Универзитет „Гоце Делчев“ – Штип

Факултет за информатика

Катедра: Компјутерски технологии и интелигентни системи

Штип

Александар Велинов

Анализа на енергетската потрошувачка на протоколи на апликациско ниво кај IoT и развој на Андроид MQTT клиент

Магистерски труд

Штип, ноември 2016

КОМИСИЈА ЗА ОЦЕНКА И ОДБРАНА

ПРЕТСЕДАТЕЛ: проф. д-р Цвета Мартиновска - Банде

Редовен професор,

Универзитет „Гоце Делчев“ – Штип,

Факултет за информатика

ЧЛЕН: доц. д-р Доне Стојанов

Доцент,

Универзитет „Гоце Делчев“ – Штип,

Факултет за информатика

ЧЛЕН-МЕНТОР: проф. д-р Александра Милева,

Вонреден професор,

Универзитет „Гоце Делчев“ – Штип,

Факултет за информатика

Датум на одбрана: 14.11.2016

Благодарност

Сакам да му изразам голема благодарност најпрво на моето семејство. Ви благодарам што верувавте во мене, што постојано бевте со мене, што ме охрабрувавте и ми помагавте како на финансиски така и на морален план. Оваа магистерска работа ви ја посветувам токму вам.

Посебна благодарност до мојата менторка проф. д-р Александра Милева, која ме водеше во целиот процес на изработка на магистерската работа. Таа постојано се ангажираше, ми помагаше и ме насочуваше. Вистински ментор.

Штип, ноември 2016 година

Објавени трудови:

1. Velinov, A., Mileva, A., “Running and Testing Applications for Contiki OS Using Cooja Simulator,” in Proceedings of the International Conference on Information Technology and Development of Education – ITRO 2016, Zrenjanin, Republic of Serbia, pp.279-285, 2016.

Трудови кои чекаат за објава:

1. Velinov, A., Mileva, A., “Power Consumption Analysis of Application Layer Protocols for the Internet of Things,” 8th ICT Innovations Conference, Ohrid, Republic of Macedonia, 2016.

Power consumption analysis of application layer protocols for the IoT and development of Android MQTT client

Abstract

Internet of Things is a concept by which a large number of devices which have different characteristics and functions are connected to each other in order to communicate and exchange data. This concept adds a new dimension to the world of computer and communication sciences. Today, the notion of the Internet of Things is used to describe scenarios for various objects, devices, sensors, vehicles and household appliances. The number of the Internet of Things devices will grow exponentially in the coming years. These devices have limited performances in terms of memory, processing power, and battery capacity. Because of the limited capacity of batteries, it is necessary to determine which technologies will be used to ensure efficiency in terms of power consumption. This is precisely the goal of this master thesis, or analysis of power consumption of several protocols at the application level for the IoT: CoAP, MQTT and XMPP. According to the test results, MQTT and CoAP provide major energy savings, unlike XMPP which consumes more power. For all protocols, the most energy is spent in a state RX, while the least is spent in a state LPM.

For the purpose of this master thesis, it is developed a MQTT client for the operating system for mobile devices – Android. It was developed using the Eclipse Paho library for Android. This client uses a publish/subscribe model. We can publish messages and subscribe to a specific topics in order to receive messages.

Keywords: Internet of Things, CoAP, MQTT, XMPP, MQTT client

Анализа на енергетската потрошувачка на протоколи на апликациско ниво кај IoT и развој на Андроид MQTT клиент

Краток извадок

Интернетот на нештата претставува концепт со кој голем број уреди што имаат различни карактеристики и функции се поврзани помеѓу себе со цел да комуницираат и да разменуваат податоци. Со овој концепт е додадена нова димензија во светот на компјутерските и на комуникациските науки. Денес, поимот за Интернет на нештата се користи за опишување сценарија за различни објекти, уреди, сензори, возила и апарати во домаќинството. Бројот на уреди кај Интернетот на нештата ќе расте експоненцијално во наредните години. Овие уреди имаат ограничени перформанси во однос на меморијата, обработувачката моќ и капацитетот на батеријата. Токму поради ограниченият капацитет на батериите потребно е да се одреди кои технологии ќе бидат користени за да се обезбеди ефикасност во поглед на енергетската потрошувачка. Токму ова е и целта на оваа магистерска работа, односно анализа на енергетската потрошувачка на неколку протоколи на апликациско ниво кај IoT и тоа: CoAP, MQTT и XMPP. Според добиените резултати од извршените пробни мерења се утврди дека протоколите CoAP и MQTT обезбедуваат најголеми заштеди на енергија, за разлика од XMPP кој троши повеќе енергија. За сите протоколи најмногу енергија се троши во состојба RX, додека најмалку се троши во состојба LPM.

За целите на оваа магистерска работа развиен е MQTT-клиент за оперативниот систем за мобилни уреди – Андроид. Тој е развиен со користење на библиотеката Eclipse Paho за Андроид. Овој клиент го користи објави/претплати моделот. Со него може да се објавуваат пораки и да се претплатуваме на одредени теми се со цел да добиваме пораки.

Клучни зборови: Интернет на нештата, CoAP, MQTT, XMPP, MQTT-клиент

Содржина

Листа на слики

Листа на табели

Листа на кратенки

IoT - Internet of Things

MIT - Massachusetts Institute of Technology

IP - Internet Protocol

IAB - Internet Architecture Board

RFC - Request for Comments

ITU - International Telecommunication Union

W3C - World Wide Web Consortium

RFID - Radio-Frequency Identification

NFC - Near Field Communication

IPv6 - Internet Protocol version 6

IEEE - Institute of Electrical and Electronics Engineers

IPv4 - Internet Protocol version 4

RAM - Ransom Access Memory

ROM - Read Only Memory

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

CoAP - Constrained Application Protocol

MQTT - Message Queuing Telemetry Transport

XMPP - Extensible Messaging and Presence Protocol

Wi-Fi - Wireless Fidelity

DTLS - Datagram Transport Layer Security

API - Application Programming Interface

ISO - International Organization for Standardization

M2M - Machine To Machine

HTTP - Hypertext Transfer Protocol

IETF - Internet Engineering Task Force

6LoWPAN - IPv6 over Low power Wireless Personal Area Networks

PHY - Physical layer

MAC - Medium Access Control

WLANs - Wireless Local Area Network

WPAN - Wireless Personal Area Network

WSN - Wireless sensor networks

TLS/SSL - Transport Layer Security/Secure Sockets Layer

QoS - Quality of Service

AMQP - Advanced Message Queuing Protocol

REST - Representational State Transfer

URI - Uniform Resource Identifier

OASIS - Organization for the Advancement of Structured Information Standards

XML - EXtensible Markup Language

JID - Jabber ID

SASL - Simple Authentication and Security Layer

DNS - Domain Name System

IQ - Info/Query

IM - Instant Messaging

GUI - Graphical User Interface

URL - Uniform Resource Locator

RPL - Routing Protocol for Low power and Lossy Networks

MUC - Multi-User Chat

LPM - Low Power Mode

CPU - Central Processing Unit

TX - Transmission

RX - Receive

mW - milliwatts

mJ - milliJoule

mA - milliAmpere

Вовед

Терминот „Интернет на нештата“ (Internet of Things – IoT) за прв пат бил употребен уште во далечната 1999 година од британскиот технолошки пионер Кевин Аштон (Kevin Ashton) [1]. Тој е соосновач на „Auto-ID Center“ на Институтот за технологии во Масачусетс (Massachusetts Institute of Technology - MIT). Со овој нов поим тој опишува систем со кој објектите во физичкиот свет може да се поврзат, да разменуваат податоци, да комуницираат помеѓу себе и да постигнат заеднички цели. Покрај глобалниот тренд на IoT, сеуште не постои универзална дефиниција на овој поим. Различни групи користат различни дефиниции за да го опишат или, пак, да го промовираат „IoT“, како и неговите најважни особини. Некои дефиниции даваат посебен осврт на концептот на Интернет или на Интернет протокол (IP), додека други не. Во овој дел ќе разгледаме неколку дефиниции за поимот Интернет на нештата. Одборот за интернет архитектура (Internet Architecture Board – IAB) го започнува „RFC 7452“[1], „Архитектонски размислувања кај вмрежените паметни објекти“ и ја дава следната дефиниција:

Поимот „Интернет на нештата“ означува тренд кај кој голем број вградени уреди користат комуникациски услуги кои се обезбедени од страна на интернет протоколите. Голем број на овие уреди, често наречени „паметни објекти“, не се директно управувани од луѓето, но постојат како компоненти во згради, возила или, пак, се расширени во животната средина.

Според Cisco[2], дефиницијата за Интернет на нештата е следната:

Интернетот на нештата поврзува паметни објекти на Интернет. Со него може да се овозможи размена на податоци која претходно не била достапна, и да ги донесе корисничките информации на посигурен начин.

Меѓународната телекомуникациска унија (ITU) во 2012 објавува „ITU-T“ препорака „Y.2060“, „Преглед на Интернетот на нештата“[3], и го дискутира овој концепт, но конкретно не го поврзува „IoT“ со Интернет. Дадена е следната дефиниција: *Интернет на нештата: Глобална инфраструктура за информатичкото општество, овозможува напредни услуги со меѓусебно*

[1] RFC 7452, “Architectural Considerations in Smart Object Networking” (March 2015), https://tools.ietf.org/html/rfc7452
[2] Internet Of Things (IoT), http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html
[3] “Overview of the Internet of Things.” ITU, June 15, 2012. http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=Y.2060

поврзување на (физички, виртуелни) нешта врз основа на постојни и развој на интероперабилни информатички и комуникациски технологии.

Според „CERP“-„IoT“ во нивниот документ „Internet of Things: Strategic Research Roadmap“[4]:

Интернет на нештата (IoT) е интегриран дел во иднината на Интернетот и може да се дефинира како динамична глобална мрежна инфраструктура со способности за сопствено конфигурирање врз основа на стандардизирани и интероперабилни протоколи за комуникација, при што физичките и виртуелните "нешта" имаат идентитети, физички атрибути, и виртуелни способности и користат интелигентни интерфејси, и беспрекорно се интегрирани во мрежата на информации.

Според „World Wide Web Consortium“ (W3C), дефиницијата за Интернет на нештата е следната:

„Интернетот на нештата се однесува на виртуелната претстава на широк спектар објекти на Интернет или нивната интеграција на Интернет и системите и услугите базирани на веб. Врз основа на интеракциските и комуникациски интерфејси како што се „RFID“, „NFC“, баркодовите и „2D“ кодовите, тие изложуваат информации, карактеристики и функционалности што може да се интегрираат во системите или во сервисите".[5]

Сите дефиниции опишуваат ситуации кај кои мрежната поврзаност и способноста на компјутерите се прошируваат кај објектите, уредите, сензорите како и кај секојдневните предмети кои вообичаено не се сметаат за „компјутери“. Ова им овозможува на уредите да генерираат, да разменуваат и да примаат податоци, често со минимални човечки интервенции. Различните дефиниции за „IoT“ не се согласуваат бидејќи тие потенцираат различни аспекти и случаи на употреба. Тие може да доведат до конфузија кога станува збор за прашања од областа на „IoT“, особено во дискусии помеѓу засегнати страни или индустриски сегменти. Слична конфузија во последните години е настаната со неутралноста на Интернет како со и облак технологиите, кај кои различните претстави на

[4] Internet of Things: Strategic Research Roadmap, By CERP-IoT, http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2009.pdf
[5] Internet of Things, By W3C, https://www.w3.org/WAI/RD/wiki/Internet_of_Things

Слика 1. Интернет на нештата

поимите понекогаш претставуваат пречка по одредени прашања што се однесуваат на нив. Главната причина за непостоењето на единствена дефиниција за Интернетот на нештата се различните перспективи од кои се гледа овој современ концепт.

Ештон го измислил терминот за да ја прикаже моќта на поврзувањето на „RFID“ (Radio – Frequency Identification), тагови што се користат од корпоративните синџири на Интернет со цел да ги бројат и да ги следат стоките без човечка интервенција. Денес, поимот за Интернет на нештата се користи за опишување сценарија во кои Интернет врската и компјутерската способност се прошируваат кај различни објекти, уреди, сензори, возила, апарати во домаќинството и слично. Со ова била додадена нова димензија во светот на информатичките и на комуникациските техологии. Целта на Интернетот на нештата е да овозможи нештата да бидат поврзани во кое било време, на кое било место, со што било и кој било (Слика 1). Ова подразбира доделување поими како што се конвергенција, содржина, колекции (репозиториуми), пресметување, комуникација и поврзување, во контекст на постоењето лесна меѓусебна интерконекција помеѓу луѓето и нештата или само помеѓу нештата [3].

Ова е нова револуција на Интернетот. Објектите имаат препознавачки способности и тие добиваат интелигенција со донесување или овозможување контекстно поврзани решенија. Тие може да пристапат до информации што се добиени од други нешта или тие може да бидат компоненти на комплексни

Табела 1. Уреди кај Интернетот на нештата

Број на жители во светот	6.3 милијарди	6.8 милијарди	7.2 милијарди	7.6 милијарди
Поврзани уреди	500 милиони	12.5 милијарди	25 милијарди	50 милијарди
Поврзани уреди по лице	0.08	1.84	3.47	6.58
Година	2003	2010	2015	2020

сервиси. Оваа трансформација е проследена со појавата на облак-технологиите, како и со транзицијата на Интернетот кон „IPv6“ со речиси неограничен адресен капацитет [2].

Се претпоставува дека бројот на уреди кај Интернетот на нештата ќе расте експоненцијално во следните години (Табела 1). Бројот на уреди во 2003 година бил 500 милиони. Во 2010 година бројот пораснал на 12.5 милијарди уреди. Во 2015 година овој број се зголемил за два пати и во оваа година имало околу 25 милијарди уреди. Се претпоставува дека бројот на уреди кај интернетот на нештата до 2020 година ќе ја достигне бројката од 50 милијарди уреди. Според претпоставките, ако го погледнеме бројот на жители во светот во периодот од 2015 до 2020, ќе видиме дека тој ќе се зголеми за 0.4 милијарди, додека бројот на уреди се очекува да се зголеми за 25 милијарди. Ако бројот на уреди по човек во 2015 година бил околу 3, се претпоставува дека бројот на уреди по човек до 2020 година ќе се зголеми двојно, односно околу 6 уреди по човек. Со тоа се потврдува важноста на овој нов концепт на иднината.

„Cisco“ претпоставува дека бројот на уреди до 2020 година ќе ја достигне бројката од 50 милијарди[6]. Според Juniper Research се претпоставува дека бројот на поврзани уреди кај „IoT“ ќе биде 38,5 милијарди до 2020 година, за разлика од 13.4 милијарди во 2015 година[7]. „Business Insider“ претпоставува дека бројот на уреди што се поврзани на Интернет до 2020 година ќе изнесува 34 милијарди[8]. Се гледа дека бројот на уреди се разликува во однос на

[6] Internet of Things (IoT), http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html
[7] 'Internet of Things' Connected Devices to Almost Triple to Over 38 Billion Units by 2020, http://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020
[8] BI Intelligence projects 34 billion devices will be connected by 2020, http://www.businessinsider.com/bi-intelligence-34-billion-connected-devices-2020-2015-11

претпоставките, но сигурно е тоа дека бројот на уреди ќе биде во милијарди и тој ќе продолжи да се зголемува експоненцијално во иднина.

