЗСЛЕДВАНЕ НА MQTT ПРОТОКОЛ В ЗАВИСИМОСТ ОТ РАЗМЕРА НА СЪОБЩЕНИЕТО

Съобщение ASCII	Съобщение bytes	Packet Frame Bytes	Data Payload Bytes	Topic	Header Bytes
„1“	1	63	9	“test”	62
„12“	2	64	10	“test”	62
„123“	3	65	11	“test”	62
„1234“	4	66	12	“test”	62
„12345“	5	67	13	“test”	62
„123456“	6	68	14	“test”	62
„1234567“	7	69	15	“test”	62
„1234567..“	91	153	99	“test”	62

4.4 Изследване на реализиран протокол през TCP sockets за пренос на данни

За целта на изследването е използвана IoT вградената система NodeMcu Esp8266, която се свързва към Облака чрез TCP sockets. Чрез програмата за анализ на мрежа - Wireshark е направен анализ на TCP socket пакетите между IoT устройството и Epoll сървъра. Вградената система е настроена така, че се изпращат поредица от ASCII стрингове с различна сължина – фиг.8.9, фиг.9.0, фиг.9.1, фиг.9.2, фиг.9.3. На фиг. 8.9 е показана анатомията на изпратен един символ - „1“ с тема “test” от вградената система. Вижда се съобщението, като полето Data е с големина 1 байт. Дължината на TCP sockets пакета съдържащ съобщението - Frame е цели 60 байта. На фиг. 8.9 е показано изпращането на три символа - „123“, на фиг. 9.0 е показано изпращането на пет символа - „12345“ , на фиг. 9.2 е показано изпращане на шест символа - „123456“ а на фиг. 9.3 е показано изпращане на седем символа“ към Облака.

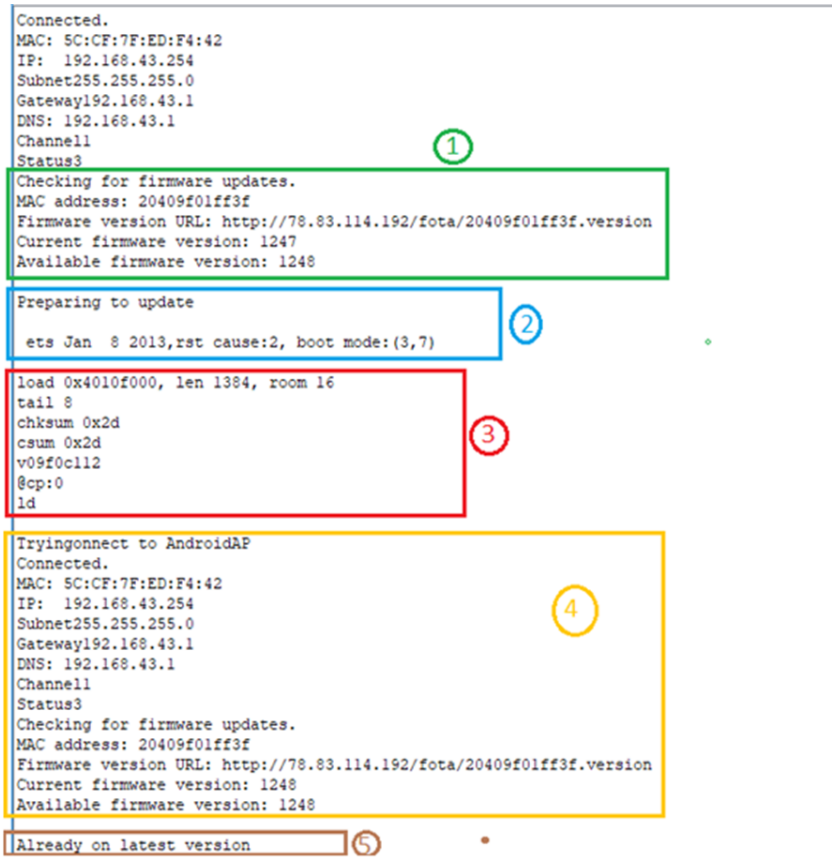
Всички резултати са снети от фиг.8.9, 9.0, 9.1, 9.2 и 9.3, като те са обощени в Таблица 3. На таблицата се виждат всичките стойности на параметрите при определените изпратени съобщения.Вижда се че служебната информация на пакета TCP socket – Header е променлива стойност, която зависи от изпратените байтове. На фиг. 9.4 е представена графиката от стойностите на Таблица 13. На нея се вижда, че е почти линейна графиката на зависимостта от Data Payload и Packet Frame.