Кај уредите со ограничени перформанси од огромно значење е потрошувачката на енергија. Потрошувачката на енергија кај безжичните сензорски мрежи е во тесна врска со комуникацијата. Модулите кои комуницираат помеѓу себе обично се најголемите потрошувачи на енергија. Во основа тие се уреди што имаат ограничени перформанси и мал капацитет на батеријата. Затоа е многу важно тие да обезбедуваат заштеди во поглед на енергетската потрошувачка.

Малата потрошувачка на енергија е многу важна кај различни типови безжични системи. За системите кои се напојуваат од батерии, потрошувачката на енергија го одредува и нивниот животен век. Бидејќи батериите често не може да бидат заменети, ниската потрошувачка на енергија овозможува и помали оперативни трошоци. Кај системи кои се напојуваат со помош на обновливи извори на енергија, како соларните ќелии, се добиваат мали количества енергија. Дури и кај системите што користат постојано напојување потрошувачката на енергија исто така е многу важна. Кај паметните мрежи на електрична енергија основна цел е обезбедувањето соодветни заштеди. Потрошувачката на енергија на уредот често ги одредува неговата цена и капацитетот на неговиот конвертор. Така овие уреди со мала потрошувачка на енергија најчесто се помали системи и чинат многу помалку во споредба со другите системи.

Бројот на уреди кај безжичните сензорски мрежи од ден на ден постојано се зголемува. Тие имаат важен дел и постојат како посебни уреди или пак се интегрирани во други системи кои ги користат нивните функции. Се користат при развивањето системи за електрични мерења, системи за управување со енергија, контрола на системите за греење, системи за одредување температура, светлина и други податоци од животната средина. Тие се сметаат како основни уреди за идните паметни мрежи и паметни градови. Голем дел од овие уреди најчесто се напојуваат со помош на батерија, при што значајна улога има потрошувачката на енергија.

Обезбедувањето енергетско-ефикасни механизми е основна тема и кај голем број апликации како: апликации за мобилни телефони, серверски системи, податочни центри. Интернетот на нештата како иднина во компјутерските науки бара обезбедување механизми за енергетска ефикасност кај уредите. На ова поле денес постојано се развиваат нови оперативни системи, апликации, протоколи и потрошувачката на енергија игра важна улога кај нив.

1. Комуникациски модели кај Интернетот на нештата

Уредите кај Интернетот на нештата се поврзани помеѓу себе и комуницираат. Од оперативна гледна точка важно е да се размислува како ќе бидат поврзани тие и кои комуникациски модели ќе бидат применети. Во март 2015 година „Internet Architecture Board“ (IAB) објавил водечки документ за архитектурата на поврзувањето на паметните уреди (RFC 7452). Во овој документ се претставени четири општи комуникациски модели што се користат од уредите кај „IoT“. Во следниот дел поединечно ќе бидат објаснети сите модели како и нивните карактеристики. Дополнително ќе биде разгледан и референтниот модел.

1.1 Комуникациски модел „уред-до-уред“

Комуникацискиот модел „Уред-до-уред“ претставува модел со кој два или повеќе уреди се директно поврзани и комуницираат еден со друг без притоа да користат апликативен сервер како посредник [1]. Овие уреди комуницираат со помош на различни типови мрежи, вклучувајќи „IP“-мрежи или, пак, Интернет. Често овие уреди користат протоколи како „Bluetooth“, „Z-Wave“, „ZigBee“ или други типови безжични мрежи за да остварат директна комуникација помеѓу уредите. На Слика 2 е прикажан пример со директна комуникација помеѓу два уреда и тоа: светилка и прекинувач за светло. Во овој случај тие се изработени од различни производители означени како Производител 1 и Производител 2 иако може да потекнуваат и од исти производители. За да може уредите од различни производители да комуницираат потребно е да се договорат за свитата (стекот) на протоколи која ќе ја користат. Потребно е да се утврди кој физички слој треба да се користи и дали да се користат нискомоќни радиотехнологии како: „Bluetooth“ или „IEEE 802.15.4“. На мрежно ниво треба да се договори дали ќе се користи „IPv6“ или, пак, протоколот „IPv4“ и да се види

Слика 2. Комуникациски модел „уред-до-уред“

кои „IP“ адресни конфигурациски механизми се интегрирани кај уредите. Треба да се утврди кои уреди се ограничени, и кои се нивните ограничувања (RAM, ROM, процесирачка моќ, батерија). Токму енергетската потрошувачка кај уредите ќе биде разгледана овде. Најчесто уредите имат ограничен капацитет на батеријата, па затоа е важно колкава е потрошувачката на енергија. За таа цел се користат одредени механизми за одредување на потрошувачката што ќе бидат разгледани во оваа магистерска работа. Треба да се утврди дали е потребен механизам за откривање сервиси кој ќе им помогне на корисниците, на пример, да ги откријат светилките во нивната соба или канцеларија. На транспортно ниво исто треба да се определи кој протокол (TCP или UDP) ќе се користи за пренесување на податоците кои се читаат од сензорите или од командите. На апликациско ниво кај „IoT“ се користат повеќе протоколи (на пример: „Constrained Application Protocol“ - „CoAP“, „Message Queue Telemetry Transport“ - „MQTT“, „Extensible Message and Presence Protocol“ - „XMPP“). Токму овие протоколи се тема на истражувањата во оваа магистерска работа и тие ќе бидат подетално разгледани во поглавје 2. Потребно е да се утврди и кој податочен модел ќе се користи за кодирање на информациите. Безбедноста и приватноста се исто така важни. Воведувањето безбедносни механизми знае да доведе до зголемување на потрошувачката на енергија. Потребно е да се утврдат безбедносните закани и кои безбедносни сервиси треба да бидат обезбедени за да се справиме со нив. Треба да се идентификуваат и слоевите од свитата на протоколи за кои треба да бидат обезбедени безбедносните механизми. Стандардизирањето комплетно решение за постигнување целосно ниво на интероперабилност помеѓу два уреди од различни производители бара време, меѓутоа придобивките од него се големи како за корисниците, така и за производителите [4].

За комуникација помеѓу уредите се користат безжични мрежи за остварување директна комуникација. Овие мрежи користат одредени протоколи за комуникација и размена на пораки поради остварување одредена цел. Овој комуникациски модел најчесто се користи кај апликации како домашни системи за автоматизација. Тие користат мали пакети со информации за комуникација помеѓу уредите и бараат релативно ниски податочни стапки. Уреди за кои е погоден овој комуникациски модел се следните: светилки, термостати,

прекинувачи за светло, брави на врати и слично. Сите тие праќаат мали количества податоци едни на други [1].

1.2 Комуникациски модел „уред-до-облак“

Кај комуникацискиот модел „Уред-до-облак“ уредите се поврзуваат директно на интернет сервис кој се наоѓа во облакот како апликациски сервис-провајдер (обезбедувач на услуги) за размена на податоци и контрола на пораките кои се пренесуваат. Овој пристап често ги користи предностите на постојните комуникациски механизми како „Ethernet“ или „Wi-Fi“ за да воспостави врска помеѓу уредот и „IP“-мрежата, која на крајот се поврзува со сервис-провајдерот во облакот [1] [4]. На Слика 3 е прикажан пример за комуникациски модел „уред-до-облак“. Дадени се уреди кои имаат два типа сензори и тоа: уред со температурен сензор и уред со сензор за јаглерод моноксид. Податоците кои се добиваат од сензорите се праќаат до апликацискиот сервис-провајдер, кои понатаму може да се обработуваат и да се преземаат. Иако овој модел овозможува користење на „IP“-базирана комуникација „крај-до-крај“, сè уште може да доведе до создавање силоси[9] [4]. За да се спречи ова, сервис-провајдерите може да им дозволат на продавачите на уреди да се поврзат со нивната серверска инфраструктура. Во овие случаи се користат интерфејс-протоколи кои се користат за комуникација со серверската инфраструктура и се достапни различни стандарди како што се: „CoAP“, „Datagram Transport Layer Security“ (DTLS), „UDP“, „IP“ (Слика 3). Загрижувачки за крајните корисници е ако

Слика 3. Комуникациски модел „уред-до-облак“

[9] Податочен силос е складиште на основни податоци кои се под контрола на еден оддел и се изолирани од остатокот на организацијата, http://searchcloudapplications.techtarget.com/definition/data-silo.

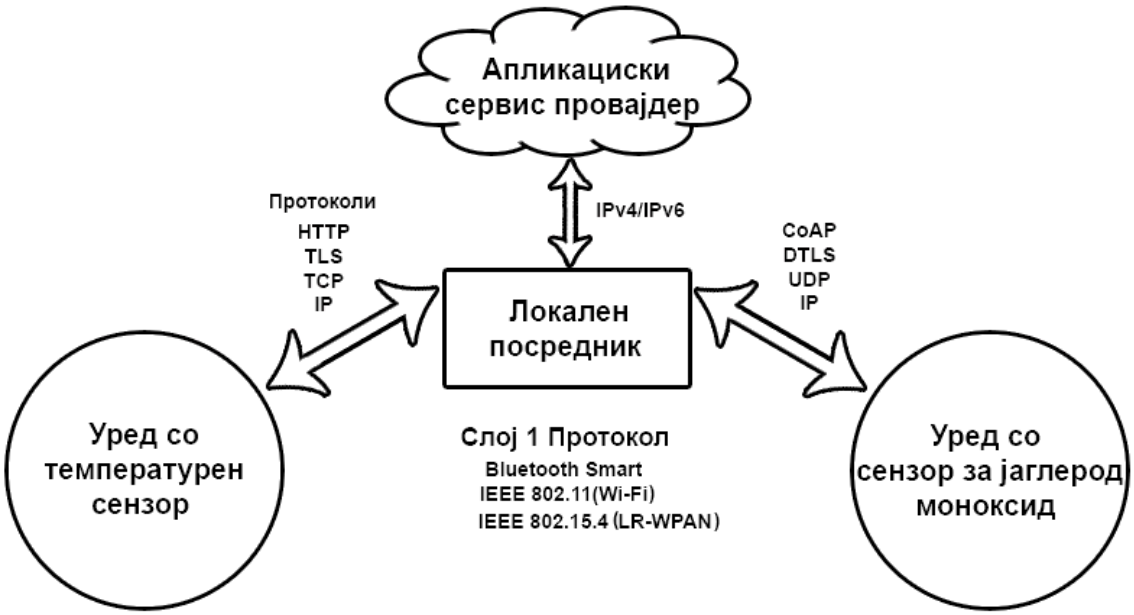
дојде до промена на бизнис моделот на „IoT“-уредот или сервисот, што може да доведе до неупотребливост на хардверот. Затоа е потребно компаниите да го ослободуваат изворниот код за уредот и да дозволат други оперативни системи за Интернетот на нештата заедно со апликативен софтвер да може да се инсталираат на уредот.

Овој комуникациски модел го користат повеќе уреди како: „Nest Labs Learning Thermostat“[10] и „Samsung SmartTV“[11] [1]. Паметниот термостат „Nest“ пренесува податоци до базата на податоци што се наоѓа во облакот. Податоците се користат за анализа на енергетската потрошувачка во домовите. Ова поврзување со облакот му овозможува на корисникот да добие далечински пристап до термостатот преку паметен телефон или преку веб-интерфејс, а поддржува и софтверски ажурирања на термостатот. Паметниот телевизор на „Samsung“ користи интернет врска за пренос на кориснички податоци до „Samsung“ за анализа и да овозможи интерактивни особини за препознавање на гласот на телевизорот. Со ова се зголемуваат карактеристиките на уредот, во однос на основните карактеристики кои веќе ги поседува. Проблеми во однос на интероперабилноста може да настанат ако се интегрираат уреди од различни производители. Најчесто, уредите и сервисите во облакот се од исти производители. Ако се користат комерцијални податочни протоколи помеѓу уредот и сервисот, во тој случај доаѓа до ограничување и спречување на употребата на алтернативни сервисни провајдери. Ова најчесто се нарекува „заклучување од производителот“. Позитивна страна е тоа што корисниците може да имаат доверба дека уредите наменети за одредена платформа може да бидат интегрирани.

1.3 Комуникациски модел „уред-до-посредник“

Често за комуникација помеѓу уредите се користат помалку распространети радио технологии (како IEEE 802.15.4) или, пак, мора да бидат обезбедени специјални функционалности на апликациско ниво (локална автентикација и авторизација). Некогаш е потребно да се обезбеди интероперабилност со посредство или да се користат уреди кои не се базирани на „IP“. Во овие случаи е потребно да се користи некој тип на посредник во

[10] Meet the Nest Thermostat, https://nest.com/thermostat/meet-nest-thermostat/
[11] Samsung Smart TV - TV Has Never Been This Smart, http://www.samsung.com/us/experience/smart-tv/



Слика 4. Комуникациски модел „Уред-до-посредник“

комуникацискатa архитектура кој претставува еден вид мост помеѓу различните технологии и извршува други мрежни и безбедносни функционалности (Слика 4) [4]. Овие посредници најчесто се обезбедени од производителот на „IoT“-уредот. Тоа е така поради користењето комерцијални протоколи, поради намалувањето на зависноста од други производители или поради избегнувањето потенцијални интероперабилни проблеми. Во многу случаи, локалниот посредник е паметен телефон на кој работи апликација за комуникација со уред и пренесува податоци до сервисот во облакот. Овој модел е застапен и кај личните фитнес-следачи. Овие типови уреди немаат способност да се поврзат директно со сервисот во облакот, па тие често користат апликација за паметен телефон која е еден вид посредник за да се поврзе фитнес-уредот со облакот [1]. Овој комуникациски модел се јавува и кај апликациите за автоматизација во домовите. Неговата примена е зачестена со појавата на централни (hub) уреди. Тоа се уреди кои се однесуваат како посредници помеѓу индивидуалните „IoT“-уреди и сервисите во облакот. Со ова се надминува и интероперабилниот јаз помеѓу уредите. Пример за ова е „SmartThings“[12] хаб-от, кој е самостоен посреднички уред што има „Z-Wave“ и „Zigbee“ примопредаватели кои се инсталирани за да комуницираат со двата вида уреди. Хаб-от се поврзува потоа со сервисот „SmartThings“ во облакот, и му дозволува на корисникот да добие пристап до уредите користејќи

[12] Smart Home. Intelligent Living, SmartThings, https://www.smartthings.com/

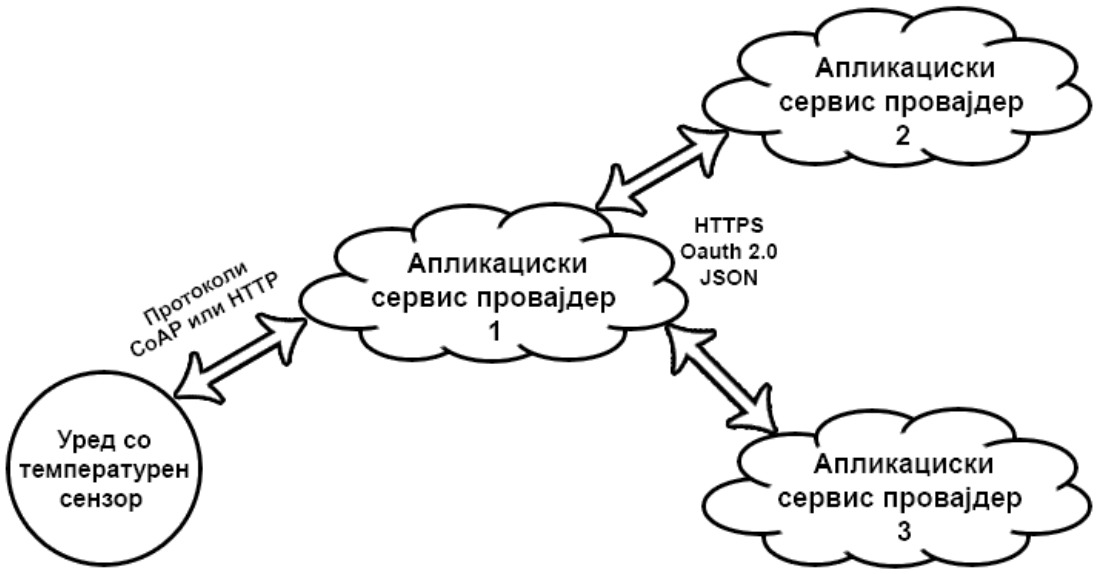
притоа апликација за паметен телефон и интернет врска. Во документот „IAB RFC 7452“ за овој модел се споменува следното [4]:

Во иднина се очекува дека ќе бидат развиени повеќе посредници за да се намалат трошоците, како и комплексноста на инфраструктурата за крајните корисници, претпријатија и индустриски средини. Такви генерички посредници е поверојатно да постојат ако дизајните на „IoT“-уредите користат генерички интернет протоколи кои не бараат посредници на апликациско ниво кои претвораат еден протокол на апликациско ниво во друг.[13]

Овој комуникациски модел е исто така корисен во случаите кога треба да се обезбеди безбедност на апликациите како и на податоците што се пренесуваат од уредите до сервисите кои се наоѓаат во облакот.

1.4 Модел за споделување податоци во „Back-end“

Моделот за споделување податоци во „back-end“ претставува комуникациска архитектура која им овозможува на корисниците да ги извезуваат и да ги анализираат паметните објекти од сервис кој се наоѓа во облакот во комбинација со податоци кои доаѓаат од други извори (Слика 5). Со оваа архитектура корисниците може да добијат пристап до трети страни (други



Слика 5. Модел за споделување податоци во „back-end“

[13] Architectural Considerations in Smart Object Networking, Internet Architecture Board (IAB), https://tools.ietf.org/html/rfc7452, p.6

сервис-провајдери) и да ги прикачат податоците кои се добиваат од сензорите. Овој пристап во основа е продолжување на комуникацискиот модел „уред-до-облак“, каде што „IoT“-уредите може да испраќаат податоци само до еден апликациски сервис-провајдер. Овој модел овозможува податоците кои се собрани од еден „IoT“ податочен проток да бидат складирани и анализирани.

Корисниците кои се задолжени за одржување згради и канцеларии се заинтересирани за анализа на потрошувачката на енергија и услугите кои се добиени од сите „IoT“-сензори и системи во просториите. Кај моделот „уред-до-облак“, податоците кои се добиваат од сензорите се праќаат до самостоен податочен силос. Една ефективна архитектура за споделување податоци во „back-end“ ќе им овозможи на корисниците да можат лесно да пристапат и да ги анализираат податоците во облакот кои се добиени од сите уреди во зградата. Со овој модел се олеснува и преносливоста на податоците ако се јави потреба за тоа. Така корисниците може да ги пренесат податоците кога тие се префрлаат помеѓу „IoT“-сервиси. Со тоа се надминуваат пречките кои се создаваат со формирањето на податочни силоси. За постигнување интероперабилност на податоците кои се добиени од паметните уреди и се одржуваат во облакот, потребни се федерални сервиси во облакот или апликациски програмирачки интерфејси (APIs). Федералните сервиси се сервиси кои комбинираат ресурси од одделни сервис-провајдери во облакот [1].

Овој архитектурен модел е пристап со кој може да се постигне интероперабилност помеѓу „back-end“-системите. Стандардните протоколи може да помогнат, но не се доволни за да се отстранат податочните силоси бидејќи за тоа се потребни заеднички информациски модели за производителите на уреди. Овој комуникациски модел е ефективен како и основните системски дизајни кај „IoT“ и тој не може целосно да ги надмине пречките кои настануваат со таканаречените „затворени системски дизајни“.

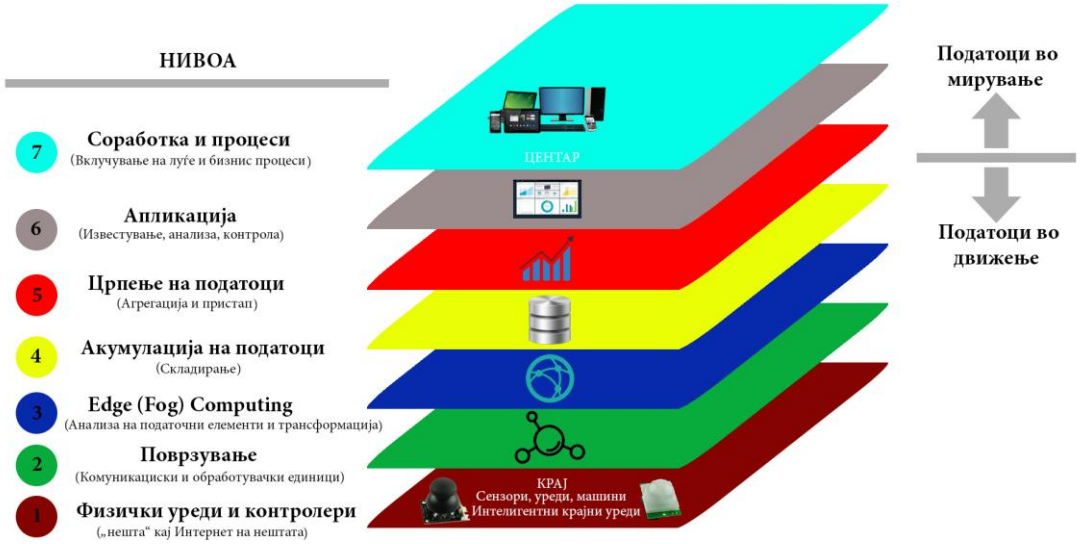
1.5 Референтен модел кај Интернетот на нештата

Според „Cisco“ потребен е нов референтен модел кај „IoT“ сè со цел да се овозможи брза, лесна и непречена комуникација помеѓу уредите [5]. Постојните мрежни, пресметувачки, апликациски архитектури, како и архитектурите за управување со податоци бараат различни модели за комуникација и обработка. Денес не постои стандарден начин за опишување и разбирање на моделите кај

„IoT“. Како резултат на тоа, нејасни се линиите помеѓу „IoT“ уредите и уредите кои не влегуваат во оваа група заедно со системите. Не мора секоја апликација или мрежа да биде поврзана со Интернетот на нештата. Често податоците се генерираат и се под контрола на машини (нешта) или опрема и се праќаат преку мрежа. Во тој случај веројатно е дека станува збор за „IoT“ систем. Меѓутоа, може да има и исклучоци така што оваа генерализација да не важи за некои типови уреди.

Основната цел на референтниот модел кој го предлага „Cisco“ е да се обезбедат јасни дефиниции и опис кои може да се применат точно на елементите и функциите кај „IoT“ системите и апликациите. Овој модел овозможува разгледување на архитектурата на комплексните системи на делови, со што секој дел може да се разбере подобро. Тој овозможува дополнителни информации со кои може да се идентификуваат различните нивоа кај „IoT“ и да се обезбеди заедничка терминологија. Знаеме дека често уредите се произведуваат од различни производители така што обезбедувањето соодветни заеднички термини за најважните концепти во архитектурата на системите е многу важно. Со тоа се овозможува и стандардизација на уредите, што е почетна точка за производителите да може да создаваат уреди кои ќе може да работат и да комуницираат еден со друг. Со примената на овој модел се врши и идентификување на специфични типови обработка кои се оптимизираат во различни делови од системот. Тој овозможува организираност што го прави „IoT“ реален и пристапен. Реален во смисла на изводливост и примена во реалниот живот. Пристапен се мисли на пристапност на уредите кои ќе имаат можност да се поврзуваат еден со друг и да разменуваат корисни информации.

Податоците кај „IoT“ системите се генерираат од различни уреди, се обработуваат на различни начини и се пренесуваат на различни локации. Тие може да бидат користени и од апликации. Со референтниот модел се дефинираат нивоа кои ги опишуваат системите и процесите кај „IoT“. Постојат вкупно седум нивоа. Секое ниво е дефинирано според соодветна терминологија која понатаму може да се стандардизира. Со овој модел не се ограничува опсегот или, пак, локалноста на неговите компоненти. Тоа значи дека елементите може да бидат локални и да се наоѓаат на едно место или тие може



Слика 6. Нивоа на референтниот модел кај „IoT“

да бидат дистрибуирани. Моделот опишува како треба да се управува со задачите на секое ниво за да се одржува едноставноста на системот и да се овозможи приспособливост и поддршка. Со него се објаснуваат и сите функции кои се потребни на еден систем кај „IoT“ да биде целосен и да може непречено да функционира.

На Слика 6 се прикажани сите нивоа од кои се состои овој модел. Податоците кои протекуваат низ нивоата се движат во двете насоки. Притоа, контролата на текот на информациите се движи од последното (Ниво 7), па до првото (Ниво 1). Кај шемата на следење текот на податоците е обратен, односно од првото (Ниво 1), па до последното (Ниво 7). Овие текови кои се одвиваат во двете насоки се застапени кај поголем дел од системите кај Интернетот на нештата.

1.5.1 Ниво 1: Физички уреди и контролери (управувачи)

Првото ниво кај референтниот модел кај „IoT“ го составуваат физичките уреди и контролерите кои може да контролираат повеќе уреди во зависност од нивните карактеристики. Во основа, тие го претставуваат и зборот „нешта“ кој се јавува како дел од фразата „Интернет на нештата“. Тие вклучуваат голем број уреди што имаат способност да примаат и да праќаат податоци. Како што споменавме и на почетокот, бројот на уреди во моментот е голем и изнесува милијарди, а и во иднина се очекува овој број да се зголемува експоненцијално. Тој ќе стане речиси неограничен со постојаното додавање нова опрема кај „IoT“. Сите уреди се различни и имаат посебни карактеристики според кои се

разликуваат едни од други. Тие се разликуваат според формата, големината, локацијата или потеклото. Некои уреди се многу минијатурни, со големина на силиконски чип, додека други се големи, комплексни системи кои извршуваат најразлични функции. „IoT“ мора да ги поддржува сите уреди без разлика на нивните карактеристики. Голем број производители на опрема во иднина ќе произведуваат „IoT“ уреди. Овие уреди треба да бидат способни да ги конвертираат податоците од аналогна во дигитална форма, да генерираат податоци, да може да се доставуваат команди до нив и да може да се контролираат преку Интернет. Во иднина е важно за тие уреди да се обезбеди соодветна компатибилност и поддршка, сè со цел поедноставно вмрежување на уредите и создавање еден глобален екосистем на Интернетот на нештата.

1.5.2 Ниво 2: Поврзување

Поврзувањето и комуникациите се наоѓаат на Ниво 2 кај референтниот модел на „IoT“. На ова ниво најважно е да се обезбеди доверливо и навремено пренесување информации. Притоа, постојат повеќе типови пренос како што се: помеѓу уредите (Ниво 1) и мрежата, преку мрежите и помеѓу мрежата (Ниво 2) и обработката на информации што се случува на Ниво 3. Често традиционалните комуникациски мрежи имаат повеќе функции, како што било потврдено од Меѓународната организација за стандардизација (ISO) со 7-слојниот рефрентен модел. Комплетниот систем кај „IoT“ содржи голем број нивоа во прилог на комуникациските мрежи. Според овој модел, комуникацијата и обработката ќе се извршуваат со користење на постојните мрежи. За остварување на неговите цели, не е потребно креирање други мрежи кои ќе се разликуваат од постојните. Постојат уреди што немаат поддршка за „IP“. За нив ќе бидат користени посредници со кои ќе може да се оствари поврзување и комуникација. Други уреди, пак, бараат соодветни контролери со кои ќе може да ја остварат функцијата за комуникација. Со текот на времето се очекува да се обезбеди соодветна стандардизација. Како што бројот на уредите на Ниво 1 се зголемува, така и начините за комуникација со Ниво 2 ќе се менуваат. Уредите на Ниво 1 комуницираат преку системот на „IoT“ со интеракција со опремата за поврзување на Ниво 2. Притоа, поврзувањето вклучува: сигурно доставување преку мрежата (доставување на податоците што се пренесуваат), имплементација на различни протоколи со кои ќе се овозможи стандардизација за уредите кај Интернетот на

нештата (стандардизирана свита на протоколи), прекинување и рутирање, преведување помеѓу протоколите, безбедност на мрежно ниво и анализа на вмрежувањето.

1.5.3 Ниво 3: Edge (Fog) computing

Функциите на Ниво 3 (Edge computing[14]) се претворање на мрежните податочни протоци во информации кои потоа може да се складираат и да се обработуваат на повисокото ниво за обработка, односно Ниво 4 (акумулирање податоци). Активностите на ова ниво се фокусираат на анализа на големиот број податоци и на нивна трансформација. Овие податоци може да доаѓаат од сензорен уред кој може да генерира примероци од податоци повеќепати во секунда, 24 часа на ден, 365 дена во годината. Референтниот модел се води од принципот кој го имаат повеќето интелигентни системи што иницираат обработка на информациите колку што е можно поблиску до работ на мрежата. Овој принцип често се нарекува „Fog computing[15]“. Сето ова се случува на Ниво 3. Обработката на податоците на ова ниво се врши „пакет-по-пакет“. Обработката е ограничена бидејќи ние сме запознаени само со единиците на податоците и не постојат „сесии“ или пак „пренесувања“. Ова ниво може да опфати повеќе примери како:

- Оценување на податоците по критериуми за тоа дали тие треба да се обработуваат на повисоките нивоа;
- Форматирање на податоците за соодветна обработка на повисоките нивоа;
- Управување со шифрирани податоци кои треба да се дешифрираат
- Намалување и сумирање на податоците за да се намали влијанието на податоците и сообраќајот на мрежата на повисоките нивоа;
- Утврдување дали податоците претставуваат некој праг или, пак, предупредување што може да доведе до пренасочување кон други дестинации.

[14] „Edge computing“ овозможува анализа и производство на знаење од изворот на податоци, https://en.wikipedia.org/wiki/Edge_computing

[15] „Fog computing“, дистрибуирана компјутерска архитектура со која некои апликациски сервиси се извршуваат на работ на мрежата кај паметните уреди и некои апликациски сервиси се извршуваат во далечински центар со податоци. Целта на оваа архитектура е да се подобри ефикасноста и да се намали количеството податоци што треба да биде пренесено до облакот (cloud) за обработка на податоците, анализа и складирање, http://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging

Во ова ниво влегуваат податочни пакети, а излегуваат информации кои се разбирливи за погорните нивоа.