ТАБЛИЦА 13
ИЗСЛЕДВАНЕ НА TCP SOCKET ПРОТОКОЛ В ЗАВИСИМОСТ ОТ РАЗМЕРА НА СЪОБЩЕНИЕТО

Съобщение ASCII	Съобщение bytes	Packet Frame Bytes	Data Payload Bytes	Header Bytes
„1“	1	60	1	59
„12“	2	60	2	58
„123“	3	60	3	57
„1234“	4	60	4	56
„12345“	5	60	5	55
„123456“	6	60	6	54
„1234567“	7	61	7	54
„1234567..“	91	145	91	54

4.5 Изследване процеса обновяване софтуера на вградена система

На фиг. 9.7 са показани стъпките при обновяване Firmware на IoT вградената система от Облака. Стъпка-1 е проверката на нов firmware на Облака, като се сравняват текущата версия инсталирана на IoT устройството със тази хостната на Облака. При наличие на по висока версия се преминава към обновяването , стъпка-2. При стъпка-3 се случва самото обновяване като се налива новият firmware от облака във втория дял на вградената система esp 8266. Правят се проверки , като се сравнява Checksum –мата на инсталирания firmware със този на Облака. След като е преминало успешно обновяването се преминава към стъпка-4, която е рестарт на вградената система и отново свързване към WiFi мрежата и проверка за наличен софтуер-5.



Фиг. 9.7. Стъпки при обновяване Firmware на IoT вградената система от Облака

4.6 Изследване на работата на реализирано мобилно приложение за контрол на IoT устройства

Мобилното приложение е изградено на операционната система Андроид, като то се свързва към облачната структура и прочита данни за температурата, влажността и състоянието на LED диода фиг.9.8 . Използван е телефон LG Nexus 5 с Андроид операционна система, весия 6 Marshmallow. Бутоните „CONNECT“ и „DISCONNECT“ се използват за свързване и разкачане от Epoll сървъра, а бутоните „“ и „“ контролират дали да е включен или изключен LED диода. В случая показанията са дадени в Таблица 14

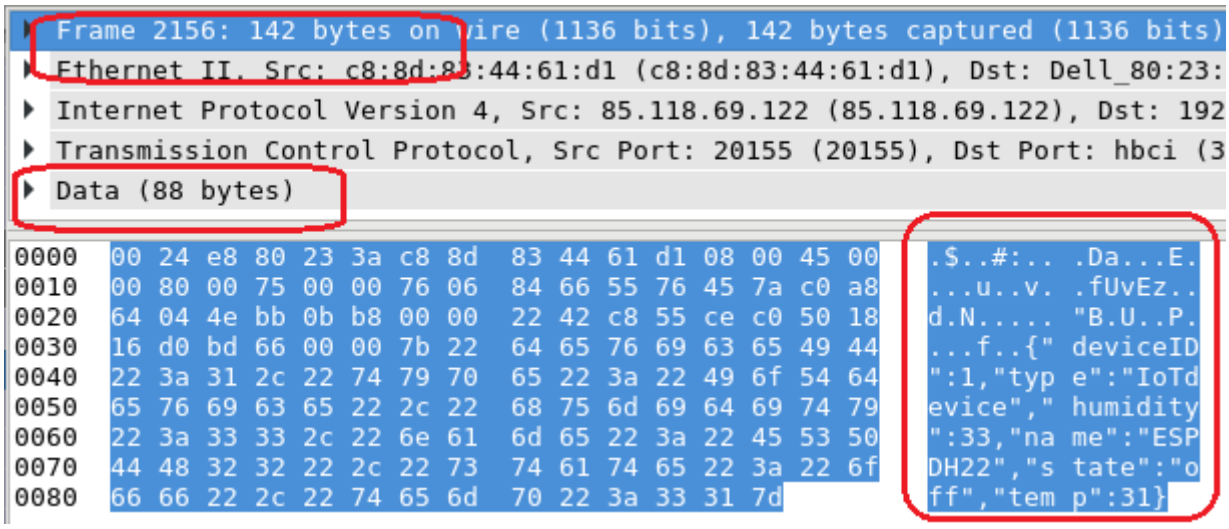
ТАБЛИЦА 14
ИЗСЛЕДВАНЕ НА АНДРОИД МОБИЛНО ПРИЛОЖЕНИЕ И ПОКАЗАНИЯ НА ОТЧЕТЕНИ СТОЙНОСТИ

Температура	Влажност	Състояние на LED диода
29.0	37.0	off

4.7.1 Изследване на работата на Epoll сървър и вградена система

На фиг. 10.0 е показано съобщение за отговор на подадено напрежение (включен) към LED диод, което се изпраща от IoT вградената система към Epoll сървъра. На фиг. 10.1 е показано сравнение между конзолата за четене на съобщения и дебъгване на IoT вградената система NodeMCU през средата Arduino IDE Terminal и конзолата на съобщения на Epoll сървъра за точност на данните. На фигурата се вижда, че Android приложението връща отговор какво е прочело и визуализирало на потребителя от Облака.

На фиг. 10.2 е показана анатомията на TCP socket пакета реализиран чрез JSON формат за пакетиране на данни. От изследването чрез програмния инструмент Wireshark се виждат параметрите на пакета, който са представени в Таблица 15.



Фиг. 10.2. Анатомия на TCP socket пакета реализиран чрез JSON формат за пакетиране на данни

ТАБЛИЦА 15
ПАРАМЕТРИ НА TCP SOCKET ПАКЕТА МЕЖДУ IOT УСТРОЙСТВОТО И ОБЛАКА

Съобщение bytes	Packet Frame Bytes	Data Payload Bytes	Header Bytes
88	142	88	54

4.7.2 Изследване на работата на Mosquitto брокер и MQTT клиент

В тази точка е изследвана работата на MQTT брокера – Mosquitto и MQTT клиент. Клиента в Облака за целта на изследването се реализира в конзолата на операционната система Centos 7, като се въведе следната команда (55):

- Mosquito_sub –v –t ‘test’ (55)

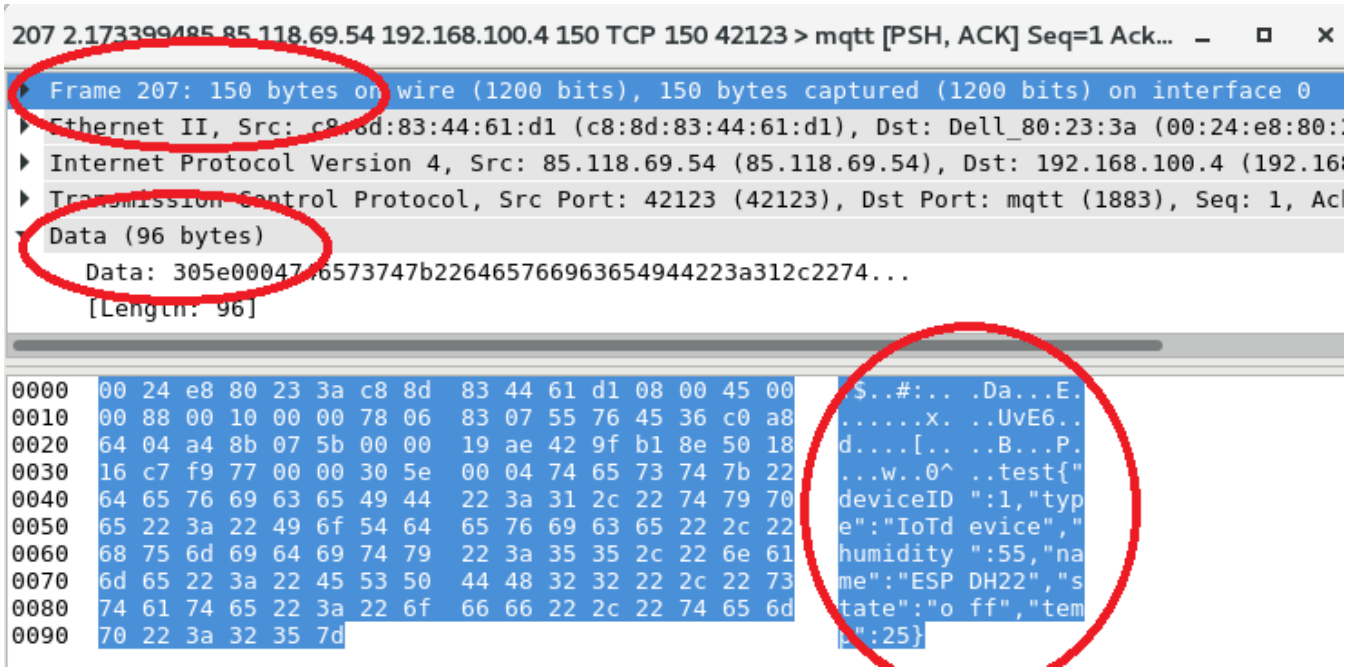
като темата на MQTT съобщението е „test“ и се стартира тестов клиент, като sub идва от Subscribe фиг. 10.3. MQTT клиента който ще изпраща данните към Облака е вградената система Node MCU Esp8266, която също е абонирана за тема на съобщението „test“. Стартира се клиента чрез команда (2) в конзолата и тя очаква да се изпрати съобщение от IoT вградената система, която играе роля на Publisher. Типа на изпратеното съобщение е JSON формат в който са пакетирани данни за температурата, влажността , състоянието на LED диода – включено или не и други полета (56) :