1.5.4 Ниво 4: Складирање податоци

Мрежните системи се изградени со цел да пренесуваат податоци до одредени дестинации. Со тоа податоците се во движење. Пред да се дојде до Ниво 4, податоците се движат низ мрежата според степен и организација кои се одредени од уредите што ги генерираат податоците. Кога податоците доаѓаат на ова ниво, тие веќе не се „во движење“, туку се складираат и се наоѓаат во состојба на мирување. Ова ниво одредува:

- Дали податоците им се потребни на погорните нивоа бидејќи ова ниво е конфигурирано да може да ги опслужува нив;
- Дали податоците треба да се зачуваат постојано, односно дали треба да се чуваат во меморијата постојано или тие се само за краткотрајна употреба;
- Типот на складот кој е потребен, односно дали е потребен едноставен систем на документи, систем што содржи голем број податоци или релациона база на податоци;
- Дали податоците се правилно организирани за соодветниот систем за складирање;
- Дали податоците може да се рекомбинираат, пресметуваат и да се складираат со претходно зачувани податоци, од кои некои може да не доаѓаат од извори на „IoT“.

Ова ниво ги зачувува податоците кои потоа може да се користат од апликациите и тие ќе може да пристапуваат до нив, да ги читаат и да ги менуваат. Тоа значи дека податоците во складот може да бидат зачувани во кое било време. Апликацијата исто така може да пристапува до податоците во кое било време веднаш по нивното складирање. Ниво 4 ги претвора податоците кои се базирани на настани до обработка која е базирана на барања. Ова е основната разлика помеѓу вмрежувањето во реално време и апликациите кои не пристапуваат во реално време. Податоците овде исто така може да се намалат со филтрирање и селективно складирање. Основната цел на ова ниво е да ги претвори мрежните податоци во податоци кои ќе бидат користени од апликациите.

1.5.5 Ниво 5: Црпење податоци

Функциите за црпење податоци на Ниво 5 од референтниот модел се фокусираат на рендерирање (враќање) на податоци и нивно складирање на начин кој овозможува развивање поедноставни апликации со подобрени перформанси. Често податоците се генерираат, односно потекнуваат од различни извори и не може да се складираат на исти места. За тоа постојат и други причини како:

- Големиот број податоци кои треба да се складираат;
- Користење голема процесирачка моќ при складирање на податоците во базите на податоци, така што прибирањето податоци треба да биде одвоено од процесот на генерирање податоци;
- Уредите може да бидат оддалечени и обработката е оптимизирана на локално ниво;
- Складот за податоци може да биде голем податочен систем во кој ќе се зачувуваат проточни податоци или може да биде релациона база на податоци за складирање настани;
- Можеби е потребен различен тип на обработка на податоците;

На ова ниво е потребно е да се усогласат форматите на податоците кои доаѓаат од различни извори и да се обезбеди постојана семантика. Податоците е потребно да се консолидираат на едно место или да се обезбеди пристап до повеќе складови на податоци со податочна виртуализација. За нив исто така треба да се обезбеди соодветна заштита со обезбедување ефикасни методи за автентикација и авторизација. За да може апликациите брзу да пристапуваат до нив, потребни се методи за нормализирање и индексирање.

1.5.6 Ниво 6: Апликација

Ниво 6 е апликациско ниво. На ова ниво се одвива толкувањето на информациите. Ова ниво соработува со Ниво 5, како и со податоците што се складирани. Ова ниво стриктно не ги дефинира апликациите кои често се разликуваат по природата на податоците, по деловните потреби и по пазарот за кој се наменети. На пример, некои апликации се наменети за следење податоци кај уредите, други се наменети за контрола на уреди, трети комбинираат податоци кои доаѓаат од уреди или кои не потекнуваат од нив. Апликациите за следење и контрола претставуваат различни типови апликациски модели,

програмирачки шеми и софтвер. Комплексноста на апликациите исто така варира. Примери за апликации се: бизнис-апликации и специјализирани индустриски решенија, мобилни апликации, аналитички апликации и апликации за контрола и управување на системите. Сите тие имаат различни карактеристики како во начинот на обработка на податоците, така и во начинот на нивното претставување. Во зависност од потребите треба е да се оцени кој тип апликација може најмногу да придонесе за соодветната цел за која треба да биде наменета.

Ако претходните нивоа од 1 до 5 се соодветно предвидени и ако сите активности се одвиваат непречено, тогаш работата на Ниво 6 ќе биде значително намалена. Ако апликациското ниво е соодветно дизајнирано, тогаш корисниците ќе може да ги извршуваат нивните работи побргу.

1.5.7 Ниво 7: Соработка и процеси

Системите кај „IoT“, како и информациите кои ги креираат тие, имаат многу мала важност ако не е преземена некоја акција што вклучува луѓе или, пак, процеси. Апликациите ја извршуваат деловната логика за да ги поттикнат луѓето. Луѓето ги користат апликациите, како и податоците за нивните специфични потреби. Често, повеќе луѓе користат иста апликација за голем број различни цели. Нивната цел не е примената, туку поттикнувањето на луѓето да ја извршуваат својата работа подобро. Апликациите ги обезбедуваат вистинските податоци во точно време, така што оние што ги употребуваат ќе имаат можност да ги употребат на вистински начин и да ги остварат своите цели.

Извршувањето на акциите често бара вклучување на повеќе луѓе. Луѓето мора да бидат способни да комуницираат и да соработуваат помеѓу себе. Често за тоа се користи традиционалниот Интернет, со што „IoT“ се прави корисен. За остварување на соработката и комуникацијата, потребни се повеќе чекори кои треба да ги поминат повеќето апликации.

2. Протоколи кај Интернетот на нештата

Голем број уреди кај Интернетот на нештата се со ограничена природа. Тие се карактеризираат со мал капацитет на меморија, ниска моќ, мали барања за пропусен опсег, голема загуба на пакети [6]. Еден од најголемите предизвици кај „IoT“ е обезбедувањето стандардизирана свита на протоколи која ќе се користи кај овој тип уреди. Поради ограниченоста на уредите во поглед на нивните карактеристики, тешко е да се примени комплетната свита на TCP/IP протоколи, како и постојните веб-технологии. Постојат голем број протоколи во полето на „IoT“, „Machine-to-machine“[16] (M2M), како и кај системите за автоматизација на домовите. Пример за вакви протоколи се: „ZigBee“ и „Z-Wave“. „ZigBee“ е отворен, глобален стандард кој се користи за адресирање уникатни потреби за нискомоќните и нискобуџетни безжични мрежи кај „M2M“ [7]. Намената на протоколот „Z-Wave“ е слична, со тоа што негова главна цел е остварување комуникација со контролни пораки на доверлив начин, од контролната единица до еден или до повеќе јазли во мрежата [8]. Како и да е, овие протоколи се поддржани главно од сојузот на производители на опрема. Тие не се стандардизирани протоколи како што се: „TCP“, „IP“ или „HTTP“. Комуникацискиот стек треба да се карактеризира со следните особини:

- Комуникацискиот стек да обезбедува заштеди на енергија [9]. Поголемиот дел уреди кај интернетот на нештата не се во можност да користат електрична енергија преку напојување. Причината за тоа најчесто е нивната локација, непристапноста, малата потрошувачка и слично. Тие најчесто користат електрична енергија од батерии. Многу е важно да се одредат протоколи, односно комуникациски стек кој ќе овозможи заштеди на енергија. Токму ова е темата на овој магистерски труд. Понатаму ќе направиме анализа на протоколи и како влијаат тие врз потрошувачката. Многу е важно да се одредат протоколи кои одговараат на моделот што се бара како решение за комуникација помеѓу уредите, а воедно тој модел да обезбеди и соодветни заштеди на енергија.

  Ние ги полниме нашите телефони речиси секој ден, но би било многу непрактично ако тоа треба да се прави и со уредите кај Интернетот на

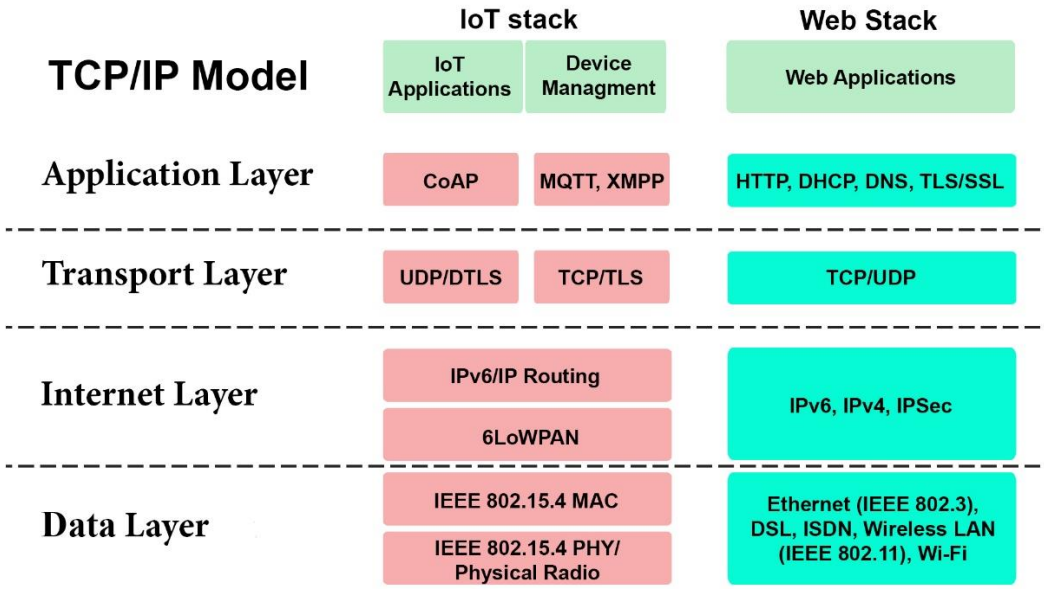
[16] „Machine to Machine“ (M2M) е поим со кој се опишува секој тип на технологија што овозможува вмрежените уреди да разменуваат податоци и да извршуваат акции без да бидат асистирани притоа од човечки фактор, http://internetofthingsagenda.techtarget.com/definition/machine-to-machine-M2M

нештата. Како што е претходно споменато, бројот на уреди драстично ќе се зголемува секоја година достигнувајќи милијарди уреди. Секој од нас тогаш би поседувал повеќе вакви уреди. Би било многу непрактично ако е потребно да ги полниме нив секој ден.

- Доверлив комуникациски стек. Иако Интернетот е најдобриот медиум за пренесување информации, сепак протоколите вклучуваат откривање грешки, повторни трансмисии, како и контрола на тек [9]. Овие техники се применуваат на различни слоеви и тоа истовремено, што доведува до доверливо „крај-до-крај“ искуство. За уредите кај „IoT“ да функционираат непречено потребно е да се обезбеди истото ниво на доверливост како и кај Интернетот. Со тоа уредите ќе бидат сигурни дека информациите кои ги споделуваат или, пак, командите ќе бидат доделени со најголема сигурност без никакви влијанија и пренасочувања.
- Интернет-овозможен комуникациски стек. Со овој пристап треба да се обезбеди олеснета комуникација од уредите и до уредите. Бидејќи комуникацијата е двонасочна, во тој случај се појавуваат различни однесувања како во пристапот, така и во начинот на примање или, пак, на испраќање на податоците. Со ова машините ќе бидат способни да „зборуваат“ со други типови машини и сето тоа е олеснето со користење на само еден јазик. Од огромно значење за „IoT“ е тоа што тој е овозможен со „IP“.

Денес постои еден збир од протоколи кај Интернетот на нештата кои сè почесто се употребуваат. Организации како „IETF“, „IEEE“, „W3C“ имаат стандардизирано протоколи како што се „6LoWPAN“ или „CoAP“. Во иднина овие протоколи ќе ја прошират нивната употреба и ќе придонесат за зголемување на поврзаните уреди. Со тоа ќе се стандардизираат како веб-стандардите кои се користат кај денешниот веб.

На Слика 7 се прикажани стандардизирани протоколи кои се користат кај „IoT“. Дадени се протоколите на секое ниво кај моделот „TCP/IP“ и тоа: мрежно, интернет, транспортно и апликациско ниво. За споредба, на истата слика се дадени и протоколите на истите нивоа кои се користат кај веб, сè со цел за подобро да се сфати улогата на протоколите кај „IoT“. На мрежно/линк ниво се користи протоколот „IEEE 802.15.4“. На интернет нивото се користи „IPv6/IP



Слика 7. Стандардизирани протоколи кај „IoT“

Routing“, но најчеста е употребата на протоколот што претставува приспособена верзија на „IPv6“ за Интернетот на нештата позната како „6LoWPAN“. На транспортниот слој се користи протоколот „UDP“ заедно со „DTLS“. На ова ниво исто така може да се користи и протоколот „TCP“ иако „UDP“ е подобрата варијанта. Последното ниво е апликациското ниво. На ова ниво се наоѓаат протоколи како: „CoAP“, „MQTT“ и „XMPP“. Тоа се протоколи што подетално ќе ги разгледаме во овој магистерски труд. Најпрво ќе ги разгледаме нивните карактеристики, а потоа ќе извршиме и анализа на енергетската потрошувачка со нивна примена во повеќе сценарија.

2.1 Податочно ниво

Ова ниво е најниските слој од свитата на интернет протоколите, позната како „TCP/IP“. Тоа е група од методи и комуникациски протоколи што оперираат на линкот на кој е физички поврзан хостот. На ова ниво се остварува врската помеѓу хостовите или јазлите во мрежата.

2.1.1 IEEE 802.15.4

Стандардот „IEEE 802.15.4“ ги дефинира физичкото ниво (PHY) и поднивото за контрола на пристап (MAC) за ниско-податочни стапки за безжично поврзување за фиксни, преносливи и уреди во движење што немаат батерија или, пак, имаат батерија со ограничен капацитет [10]. Тој е стандардизиран од „IEEE“ слично како „IEEE 802.3“ за „Ethernet“ и „IEEE 802.11“ за безжични мрежи (WLANs) или „Wi-Fi“ [6]. Овој стандард се фокусира на ограничени средини со

мали ресурси (меморија, моќ и пропусен опсег). Тој ги адресира оние апликации за кои користењето безжични персонални мрежи (WPAN) е многу скапо или, пак, користењето на технологија како „Bluetooth“[17] не е соодветно. Притоа, овде влегуваат различни типови апликации како: индустриски, апликации од сферата на земјоделството, транспортни, апликации кои користат медицински сензори и активатори.

За да се постигне мала потрошувачка на енергија, се претпоставува дека е мал бројот на податоците кои се пренесени и тие не се пренесуваат постојано. Основната цел на овој стандард е да се овозможи сигурна и робусна технологија која ќе може да работи со години на стандардните батерии на уредите и да им овозможи на развивачите кои немаат многу познавања во оваа област да ги искористат поволностите што ги нуди овој стандард.

2.2 Интернет ниво

Интернет нивото претставува група методи, протоколи и спецификации кои се користат за пренесување датаграми (пакети) од изворните хостови до дестинации на други хостови кои се одредени со мрежна адреса („IP“-адреса).

2.2.1 „6LoWPAN“

„IETF“ го дефинира „6LoWPAN“ (IPv6 over Low-power Wireless Personal Area Network) како техника со која „TPC/IP“ се применува кај „WSN“[18] (Wireless sensor networks) [11]. Овој протокол е дефиниран во документот на „IETF“ под кодно име „RFC 6282“ [12]. Тој ги обезбедува јазлите кај безжичните сензорски мрежи со „IP“ комуникациски можности со поставување адаптациски слој веднаш над линк-нивото. Сензорните јазли кај „6LoWPAN“ се уреди кои се во согласност со „IEEE 802.15.4“ и, како што споменавме претходно, тие се карактеризираат со: мал опсег, мали брзини, мала потрошувачка на енергија, ограничени мемории и имаат релативно ниски цени.