{"deviceID":1,"type":"IoTdevice", (56)

"humidity":54,"name":"ESPDH22",

"state":"off","temp":25}

като полетата "deviceID", "type" и "name" са за описание на IoT устойството, което изпраща данните към Облака. Вградената система Node Mcu Esp8266 изпраща JSON пакета (3) към Облака а клиента получава пакета фиг.10.3. На фиг. 38 е показана анатомията на MQTT пакета изпратен от IoT устройството, съръжащ JSON формат на данните. Резултатите от параметрите на MQTT съобщението са описани Таблица 16.



Фиг. 10.4. Анатомия на MQTT пакета изпратен от IoT устройството, съръжащ JSON формат на данните

ТАБЛИЦА 16
ПАРАМЕТРИ НА MQTT ПАКЕТА МЕЖДУ IOT УСТРОЙСТВОТО И ОБЛАКА

Съобщение bytes	Packet Frame Bytes	Data Payload Bytes	Header Bytes	Topic
88	150	96	54	„test“

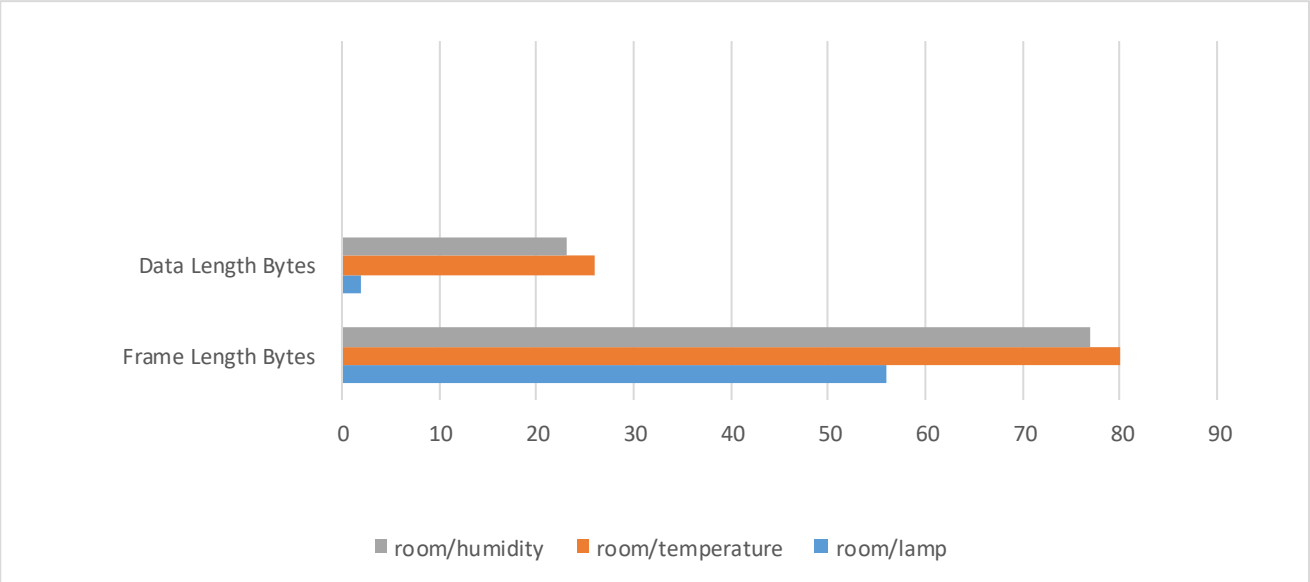
4.7.3 Изследване на работата на Node RED сървър и вградена система