Пакетите кај „IPv6“ се премногу големи за да може да се сместат во единечна „802.14.5“ рамка. За да се постигне тоа „6LoWPAN“ обезбедува два метода и тоа:

[17] Блутут (Bluetooth) – безжична технологија за разменување на податоци на кратки растојанија
[18] „WSN“ - мрежа од сензорни уреди кои имаат интегрирано сензори за температура, светлина, движење и слично.

- Компресија на заглавјето (header) – Компресија на заглавјето на пакетот кај „IPv6“ за да се намали неговата големина. „IETF“ го предлага кодирачкиот формат LOWPAN_IPHC[19] за компресија на заглавјето кај IPv6[12].
- Фрагментација и повторно склопување. Се врши фрагментација на пакетот кај „IPv6“ и се праќа преку повеќе пакети помали по големина кои што може да ги прифати една рамка кај „802.15.4“. На другата страна се врши повторно составување на фрагментираните пакети, со што повторно се креира „IPv6“ пакет [6].

2.3 Транспортно ниво

Транспортното ниво го овозможува транспортот на податоците. Тоа обезбедува средства за прифаќање на податоците од различни апликации и ги насочува тие податоци до апликација на приемниот уред (Multiplexing). На истиот начин, податоците на приемниот уред треба да бидат насочени до соодветна апликација, за која се наменети податоците (De-multiplexing) [13]. Ова ниво обезбедува интерфејс со кој мрежните апликации може да пристапат на мрежата. Тоа исто така дополнително обезбедува проверка на грешки, контрола на проток и верификација. Кај „IoT“ најчесто се користи протоколот „UDP“ заедно со „DTLS“ иако има случаи кога може да се користи и „TCP“.

2.3.1 „UDP“

Додека „TCP“ доминира кај Интернетот како протокол на транспортно ниво (со исклучок на игрите и на видеостримовите кои користат „UDP“), поголемиот дел од сценаријата кај „IoT“ користат „UDP“ [6]. „UDP“ е ненадежен сервис кој не обезбедува гаранции за испорака и нема заштита за копирање (ако се случи тоа, на пример, како резултат на софтверска грешка) [14]. Едноставноста на протоколот го прави соодветен за многу случаи. Овој протокол обезбедува минимален и ненадежен транспорт за пренесување на пораки до протоколите на погорното ниво, односно апликациското ниво. „UDP“ е уникатен по тоа што тој не воспоставува „крај-со-крај“ врска помеѓу комуникациските системи. Со ова се намалува оптоварувањето и се воспоставува една општа и минимална состојба помеѓу системите. Со овие карактеристики „UDP“ може да понуди ефикасна

[19] LOWPAN_IPHC – Compression Format for IPv6 Datagrams in 6LoWPAN Networks, https://tools.ietf.org/html/draft-hui-6lowpan-hc-00

комуникација кај некои апликации. Големината на заглавјето кај „UDP“ е многу помало во споредба со „TCP“ и тоа го прави овој протокол посоодветен за уреди што имаат ограничени перформанси и сензори.

2.3.2 „DTLS“

За безбедност кај протоколот „TCP“ се користи „TLS/SSL“, додека за безбедност кај „UDP“ се користи „DTLS“. Како и „TLS/SSL“, така и „DTLS“ ги обезбедува истите безбедносни особини на „UDP“ или датаграмите.

2.4 Апликациско ниво

Последното ниво од стекот на протоколите кај „IoT“ е апликациското ниво. Тоа е нивото кое главно се занимава со човечката интеракција и имплементација на софтверски апликации и соодветни протоколи. Тоа е нивото на кое се одвива целата комуникација помеѓу уредите за пристап кон други мрежи, јавниот Интернет и финалните апликации. На овој начин се овозможува онлајн-серверите да се ажурираат со последните вредности на крајните уреди, но исто така се овозможува пренесување команди од апликациите до крајните уреди [15]. Најупотребувани протоколи кои се користат на апликациско ниво кај Интернетот не нештата се следните: „Constrained Application Protocol“ (CoAP), „Message Queuing Telemetry Transport“ (MQTT) и „Extensible Messaging and Presence Protocol“ (XMPP). Разликите помеѓу овие протоколи се дадени во Табела 2. Може да се забележи дека на транспортно ниво „CoAP“ го користи протоколот „UDP“ додека другите два протокола го користат протоколот „TCP“. „CoAP“ и „MQTT“ имаат поддршка за „QoS“ (Quality of Service). „CoAP“ има архитектура на „Барање/Одговор“ (Request/Response) протокол. „MQTT“ има архитектура на „Објави/Претплати“ (Publish/Subscribe) протокол. „XMPP“ има поддршка и за двата типа архитектура. „CoAP“ за безбедност го користи „DTLS“, додека другите два протокола го користат „TLS/SSL“. Главна тема на оваа

Табела 2. Разлики помеѓу протоколите на апликациско ниво кои се користат кај „IoT“

Protocol	**Transport**	**QoS options**	**Architecture**	**Security**
CoAP	UDP	YES	Request/Response	DTLS
MQTT	TCP	YES	Publish/Subscribe	TLS/SSL
XMPP	TCP	NO	Request/Response, Publish/Subscribe	TLS/SSL

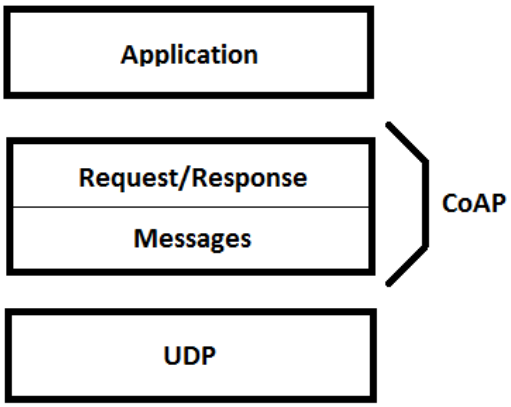
магистерска работа се токму овие протоколи и во следниот дел ќе бидат опишани подетално нивните карактеристики и функцијата што ја имаат. На ова ниво постојат и други протоколи како што се: „Advanced Message Queuing Protocol“ (AMQP), „Representational State Transfer“ (Restful services) и „Websockets“.

2.4.1 „Constrained Application Protocol“ (CoAP)

„CoAP“ е специјализиран веб-трансфер протокол кој се користи од ограничени уреди и мрежи. Уредите (јазлите) често имаат 8-битни микроконтролери со мало количество RAM и ROM, додека ограничените мрежи, како што е „6LoWPAN“ често имаат големи рати на грешки во пакетите и типичен податочен проток од 10 kbit/s. „CoAP“ обезбедува „барање/одговор“ интеракциски модел помеѓу апликациските крајни точки. Тој поддржува вградено откривање сервиси и ресурси и ги вклучува клучните концепти на вебот како што се „URI“ (Uniform Resource Identifier) и Интернет медиа типовите (Internet Media Types). „CoAP“ е дизајниран така што да може лесно да комуницира со „HTTP“ за интеграција со веб, додека се среќава со специјализирани особини како што е мултикаст-поддршката, многу малиот „overhead“ и едноставноста за ограничени средини, асинхрона размена на пораки, способности за кеширање и прокси, безбедност со користење на „DTLS“, исполнување на барањата на уредите кои имаат органичени карактеристики [16].

Интеракцискиот модел на „CoAP“ е сличен на клиент/сервер моделот кој го применува „HTTP“. Во основа „CoAP“ клиентот испраќа барање (притоа користејќи метод-кодови (Method Codes) до ресурс кој се наоѓа на серверот и кој е идентификуван со „URI“. На ова барање одговара серверот со користење кодови за одговор (Response codes). Одговорот може да содржи и одредени ресурси.

„CoAP“ дефинира четири типа пораки и тоа: „Confirmable“ (потврдувачки), „Non-Confirmable“ (непотврдувачки), „Acknowledgment“ (верификувачки), и „Reset“ (ресетирачки). Во овие типови пораки се вклучуваат метод-кодовите и кодовите за одговор, со што пренесуваат или барања, или одговори. Барањата може да се пренесуваат како „Confirmable“ и „Non-confirmable“ пораки, додека одговорите може да се пренесуваат преку овие, а и преку „piggybacked“ (тип на порака со која одговорот се враќа веднаш при испратено барање) и пораките

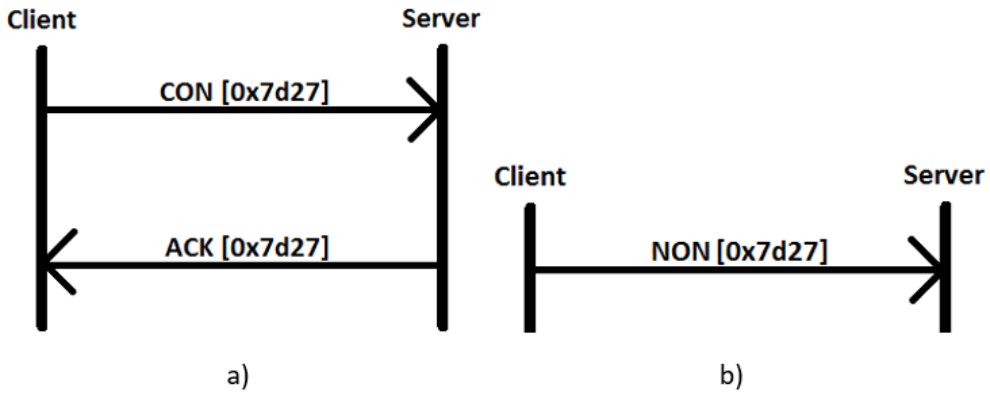


Слика 8. Апстрактни слоеви кај „CoAP“

„Acknowledgement“. Овој протокол може да го сфатиме како целина која се состои од две поднивоа (Слика 8). На првото подниво се наоѓаат пораките со кои „CoAP“ комуницира со транспортниот слој. На второто подниво се наоѓаат интеракциите за барање/одговор со користење на метод-кодовите и кодовите за одговор.

2.4.1.1 Модел на пораки кај „CoAP“

Моделот на пораки кај „CoAP“ се потпира на размената на пораки преку протоколот „UDP“ помеѓу крајните точки. „CoAP“ користи кратко заглавје со фиксна големина од 4 бајти. Заглавјето може да биде следено од бинарни опции или, пак, товар (payload). Секоја порака содржи свој идентификациски број (Message ID). Тој се користи за да може да се детектираат дупликати на пораките и за опционална доверливост. Доверливоста најчесто се обезбедува со користење на типот на порака „Confirmable“ (CON) (Слика 9a). Овој тип на порака по одредено време се препраќа, сè додека примачот не врати „Acknowledgement“ (ACK) потврдна порака што го има истиот идентификациски број на порака (Message ID), како и бројот на пораката „Confirmable“. За пораките за кои не е



Слика 9. a) Доверливо пренесување на „CoAP“ пораки; b) Недоверливо пренесување на „CoAP“ порака

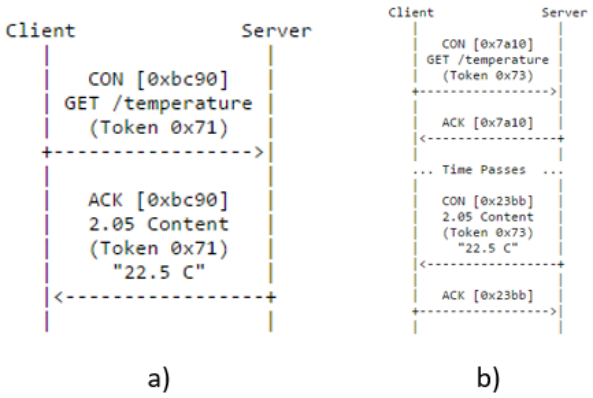
потребна доверлива трансмисија (на пример: испраќање податоци од даден сензор) може да се испратат како „Non-Confirmable“ (NON) (Слика 9b). Овие пораки не бараат потврда, но мора да имаат „ID“ за да нема повторување (дупликати) на пораката. Ако оној кој ја прима пораката „NON“ не е во можност да ја процесира, тој може да одговори со порака за ресет (Reset Message - RST).

2.4.1.2 Модел „барање/одговор“

Барањата и нивните одговори се пренесуваат со „CoAP“ пораки. Овие пораки содржат или метод-кодови или кодови за одговор. Некои информации за барања и одговори како што се URI и типот на товарот (payload media type) се пренесуваат како „CoAP“ опции. За поврзување на барањата и одговорите се користи токен (Token).

Постојат два типа одговори на барањата и тоа: „piggybacked“ и „separate“ (посебен одговор) (Слика 10). Ако барањето се пренесува со порака „CON“ или „NON“, и ако одговорот е достапен и се пренесува со порака „ACK“, тогаш станува збор за „piggybacked“ одговор (Слика 10a). Клиентот доставува до серверот барање за добивање податок за температурата користејќи порака „CON“. За добивање на температурата се користи методот „GET“. „CoAP“ ги користи „GET“, „PUT“, „POST“ и „DELETE“ на сличен начин како „HTTP“. На ова барање серверот одговара веднаш со податок „22.5 C”. Може да се види дека токенот и идентификацискиот број на пораката и во двата случаја се исти, што значи дека одговорот се однесува на тоа барање.

Ако серверот не е во можност да одговори веднаш на барањето, тој може да испрати празна порака „ACK“, со која му кажува на клиентот да престане да го препраќа барањето. Кога серверот ќе може да му одговори на клиентот, тој му



Слика 10. a) „Piggybacked“ одговор b) посебен одговор (separate response), извор: „The Constrained Application Protocol“ (CoAP), https://tools.ietf.org/html/rfc7252

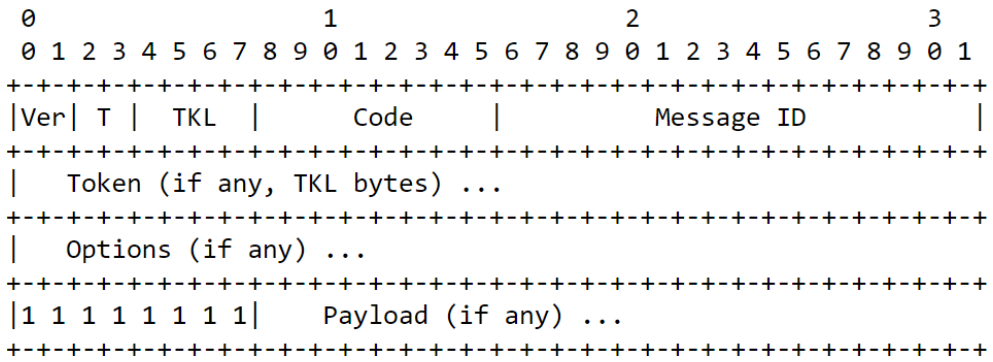
праќа порака „CON“, која потоа треба да биде потврдена од клиентот. Ова се нарекува посебен одговор (Слика 10b). Може да се види дека „ID“ на пораките „CON“ за барање и одговор се разликува бидејќи станува збор за две различни пораки „CON“. Токенот за двете пораки е ист, што значи дека одговорот се однесува на даденото барање.

Кај пораките „NON“ кои се праќаат како барање, одговорот се праќа преку нова порака „NON“, но не е исклучена и можноста серверот да може да одговори со порака „CON“.

2.4.1.3 Формат на порака

Пораките кај „CoAP“ се кодирани во едноставен бинарен формат (Слика 11). Пораките започнуваат со фиксно заглавје од 4 бајти. По заглавјето следи токен полето (Token), кое е со големина од 0 до 8 бајти. Потоа следува полето за опции (Options), кое е следено од полето „Payload“ (кое е опционално). Полињата кои го составуваат заглавјето на пораката се следните:

- „Version“ (Ver): Ова поле ја дефинира верзијата на „CoAP“. Тоа е 2 битен „unsigned integer“ (цел број).
- Type (T): Ова поле го прикажува типот на пораката. Бројот се дефинира според следниот редослед: „Confirmable“ (0), „Non-Confirmable“ (1), „Acknowledgement“ (2) или „Reset“ (3). Тоа е двобитен „unsigned integer“.
- „Token Length“ (TKL): Ја прикажува должината на полето „Token“ (0-8 бајти).
- „Code“: Тоа е осумбитен „unsigned integer“. Поделен е на два дела и тоа: 3-битна класа (најзначајните битови) и 5-битни детали (најмалку значајните битови). Форматот на кодот е „c.dd“, при што „c“ е цифра од 0



Слика 11. Формат на „CoAP“ порака,
извор: „ The Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc7252

до 7 и ја претставува класата додека „dd“ се две цифри од 00 до 31. Според класата може да го утврдиме типот на пораката и тоа: барање (0), успешен одговор (2), грешка кај клиентскиот одговор (4), грешка кај серверскиот одговор (5). „CoAP“ има посебен коден регистар во кој е даден опис за сите кодови.

- „Message ID“: Идентификациски број на пораката што се користи за да се детектираат пораки кои се дупликати и за да се поврзат пораките од типот „Acknowledgement/Reset“ со пораките од типот „Confirmable/Non-Confirmable“. Тој е 16-битен „unsigned integer“.

Веднаш по заглавјето на пораката следи полето „Token“. Ова поле се користи за поврзување на барањата и одговорите. Тоа е со должина од 0 до 8 бајти.

По полето „Token“ следи полето „Options“ кое дефинира една или повеќе опции. „CoAP“ дефинира единечен збир на опции што се користат и кај барањата, и кај одговорите. Тоа се следните: „Content-Format“, „Etag“, „Location-Path“, „Location-Query“, „Max-Age“, „Proxy-Uri“, „Proxy-Scheme“, „Uri-Host“, „Uri-Path“, „Uri-Port“, „Uri-Query“, „Accept“, „If-Match“, „If-None-Match“, „Size“.

- „Content-Format“: го дефинира форматот на товарот на пораката (payload format). Форматот е даден како нумерички идентификатор кој е дефиниран во регистарот „CoAP Content Format“.
- „ETag“ (Entity-Tag): се користи како локален идентификатор на ресурси за разликување на претставите на ист ресурс кој варира со текот на времето.
- „Location-Path“ и „Location-Query“: Овие две опции заедно го прикажуваат релативниот „URI“, кој се состои или од апсолутната патека, или стрингот за пребарување, или двата. Пример за ова е комбинација од овие опции која е вклучена кај одговорот 2.01 (Created[20]) за да ја прикаже локацијата на ресурсот што е креиран како резултат на барањето „POST“.
- „Max-Age“: Оваа опција го претставува максималното време за кое одговорот може да биде кеширан, пред да се смета како не-обновен.

[20] „2.01 Created“ – код за одговор кој се користи во случаи кога се употребуваат POST и PUT како барања.

Ова е „integer“ број и се изразува во секунди. Ако ја нема оваа опција, тогаш ова време е 60 секунди.

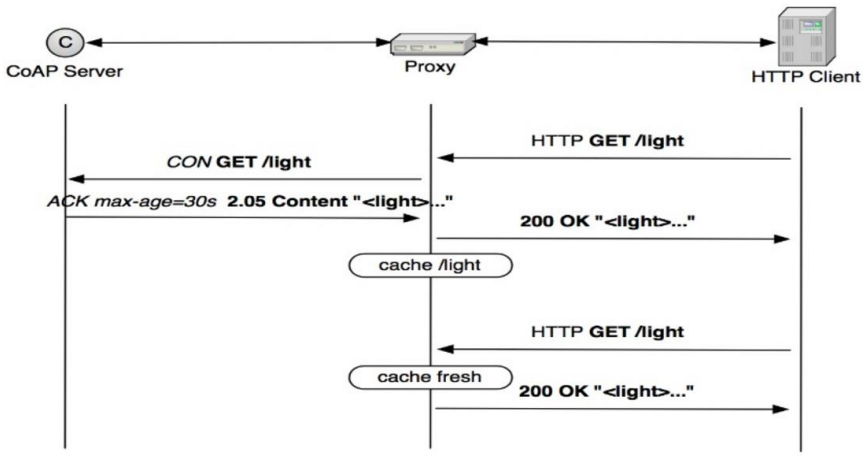
- „Proxy-Uri“ и „Proxy-Scheme“: Опцијата „Proxy-Uri“ се користи за да се направи барање за „forward-proxy“[21]. Вредноста на оваа опција е апсолутно-„Uri“. „Proxy-Scheme“ се користи за формирање „URI“.
- „Uri-Host“, „Uri-Path“, „Uri-Port“, „Uri-Query“: Овие опции се користат за да се дефинира целниот ресурс на барањето до „CoAP“ сервер. „Uri-Host“ го дефинира интернет хостот на ресурсот што се бара. „Uri-Path“ претставува еден сегмент од апсолутната патека на ресурсот. „Uri-Port“ го дефинира бројот на портата на ресурсот на транспортно ниво. „Uri-Query“ претставува доделување параметар кој влијае врз начинот на кој се враќа ресурсот.
- „Accept“: Одредува кој формат на содржина е прифатлив за клиентот.
- „If-Match“ и „If-Non-Match“: „If-Match“ се користи за барањето да се направи зависно до моменталната состојба или вредноста на „E-Tag“ за една или повеќе претстави на целниот ресурс. „If-Non-Match“ се користи за да се направи условно барање на непостоењето на даден ресурс.
- „Size1“: Обезбедува информации за големината на ресурсот во барањето.

Ако постојат опции, по нив следи и „Payload Marker“, кој го прикажува крајот на полето „Options“ и почетокот на полето „Payload“. Полето „Payload“ го содржи одговорот на барањето или тоа е претставата на ресурсот. Неговиот формат е одреден од интернет медиа типот или, пак, од „Content-Format“ опцијата. Постои и таканаречен „Diagnostic Payload“, кој се јавува во случај кога има клиентска или серверска грешка и најчесто тоа е порака за грешка која корисникот може да ја прочита.

2.4.1.4 Прокси и кеширање

Протоколот “CoAP“ поддржува кеширање на одговорите сè со цел поефикасно исполнување на барањата. Кеширањето е овозможено со користење информации за валидност (validity) и свежина (freshness) [17]. Кога одговорот е свеж (fresh), тој може да се користи како одговор на барањето, без

[21] „Forward-proxy“ – се користи за да го пренасочи барањето од валиден кеш и да го врати одговорот



Слика 12. Кеширање на одговор, извор: „ The Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc7252

да мора да се контактира серверот притоа. Свежината на одговорот се одредува според опцијата „Max-Age“. Односно, одговорот не е свеж ако времето кое го поминал (изразено во секунди) е поголемо од времето кое е дефинирано во опцијата „Max-Age“, што исто така е изразено во секунди. Моделот за валидација ја користи опцијата „ETag“ за да ја ажурира свежината на складираниот (кешираниот) одговор. Кешот може да биде лоциран на крајните точки или помеѓу нив. Клиентот поднесува барање до серверот преку посредник (proxy) кој е должен да го проследи барањето. Откако серверот може да одговори на барањето, тој одговорот го доставува до проксито каде што истиот може да се кешира (Слика 12). Потоа проксито го доставува одговорот до клиентот. Кога клиентот праќа ново барање за истиот ресурс, во тој случај ако одговорот е претходно кеширан, проксито го враќа одговорот до клиентот без да го контактира серверот притоа. Со ова се заштедува време а воедно и клиентот бргу го добива одговорот на барањето.

Употребата на посредници (proxy) е доста значајно за ограничените мрежи и тоа од неколку причини: ограничување на мрежниот сообраќај, подобрување на ефикасноста, пристап до уреди кои се во неактивна состојба (sleeping devices) и од безбедносни причини. Кога се користи прокси, „URI“ на ресурсот е вклучено во барањето, додека како целна „IP“-адреса е поставена адресата на проксито. Прокси може да се користи и како преведувач помеѓу два различни протокола како што се: „HTTP“ и „CoAP“ (cross-proxy). Прокси може да биде експлицитно селектиран од клиентите, со улога позната како „forward-proxy“, или серверите позната како „reverse-proxy“. Проксито може да мапира и

од „CoAP“ барање до „CoAP“ барање или уште попознато како „CoAP-to-CoAP proxy“.

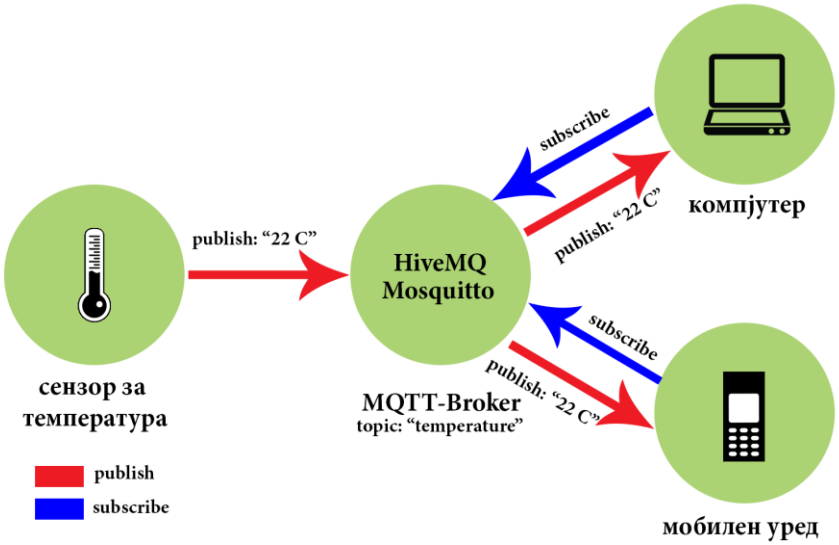
2.4.2 Протокол Message Queuing Telemetry Transport (MQTT)

Според официјалната документација за протоколот „MQTT“[18]:

„MQTT“ е клиент-сервер „објави/претплати“ протокол за пренесување пораки. Тој е лесен, отворен, едноставен и е дизајниран за да може лесно да се имплементира. Овие карактеристики го прават идеален за користење во повеќе ситуации вклучувајќи ограничени средини како што се комуникацијата „машина-до-машина“ (M2M) и Интернетот на нештата (IoT), при што се бара мала големина на кодот и пропусниот опсег е од примарно значење.

MQTT е протокол кој бил откриен од страна на Енди Станфорд Кларк (Andy Stanford-Clark) од „IBM“ и Арлен Нипер (Arlen Nipper) од „Arcom“ (сега „Eurotech“) во 1999 година и бил стандардизиран во 2013 година како стандард OASIS[22] [19]. Основната нивна цел била да развијат протокол кој ќе обезбеди минимална потрошувачка на батеријата и минимален пропусен опсег поврзувајќи нафтоводи преку сателитска конекција. Тие си задале и други цели што требало да ги исполнува нивниот протокол, и тоа: да биде лесен за имплементација, да обезбеди квалитет на сервисот при доставата на податоци, да биде лесен и ефикасен, да може да пренесува различни типови податоци (data agnostic) и да има способност да одржува сесии [20]. Денешниот протокол „MQTT“ ги има истите карактеристики кои биле предвидени тогаш. „MQTT“ денес главно се ориентира кон решенија кај „IoT“, за разлика од минатото кога повеќе бил фокусиран на комерцијални вградени системи. Често се јавуваат прашања за тоа како настанала кратенката „MQTT“. Според некои мислења, „MQTT“ официјално нема акроним, и е само „MQTT“ [20]. Според други мислења кратенката доаѓала од „MQ Telemetry Transport“, при што „MQ“ се однесува на „MQ Series“, што е продукт кој бил развиен од „IBM“ и го поддржувал овој протокол. Така и протоколот го добил своето име уште кога бил откриен во 1999 година. Во кратенката се споменува и зборот „queue“, но тоа не е точно. Не постојат редови (queues) во традиционалните решенија за пренесување пораки. „MQTT“ бил користен интерно од „IBM“ извесно време. Верзијата 3.1 била

[22] OASIS - глобален непрофитен конзорциум што работи на развој и приспособување стандарди за „IoT“, безбедност и технологија.



Слика 13. Publish/Subscribe (Објави/Претплати модел)

развиена во 2010 година како слободна верзија. Оттогаш па наваму секој можел да ја имплементира и да ја користи. По 3 години од првичното објавување, „MQTT“ бил стандардизиран од организацијата „OASIS“. Процесот за стандардизација траел цела една година и на 29 октомври 2014 година „MQTT“ бил официјално потврден како стандард „OASIS“. Последната верзија на протоколот е „MQTT 3.1.1“, која има мали измени во однос на претходната верзија „3.1“ [18]. Со стандардизацијата се овозможува и приспособување на карактеристиките на уредите кои имаат план да го користат овој протокол за свои специфични цели. Производителите на уреди при нивното креирање ќе обрнуваат внимание на карактеристиките, ограничувањата и на предностите кои им ги нуди овој протокол.

2.4.2.1 Модел Објави/Претплати

Протоколот „MQTT“ како основен модел го користи моделот „Publish/Subscribe“ (Објави/Претплати) (Слика 13). Овој модел претставува алтернатива на традиционалниот клиент/сервер модел, каде што клиентот комуницира директно со крајните уреди [20]. Кај овој модел клиентите се одделени. Постојат клиенти што праќаат пораки кои се наречени објавувачи (publishers) и клиенти што примаат пораки кои се наречени претплатници (subscribers). Кај овој модел постои уште една компонента што ја има улогата на посредник, која ја знаат и објавувачот, и претплатникот и таа е наречена брокер (broker). Функцијата на оваа компонента е да ги филтрира сите пристигнати пораки и да ги дистрибуира соодветно. Целта на овој модел е да се одделат

објавувачите и претплатниците кои може да се разликуваат по неколку нешта [20]:

- Просторно одделување: Објавувачите и претплатниците не мора да се познаваат еден со друг.
- Временско одделување: Објавувачите и претплатниците не мора да работат во исто време.
- Синхронизирано одделување: Операциите на двете компоненти не се прекинуваат за време на објавувањето или на примањето на пораките од брокерот на клиентите кои се претплатени.

Со одделувањето на клиентите и со воведувањето на брокерот кој ги филтрира се овозможува само оние клиенти на кои им е дозволен пристап (оние што се претплатени) да може да ги добијат пораките.