В глава III е описана реализацията на сървър с Node RED и свързан към IoT вградена система. На фиг. 10.5 е представена графичната среда на потребителския Web интерфейс, която се зарежда в браузъра Chrome. Вградената система NodeMCU esp866 инициализира комуникация към сензора за температура и влажност DHT22, като прочита данните. Данните се пакетират в протокола MQTT и се изпращат към Облачната структура, реализирана с Node RED. Стойностите на измерените параметри на влажността и температурата са представени на графичния интерфейс на фиг. 10.5, като графика на име ”Temperature” показва температурните промени през различните часове. На фигурата се вижда бутона „Led“, който управлява свързаният LED диод към вградената система. В случая той е във включено състояние – „ ON “. На фиг. 10.5 също се вижда и графиката на влажността „Humidity “. Текущата отчетена температура е 22 градуса със влажност 49 % и включен LED диод.

На фиг. 10.7, 10.8, 10.9 е показана комуникацията между Облака и IoT вградената система, както е и показана анатомията на MQTT пакет на дадено съобщение към облака със определена тема. За това измерване е използвана програмата - WireShark, чрез която се изследват всички пакети и съобщения прежду сървъра Node RED и IoT устройствата свързани към него. Чрез програмата могат да бъдат видяни параметри на пакета като Frame length, Data length, да се прочете съобщението стига да не е криптирано. На фигурите се вижда съобщението в ASCII формат на MQTT протокола, понеже за целите на експеримента е използван порт 1883 на Mosquito брокера, който не е криптиран. На таблица 17 са показани параметри като Frame Length в байтове, Data length в байтове и стойностите на сензорите.

ТАБЛИЦА 17
ПАРАМЕТРИ НА MQTT СЪОБЩЕНИЯТА МЕЖДУ IOT УСТРОЙСТВОТО И ОБЛАКА

MQTT Topic	Frame Length Bytes	Data Length Bytes	Value of sensor	Sensor
room/lamp	56	2	off	LED
room/temperature	80	26	27.36	DHT22
room/humidity	77	23	40.69	DHT22



Фиг. 10.6. Големина в байтове на MQTT съобщенията

На фиг. 10.6 са показани параметрите на MQTT съобщенията между IoT устройството и Облака. От Таблица 17 може да се изчисли служебната информация - header частта на MQTT съобщенията чрез Формула 57:

$$\text{Frame length} - \text{Data length} = \text{Header MQTT Packet} \quad (57)$$

и header стойностите на MQTT пакета между Облака и IoT устройството са показани в Таблица 18