Постојат неколку начини на филтрирање на пораките од серверот и тоа:

- Филтрирање што е базирано на предмет: На овој начин се врши филтрирање кое е базирано на предмет или тема (topic), што е дел на секоја порака. Клиентот се претплатува на дадена тема на брокерот и од него ги добива сите пораки за кои е претплатен. Темите во основа се стрингови со хиерархиска структура, кои овозможуваат филтрирање кое е базирано на ограничен број изрази.
- Филтрирање што е базирано на содржина: Се врши филтрирање врз основа на одредена содржина. Недостаток на овој филтер е тоа што содржината на пораките мора да се знае однапред и тие не може да се шифрираат или да се менуваат лесно.
- Филтрирање кое е базирано на тип: Во овој случај се врши филтрирање според типот на пораките. Така претплатникот може да ги слуша сите пораки кои се од даден тип или поттип.

„MQTT“ најчесто го користи филтрирањето на пораки кое е базирано на одредена тема. Така, секоја порака содржи тема, што ја користи брокерот за да може да ги проследи пораките кои биле доставени претходно до него од објавувачот со истиот назив на темата. „MQTT“ ги одделува претплатниците и објавувачите, така што тие треба само да го знаат името на хостот или „IP“ и

портата на брокерот за да може да се претплатуваат или пак да објавуваат на дадена тема [20]. Брокерот е способен да ги складира пораките за клиентите кои не се онлајн. Следниот пат кога ќе бидат онлајн, тој им ги доставува пораките. Ова посебно се однесува за претплатниците.

Со цел да се справи со предизвиците кај системите што го користат овој протокол, „MQTT“ има нивоа за квалитет на сервис (QoS). Со тоа е лесно да се прецизира дека дадена порака успешно е доставена од клиентот до брокерот или, пак, обратно.

Овој модел покрај предностите има и одредени недостатоци. Во случајот на филтрирање кое е базирано на предмет, претплатниците и објавувачите мора да знаат кои се правилните теми кои треба да ги користат. Во случајот на доставување на пораките, објавувачот не може да претпостави дека некој е претплатен на пораките што тој ги праќа. Со тоа може да се јави случај во кој пораките не се прочитани од ниту еден претплатник.

Овој протокол е лесен за користење на клиентската страна. Поголемата логика кај системите кои го користат овој модел се наоѓа кај брокерот, и доволно е да се користат клиентски библиотеки[23] за „MQTT“. Тоа го прави овој протокол лесен за користење и идеален за мали уреди што имаат ограничени перформанси.

2.4.2.2 Клиент, брокер и воспоставување врска

„MQTT“ клиент претставува кој било уред што има „MQTT“ библиотека која работи на неговата страна и е поврзана со „MQTT“ брокер преку кој било тип на мрежа. Тоа може да биде мал уред со ограничени перформанси или, пак, типичен компјутер. Овие уреди мора да имаат поддршка за „TCP/IP“ стек за да може да се имплементира „MQTT“. Постојат два вида на „MQTT“ клиенти и тоа: претплатници и објавувачи. Овој протокол ги поддржува сите платформи и голем број програмски јазици [21]. Дел од јазиците се следните: „C“, „C++“, „C#“, „Go“, „Java“, „JavaScript“, „.NET“, „Android“, „iOS“, „Arduino“. Постојат и други имплементации на клиентски библиотеки за „MQTT“ [22]. Во основата на протоколот „MQTT“ се наоѓа брокерот. Тој е одговорен да ги прими сите пораки,

[23] Постојат повеќе вакви библиотеки за различни програмски јазици за имплементација на MQTT на клиентската страна

да ги филтрира, да определи кој е заинтересиран за пораките и да ги прати пораките до сите претплатени клиенти. Во зависност од неговата конкретна имплементација, брокерот може да се справи со илјадници „MQTT“ клиенти кои се поврзани истовремено. Тој може да одржува и сесија за сите постојани клиенти вклучувајќи ги овде и претплатите, како и пораките кои се пропуштени. Брокерот е задолжен и за автентикација и авторизација на клиентите. Постои и можност за самостојна, произволна автентикација и авторизација во позадината на системот и негова интеграција со брокерот. Интеграцијата е важен аспект за овие системи бидејќи брокерот е компонента која е изложена на Интернет и пристап до неа имаат голем број клиенти до кои се проследуваат голем број пораки. Постојат повеќе типови брокери, а како попознати ќе ги споменеме: „Mosquitto“, „HiveMQ“ и „Mosca“ [23].

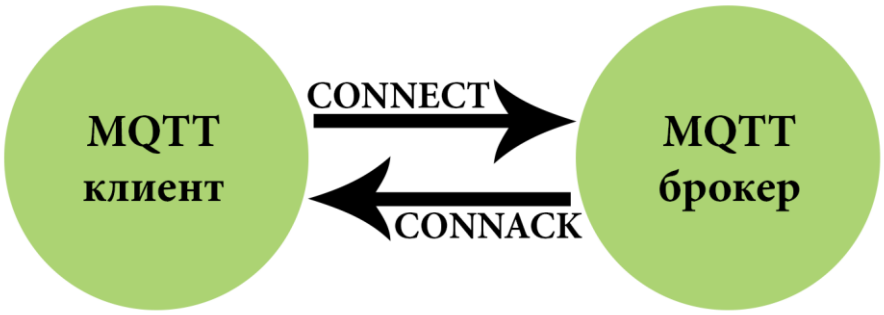
„Mosquitto“ е „MQTT“ брокер со отворен код, кој ги има имплементирано верзиите „3.1“ и „3.1.1“ на „MQTT“ протоколот [24]. Овој брокер обезбедува лесен метод за пренесување пораки користејќи го моделот „објави/претплати“. Ова го прави овој брокер идеален за сензори со мала моќ, мобилни телефони, вградени компјутери или микроконтролери како што е „Arduino“ [24].

„HiveMQ“ им овозможува на компаниите да ги поврзат нивните уреди и сервиси со минимален напор со користење на стандардниот „IoT“-протокол „MQTT“. Овој брокер се наоѓа на границата помеѓу уредите и системите на компаниите. Со неговото користење уредите и сервисите ги добиваат последните информации преку инстант, двонасочни пораки. Тековните податоци може да се пратат до бекенд-системите во реално време [20].

„Mosca“ е „MQTT“-брокер. Тоа значи дека „Mosca“ го имплементира протоколот „MQTT“, со кој може да извршува сопствен „mqtt“-сервер на „node.js“[24]. Тој ги поддржува верзиите „MQTT 3.1“ и „MQTT 3.1.1“. Има поддршка за „QoS 0“ и „QoS 1“ и поддржува различни опции за складирање на „QoS 1“ офлајн-пакети и претплати.

Протоколот „MQTT“ работи на база на „TCP/IP“. Клиентите и брокерот мора да имаат поддршка за „TCP/IP“ стекот. Врската кај „MQTT“ секогаш се

[24] „Node.js“ – платформа со отворен код за развивање веб-апликации на страната на серверот, https://nodejs.org/en/



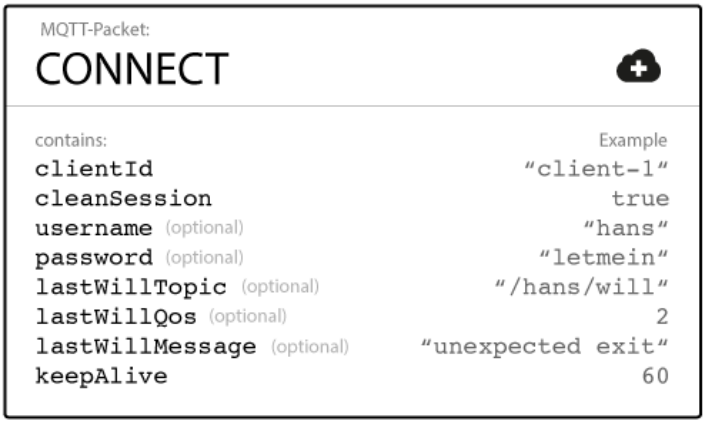
Слика 14. MQTT врска помеѓу клиент и брокер

остварува помеѓу клиентите и брокерот. Не постои директно поврзување на клиент со клиент и секогаш брокерот ја игра улогата на посредник помеѓу клиентите.

Врската кај „MQTT“ се иницира со праќање порака „CONNECT“ до брокерот. Брокерот на тоа одговара со „CONNACK“ и со статусен код (Слика 14). Откако ќе биде воспоставена врската, брокерот ја чува отворена сè додека клиентот не се дисконектира или додека не се изгуби врската. Брокерот ја затвора врската во случаите кога пораката „CONNECT“ е погрешна (не е според спецификацијата на „MQTT“) или ако отворањето на врската трае премногу долго. Тоа сè прави пред се од безбедносни причини, за да се спречат злонамерни клиенти кои имаат за цел да го нападнат или да ја забават работата на брокерот.

Пораката „CONNECT“ се состои од неколку атрибути и тоа: „clientId“, „clean Session“, „username“, „password“, „lastWillTopic“, „lastWillQoS“, „lastWillMessage“ и „keepAlive“ (Слика 15).

- „ClientId“: Овој атрибут е клиентски идентификатор на секој „MQTT“ клиент кој се поврзува на „MQTT“ брокер. Овој идентификатор треба да биде уникатен за брокерот. Брокерот го користи него за да ги идентификува клиентот, како и моменталната состојба на клиентот.
- „Clean session“: Овој атрибут му кажува на брокерот дали клиентот сака да воспостави постојана сесија или, пак, не. Кога се воспоставува постојана сесија, вредноста на „clean session“ е „false“. Тоа значи дека брокерот ќе ги складира сите претплати за клиентот и сите пропуштени пораки. Тоа важи во случаите кога претплатата е извршена со „QoS 1“ или со „QoS 2“. Ако вредноста на оваа опција е „true“, тогаш брокерот нема да



Слика 15. Порака „CONNECT" за иницирање на врска,
извор: „MQTT Essentials Part 3: Client, Broker and Connection Establishment",
http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment

зачувува ништо за клиентот и ќе ги избрише сите информации од претходната постојана сесија.

- „Username" и „password": За автентикација и за авторизација на клиентот се користат корисничко име и лозинка. Ако лозинката не е шифрирана или хеширана со имплементацијата или со користење на „TLS", тогаш лозинката се праќа како чист текст. Постои можност за идентификување на клиентите и со „SSL" сертификати, со што во тој случај не е потребно корисничко име и лозинка.
- „WillMessage": Со овој атрибут се овозможува другите клиенти да бидат известени кога даден клиент се дисконектира неочекувано. За да се имплементира тоа, поврзаниот клиент испраќа „MQTT" порака која ја содржи темата на која е претплатен. Ако дојде до губење на врската, тогаш брокерот ја праќа оваа порака во име на клиентот.
- „KeepAlive": Овој атрибут го претставува временскиот интервал во кој клиентите праќаат регуларно „PING"[25]-барање до брокерот. Брокерот одговара на ова барање, и со овој механизам се утврдува дали другата страна е достапна.

По праќање на порака „CONNECT" од клиентот, брокерот мора да одговори со своја порака „CONNACK". Тоа е порака со која брокерот го известува клиентот дали врската е возможна и успешна. Оваа порака има два типа атрибути и тоа: „sessionPresent" и „returnCode". „SessionPresent" покажува дали

[25] PING – команда со која се тестира достапноста на даден хост

Табела 3. Вратени кодови кај порака „CONNACK“

Вратен код (Return code)	**Значење на кодот**
0	Прифатена врска
1	Одбиена врска, неприфатлива верзија на протокол
2	Одбиена врска, идентификаторот е отфрлен
3	Одбиена врска, серверот е недостапен
4	Одбиена врска, погрешно корисничко име или лозинка
5	Одбиена врска, ако не сте овластен

брокерот веќе има остварено постојана врска со клиентот во некои претходни интеракции. Ако при врската на клиентот опцијата „CleanSession“ е „true“, во тој случај нема достапна сесија и „sessionPresent“ е „false“. Во спротивно, остварена е постојана сесија и на ова поле му е зададена вредност „true“. Оваа опција е додадена во последната верзија на „MQTT“ протоколот, односно „MQTT 3.1.1“ и им помага на клиентите да одредат дали тие треба да се претплатат на темите или тие веќе се складирани во сесијата. Вториот атрибут кај „CONNACK“ е „returnCode“. Тој покажува дали обидот за остварување конекција бил успешен, и ако не е која е причината за тоа. Во Табела 3 се прикажани сите можности на вратени кодови, како и нивното значење.

2.4.2.3 MQTT објава, претплата и прекинување на претплата

Откако клиентот ќе се конектира на брокерот, тој може да објавува пораки. Секоја порака мора да содржи тема која се користи од брокерот за да ја препрати пораката на заинтересираните клиенти кои се претплатени на истата тема. Секоја порака има товар (payload) кој се користи за пренесување на податоците во бајт-формат. Атрибутите на пораката за објавување се следните (Слика 16):

Слика 16. „Publish“-порака за објава,
извор: MQTT Essentials Part 4: MQTT Publish, Subscribe & Unsubscribe,

- „PacketId“: Претставува уникатен идентификатор помеѓу клиентот и брокерот за да се идентификува пораката. Ова се однесува само за квалитет на сервисот (QoS) поголем од 0.
- „TopicName“: Едноставен стринг со хиерархиска структура. Елементите на структурата се одделени со коси црти (Пример: myhome/livingroom/light).
- „QoS“: Квалитет на сервис кој со своите нивоа гарантира дека пораката го достигнува другиот крај. Има 3 вида квалитет на сервис означени како „QoS 0“ (потврдува дека пораката го достигнува другиот крај најмногу еднаш), „QoS 1“ (потврдува дека пораката го достигнува другиот крај најмалку еднаш) и „QoS 2“ (потврдува дека пораката го достигнува другиот крај точно еднаш).
- „RetainFlag“: Ова поле одредува дали пораката ќе биде зачувана од брокерот за наведената тема како последно позната вредност. Кога ќе се претплатат нови клиенти на дадената тема, тие ќе ја примат последно зачуваната вредност веднаш откако ќе се претплатат на темата.
- „Payload“: Ова е содржината на пораката. „MQTT“ подржува повеќе формати на податоци (data-agnostic). Можно е да се праќаат слики, кодирани текстови во кој било формат, кодирани податоци и сите податоци во бинарна форма.
- „DupFlag“: Ова знаменце се користи за да се утврди дали пораката е дупликат или не. Ова важи само за „QoS“ поголем од 0.

Кога даден клиент објавува на дадена тема, брокерот ја чита објавата и ако е потребно, ја потврдува (во зависност од „QoS“) и потоа ја препраќа пораката до клиентите кои се претплатени на таа тема.

Клиентите што сакаат да се претплатат на дадена тема на брокерот праќаат порака „SUBSCRIBE“ (Слика 17). Оваа порака е едноставна и се состои само од

Слика 17. Порака „SUBSCRIBE“,
извор: „ MQTT Essentials Part 4: MQTT Publish, Subscribe & Unsubscribe“,http://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe

Табела 4. Вратени кодови кај порака „SUBACK“

Вратен код (Return code)	Значење на кодот
0	Успех – Минимум QoS 0
1	Успех – Минимум QoS 1
2	Успех – Минимум QoS 2
128	Неуспех

единствен идентификатор на пакет (функцијата му е иста како кај „PUBLISH“) и листа на претплати. Листата на претплати се состои од парови на теми за претплата и квалитет на сервис. Листата може да биде произволна.