ТАБЛИЦА 18
HEADER СТОЙНОСТИ НА MQTT ПАКЕТА НА СЪОБЩЕНИЯТА МЕЖДУ ОБЛАКА И IOT УСТРОЙСТВОТО

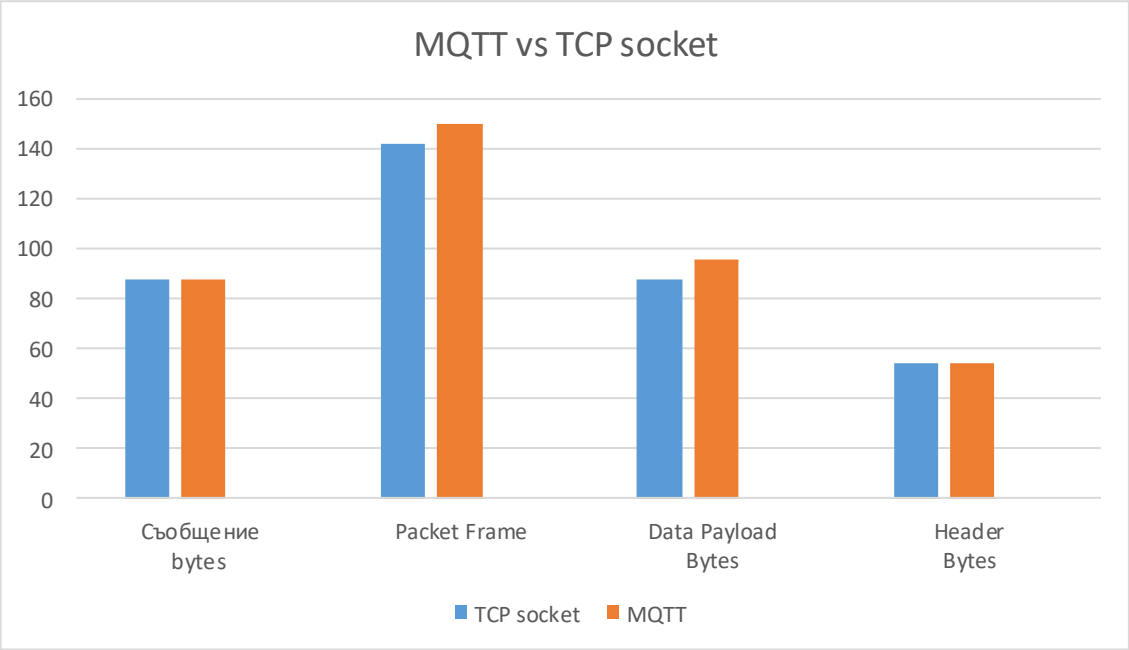
MQTT Topic	Frame Length	Data Length	Header Frame – Data =
room/lamp	56	2	54
room/temperature	80	26	54
room/humidity	77	23	54

4.9 Сравнение и анализ на получените резултати от изследването

В тази точка можем да обобщим и направим сравнение на базата на направените резултати в глава IV. Сравнява се таблица 15 и таблица 16, като база за сравнението е едно и също съобщение изпратено от вградената система през MQTT и TCP sockets протоколите. Съобщението е в тип JSON формат, както е показано на (54), като неговата големина е 88 байта. На таблица 19 е показано сравнението на двата протокола, като на фиг. 11.2 е показано графично представяне на резултатите от таблицата.

ТАБЛИЦА 19
СРАВНЯВАНЕ НА ПРОТОКОЛИТЕ MQTT И TCP SOCKET НА ЕДИН И СЪЩ JSON ПАКЕТ ИЗПРАТЕН ОТ IOT УСТРОЙСТВОТО КЪМ ОБЛАКА

Протокол	Съобщение bytes	Packet Frame Bytes	Data Payload Bytes	Header Bytes	Topic
TCP socket	88	142	88	54	-
MQTT	88	150	96	54	„test“



Фиг. 11.2. Сравняване на протоколите MQTT и TCP socket на един и същ JSON пакет между Облака и IoT устройството

От фиг. 11.2 може да се направи извода, че при изпращане на един и същ JSON пакет между Облака и IoT устройството параметъра Packet Frame е с по голяма дължина на MQTT протокола спрямо TCP socket протокола. От направения експеримент се вижда, че TCP socket протокола използва по-малко байтове за предаване на JSON формата към Облака, следователно той ще е по-подходящ в ситуации където количеството на изпратените байтове е критично. Примерно ще е по – подходящ при мрежи с нисък капацитет като мобилни и други и количеството на предадени байтове ще оскъпи цената на услугата. IoT вградената система изпраща на всеки S = 2000ms данни за състоянието на сензорите, като на таблица 20 може да се видят количеството изпратени байтове съответно за 1 час, 2 часа , 6 часа и 24 часа, както и количестото на предадени байтове за този период. Количеството данни Q за 1, 2, 6 и 24 часа можем да изчислим по следната формула (58) :

$$Q = \text{Packet Frame}\ (\text{Байтове за 2 sec}) \times (T\ (1, 2, 6, 24\ \text{часа}) \times 3600) / S \qquad (58)$$

Q – Количеството данни Q за 1, 2, 6 и 24 часа;
T – Времето за което IoT вградената система е изпращала данни;
S – Времето през което вградената система изпраща данни;

На таблица 20 е показано количеството използвани данни Q от интернет мрежата изчислено съответно формула (58).

ТАБЛИЦА 20
СРАВНЯВАНЕ НА ПРОТОКОЛИТЕ MQTT И TCP SOCKET НА ЕДИН И СЪЩ JSON ПАКЕТ ИЗПРАТЕН ОТ IOT УСТРОЙСТВОТО КЪМ ОБЛАКА ЗА КОЛИЧЕСТВОТО ИЗПОЛЗВАНИ ДАННИ ОТ МРЕЖАТА

Протокол	Съобщение bytes	Packet Frame Bytes 2 sec	Количество данни Q за 1 час	Количество данни Q за 2 часа	Количество данни Q за 6 час	Количество данни Q за 24 час

TCP socket	88	142	249.609375 kbytes	499.21875 kbytes	1497.65625 kbytes	5 990.625 kbytes
MQTT	88	150	263.671875 kbytes	527.34375 kbytes	1582.03125 kbytes	6328.125 kbytes

Изводи от IV глава

От направените измервания, таблици, графики и сравнения в клава IV може да се обобщи, че използването на протокол TCP sockets използва по малко байтове за пренос на данни отколкото протокола MQTT. Но това му предимство е подходящо когато ще използваме архитектура Клиент-Сървър за IoT цели. Протокола MQTT има предимства пред TCP sockets в ситуации, в който една вградена система трябва да изпраща данни на много други, абонирани за съответната тема - Topic. Реализираният протокол чрез JSON и TCP sockets дава по – голяма бързина, поради факта че се използва Epoll сървър, характеризиращ се със своето бързодействие. Протокола предлага по-малко пренесени байтове през мрежата, следователно при IoT устройства с батериино захранване е по-удължен живота им спрямо консумирана енергия.

НАУЧНО-ПРИЛОЖНИ И ПРИЛОЖНИ ПРИНОСИ

НАУЧНО- ПРИЛОЖНИ:

1. Предложен е вариант за комуникация между IoT вградени системи и Облачни структури със TCP socket комуникация и пренасяне на данни в JSON формат;
2. Изследван е предложения вариант за комуникация между IoT вградени системи и Облачни структури със TCP socket комуникация и пренасяне на данни в JSON формат и протокол MQTT;
3. Извършено е изследване на Облачна структура за работа, пренос на данни, обновяване на софтуера на IoT вградени системи и мобилно приложение за управление на IoT устройства;

ПРИЛОЖНИ

4. Извършено е реализиране на предложен вариант за комуникация между IoT вградени системи и Облачни структури със TCP socket комуникация и пренасяне на данни в JSON формат;
5. Извършено е реализиране на мобилно приложение за управление на IoT устройства, обновяване на софтуера на IoT вградени системи, Облачна структура, софтуерното имплементиране на протоколи като MQTT и TCP sockets, обновяването на firmware

на вградена система, четене да данни от сензори от IoT вградени системи и изпращането им на Облачна структура.

СПИСЪК НА ПУБЛИКАЦИИТЕ ПО ДИСЕРТАЦИОННИЯ ТРУД

1. **Николов, Н**.," Операционни системи за вградени системи". European University Polytechnical 2016, гр. Перник, ISSN 1314-5711
2. **Николов, Н**., "Methods for remote renovation of management software in embedded systems", European University Polytechnical 2017, гр. Перник, ISSN 1314-5711
3. **Николов, Н**., "Comunication protocols for IoT devices", 52nd International Scientific Conference on Information, Communication and Energy Systems and Technologies – Serbia , Niš, June 28 – 30, 2017, ISSN:2603-3259
4. **Николов, Н**., "Синхронизация на вградени системи, чрез оптимизиране на процеса на комуникация с обща среда разположена на облачна структура", ТЕЛЕКОМ 2017 Съюз по електроника, електротехника и съобщения (CEEC), ISSN:2603-3259
5. **Николов, Н, Наков, О**. , "Building a system for data exchange and reneval of software in embedded systems" е одобрена за публикуване в списание „Computer and Communications Engineering", 2018, ISBN 1314-2291
6. **Николов**, **Н**, **Антонов**, **С**. , "Cloud structure development for embedded systems", 53rd International Scientific Conference on Information, Communication and Energy Systems and Technologies – Sozopol, Bulgaria, June 28 – 30, 2018, ISSN:2603-3259

SUMMARY

The dissertation describes the ways of communication and updating of the control software - firmware of interconnected integrated systems - IoT. A detailed overview of existing communication protocols between embedded systems and cloud structures has been made and a communication option is proposed. A number of MQTT and TCP socket protocols have been performed using the JSON format. The innovation of IoT embedded cloud structure has been realized and researched. The experimental results are presented in tabular and graphical form, based on the results obtained. A new method of communicating and transmitting data between an embedded system and a cloud structure is proposed. The cloud structure has been studied.