Секоја претплата ќе биде потврдена од брокерот со праќање на потврда до клиентот во форма на порака „SUBACK“. Оваа порака го содржи истиот идентификатор како оригиналната порака за претплата (SUBSCRIBE) и листа на вратени кодови. Идентификаторот се користи за да се идентификува пораката. Тој е ист како кај пораката за претплата. Брокерот враќа еден код за секој пар тема/„QoS“. Овој код го дава исходот од претплатата на дадена тема (Табела 4). По претплатувањето на дадена тема и по добивањето порака за потврда (успешна претплата), клиентот ќе ја прима секоја порака која ќе биде објавена на таа тема.

За прекинување на претплата на дадена тема се користи пораката „UNSUBSCRIBE“, која ја брише претплатата на клиентот на брокерот. Таа е едноставна порака која се состои од само два атрибута и тоа: идентификатор на пакет (packetId) и листа на теми за кои е потребно да се избрише претплатата.

Брокерот ќе го потврди барањето за прекинување на претплата со порака „UNSUBACK“. Таа го содржи само идентификаторот на пакетот. Тој е ист како и кај пораката „UNSUBSCRIBE“.

2.4.2.4 Теми (Topics)

Темата е „UTF-8“ стринг кој се користи од брокерот за да се филтрираат пораките за секој поврзан клиент [20]. Темите се состојат од едно или од повеќе

нивоа кои составуваат една хиерархиска структура. Секое ниво кај темата е одделено со коса црта, како во следниот пример:

```
myhome/livingroom/light
```

Секоја тема мора да има најмалку еден карактер за да биде валидна. Таа исто така може да содржи и празни места. Само коса црта (/) е исто така валидна тема. Темите се „case-sensitive“, односно има разлика помеѓу големите и малите букви. Следните примери се две различни теми:

```
myhome/livingroom/light

MyHome/livingroom/light
```

Кога клиентите се претплатуваат на дадена тема, тие може да ја користат точната тема на која пораката е објавена или може да се претплатат на повеќе теми одеднаш со користење „wildcards“. Тие може да се користат само за претплата а не може да се користат за објавување пораки. Постојат два типа „wildcards“ и тоа:

- Со единечно ниво (single level): Се користи како замена на едно ниво во темата. Се користи знакот „+“ за означување на тоа ниво.
  Пример: „myhome/groundfloor/+/light“
  Во претходниот пример знакот „+“ може да биде заменет со кој било збор. Кога ќе се претплатиме на оваа тема, тогаш претплатата важи за сите теми што ја имаат истата структура и имаат само промена кај знакот „+“ (Пример: „myhome/groundfloor/livingroom/light“).
- Со повеќе нивоа (multi level). Се користи како замена на повеќе нивоа во темата. Се користи знакот „#“ за означување на ова ниво.
  Пример: „myhome/groundfloor/#“
  Во претходниот пример знакот „#“ може да биде заменет со повеќе нивоа кои следат едно по друго, а другиот дел од темата си останува ист (Пример: „myhome/groundfloor/kitchen/light“). Клиентот кој се претплатува на овој тип тема ќе ги прима сите пораки кои се испратени на теми што започнуваат со нивоата кои се наоѓаат пред знакот „#“. Ако се користи само „#“ како тема во тој случај клиентот ќе ги добива сите пораки кои се објавени на брокерот.

Постојат теми кои се означени како специјални теми и тие започнуваат со знакот „$“. Тие теми се резервирани за внатрешни статистики на „MQTT“ брокерот. Клиентите не може да објавуваат на овие теми. Повеќето брокери го користат форматот „$SYS/“ за добивање информации. Примери на специјални теми се следните [26]:

```
$SYS/broker/clients/connected

$SYS/broker/clients/disconnected

$SYS/broker/messages/sent
```

Најдобри практики кои се користат при конструкција на темите кај „MQTT“ се следните [20]:

- Да не се користи коса црта како прв карактер во темата (Пример: /myhome/livingroom/light);
- Да не се користат празни места бидејќи често тешко може да се прочитаат;
- Најдобро е темата да е кратка и јасна;
- Најдобро е да се користат само „ASCII“ карактери;
- Во темата најдобро е да се вметне уникатен идентификатор или „ClientId“;
- Не е препорачливо клиентите да се претплатуваат на темата означена како „#“;
- Најдобро е да се користат специфични теми, повеќе отколку генерални;
- Не треба да се заборава проширувањето на темите во случај кога е потребно претплатување на теми кои во суштина се блиски по значење;

2.4.3 „Extensible Message and Presence Protocol“ (XMPP)

„XMPP“ е апликациски профил на „Extensible Markup Language“ (XML) кој овозможува размена на структурирани податоци во реално време помеѓу кои било два или повеќе мрежни ентитети [27].

Основната синтакса и семантика на „XMPP“ оригинално била развиена од страна на заедницата Џабер (Jabber) со отворен код, уште во далечната 1999 година [27]. Во 2002 година „XMPP“ работната група работела на приспособување на основниот протокол „Jabber“ за „IETF“ инстант-пораки (IM).

Во 2004 година, биле објавени документите „RFC3920“[26] и „RFC3921“[27]. Последната верзија на ваков тип документ кој дава детален опис за основите на овој протокол е „RFC6120“[28].

„XMPP“ обезбедува технологија за асинхрона, „крај-до-крај“ размена на структурирани податоци преку директни, постојани „XML“- стримови помеѓу дистрибуирана мрежа на глобални и адресабилни клиенти (кои знаат за својата достапност) и сервери. Овој архитектурен стил вклучува сеприсутно познавање на мрежната достапност и концептуално неограничен број конкурентни текови на податоци во контекст на дадена „клиент-до-сервер“ сесија или пак „сервер-до-сервер“ сесија. Затоа овој тип архитектура се именува како „Достапност на конкурентни трансакции“ (Availability for Concurrent Transactions-ACT) за да се разликува од познатата архитектура „REST“[29] кај вебот [27]. Архитектурата на „XMPP“ е многу слична на архитектурата на „email“, меѓутоа во неа се воведени неколку промени сè со цел олеснување на комуникацијата во реално време. Типовите податоци кои се пренесуваат се во структуриран формат односно „XML“.

„XMPP“ се користи за развивање „објави-претплати“ сервиси, разговор со повеќе корисници, форма за враќање и обработка, откривање сервиси, пренос на податоци во реално време, контрола на приватност и повици на оддалечени процедури.

„XMPP“ користи глобални и уникатни адреси (врз основа на „Domain Name System“ - DNS) сè со цел да ги насочува и да ги доставува податоците преку мрежата. Во основа, серверските адреси се со формат <дел на доменот> (Пример: <im.example.com>), додека сметките кои се хостирани на серверот се со формат <localpart@domainpart> (Пример: aleksandar@im.example.com, наречен „гол JID“[30]).

[26] RFC3920, Extensible Messaging and Presence Protocol (XMPP): Core, https://tools.ietf.org/html/rfc3920

[27] RFC3921, Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, https://tools.ietf.org/html/rfc3921

[28] RFC6120, Extensible Messaging and Presence Protocol (XMPP): Core, https://tools.ietf.org/html/rfc6120

[29] REST - Архитектурен стил за развивање на мрежни апликации. Идеата е наместо користење на комплексни механизми како CORBA, RPC или SOAP за поврзување на машините, се воведува користење на HTTP за да се повикуваат машините, http://rest.elkstein.org/ .

[30] „JID“ - уникатен идентификатор за ентитети во мрежата „Jabber“, https://xmpp.org/extensions/xep-0029.html#appendix-xmpp

Уредите и ресурсите кои се авторизирани за интеракција на страна на сметката имаат адреси во следниот формат <localpart@domainpart/resourcepart> (Пример: aleksandar@im.example.com/balcony, наречен „целосен JID“). Адресите кај XMPP се наречени „Jabber Ids“ или само „JID“. Тоа е така од историски причини. Форматот на адреси е прикажан во документот „RFC6122“.

„XMPP“ ја вклучува можноста ентитетите да можат да го прикажат своето присуство или достапност на другите ентитети. Со тоа корисникот кој сака да прати порака, пред да ја испрати, ќе знае дали другата страна е достапна или не (RFC6121[31]).

„XMPP“ главно се состои од стримови. Структурираните податоци се праќаат помеѓу мрежните крајни точки користејќи ги овие стримови [28]. Овие стримови се воспоставуваат помеѓу клиентот и серверот.

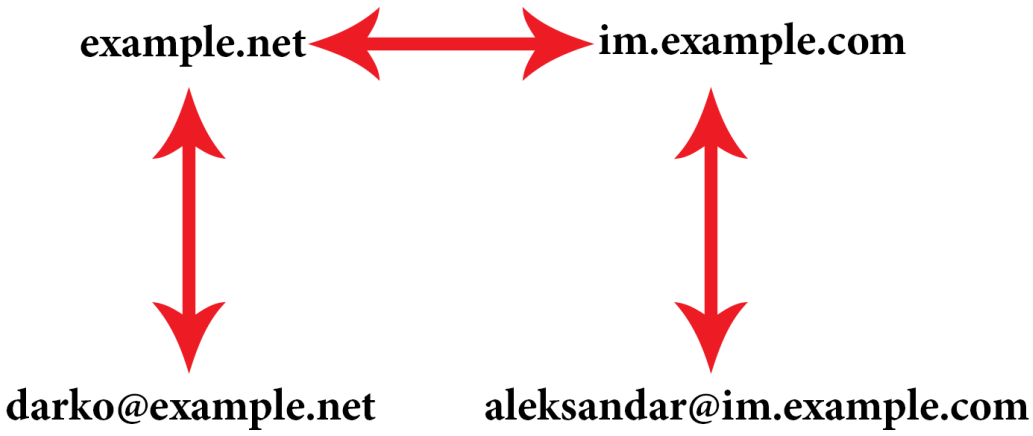
Клиент е ентитет кој воспоставува „XML“-стрим со серверот, со автентикација користејќи ги ингеренциите на регистрираниот корисник. Клиентот го користи „XMPP“ за да комуницира со серверот, другите клиенти и кои било други ентитети на мрежата, кај кои серверот е одговорен за доставување податоци до други клиенти на истиот сервер или, пак, насочување кон други оддалечени сервери. Повеќе клиенти може истовремено да се поврзат на серверот на една сметка (account), со таа разлика што сите тие во адресата се разликуваат во делот кој е наменет за ресурси (resourcepart).

Серверот е ентитет кој има одговорност да управува со „XML“-стримовите со поврзаните клиенти и со оддалечените сервери. Во случајот кога серверот управува со поврзани клиенти, тој треба да се осигури дека клиентот е автентикуван на серверот пред да добие пристап до „XMPP“-мрежата.

Структурираните податоци кои се користат кај „XMPP“ се познати како строфи (Stanzas). Сите функции на комуникацијата што „XMPP“ серверите ги обезбедуваат се остваруваат со размена на овој вид на строфи.

Во практика, XMPP се состои од мрежа на клиенти и сервери што комуницираат помеѓу себе (Слика 18). Комуникацијата помеѓу два сервера е

[31] „RFC6121“ - Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, https://tools.ietf.org/html/rfc6121



Слика 18. Дистрибуирана клиент-сервер архитектура

строго дискреciска и е прашање на локалната политика на сервисите. „Крај-до-крај“ комуникацијата логички е „peer-to-peer“ комуникација меѓутоа физички таа е „клиент-до-сервер-до-сервер-до-клиент“ комуникација.

2.4.3.1 XMPP врска

Процесот со кој клиентот остварува поврзување со серверот, разменува „XML“-строфи и ја завршува врската е даден во неколку чекори [27]:

1. Одредување на „IP“-адресата и портата за поврзување, типично врз основа на разрешување име на домен;
2. Отворање „TCP“-врска;
3. Отворање на „XML“-стрим преку „TCP“;
4. Преговарање за „TLS“ за кодирање на каналот;
5. Автентикација со користење на механизмот „SASL“ (Simple Authentication and Security Layer);
6. Поврзување ресурс на стримот;
7. Размена на неограничен број „XML“-строфи со други ентитети на мрежата;
8. Затворање XML стрим;
9. Затворање на „TCP“-конекцијата;

Кај „XMPP“ еден сервер може опционално да се поврзе со друг сервер за да овозможи комуникација помеѓу серверите или домените. За да може ова да се случи, серверите треба да преговараат за остварување на врска помеѓу нив за потоа да можат да разменуваат „XML“-строфи. Процесот за остварување врска помеѓу серверите е сличен со процесот за остварување врска помеѓу клиент и сервер. Разликата е тоа што кај врската помеѓу сервери е исфрлен чекорот 6 за поврзување ресурс на стримот.

За да може да се воспостават стримови, најпрво мора да се овозможи конекција. Кога ние пишуваме „IP“-адреса во адресното поле на пребарувачите, тие пребаруваат за да ја добијат „IP“-адресата и да можат да ја симнат страницата. На пример: една веб страница (како www.example.com) и соодветна „IP“-адреса (како 192.142.142.142) се чува како запис на „DNS“ серверот. Така, пребарувачот од „DNS“ серверот бара да му ја даде „IP“-адресата на „www.example.com“ и потоа оди на определената „IP“-адреса, ја симнува страната и ни ја прикажува. Слично е и во случајот на „XMPP“, при што се користат серверски записи (SRV records). Кога клиентот сака да се поврзе на сервер или со друг сервер, тој прво го бара серверскиот запис на серверскиот домен. Откако записот ќе биде вратен, конекцијата ќе биде воспоставена.

2.4.3.2 Креирање стримови

Откако врската е воспоставена, потребно е да се отворат стримови за двете страни, со цел да се овозможи размена на елементи (строфи). Ова се прави со праќање на <stream:stream> до серверот. Серверот потоа одговара со <stream:stream> таг. Во следните кодови се прикажани почетен стрим (I) и стрим за одговор (R):

```
I: <?xml version='1.0'?>
<stream:stream
from='aleksandar@im.example.com'
to='im.example.com'
version='1.0'
xml:lang='en'
xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.
org/streams'>


R: <?xml version='1.0'?>
<stream:stream
from='im.example.com'
id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
to='aleksandar@im.example.com'
version='1.0'
xml:lang='en'
xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.
org/streams'>
```

Откако е направено ова, стримовите се отворени и на двете страни. Заедно со тагот <stream:stream> од серверот се испраќа и тагот <stream:features> (ова се случува само еднаш, кога се воспоставуваат стримовите). Овој таг ги опфаќа карактеристиките на стримот воспоставени со серверот. Овие карактеристики

Табела 5. Почетен стрим и стрим за одговор

Почетен стрим	**Стрим за одговор**
<stream>	
	<stream>
<presence> <show/> </presence>	
<message to='foo'> <body/> </message>	
<iq to='bar' type='get'> <query/> </iq>	
	<iq from='bar' type='result'> <query/> </iq>
</stream>	
	</stream>

најчесто се однесуваат на енкрипциските и автентикациските опции кои се поддржани од стримот (На пример, дали стримот користи „TLS“ или не).

Во основа еден „XML“-стрим фукционира како контејнер за „XML“-строфи кои се испраќаат за време на сесија, а друг „XML“-стрим функционира како контејнер за „XML“-строфи кои се примени за време на сесија (Табела 5). Постојат неколку типа „XML“-строфи (Stanzas) и тоа: „message“, „presence“ и „IQ“ (Info/Query). Тие обезбедуваат три различни комуникациски механизми: механизам за испраќање на пораки „push“ (message), специјализиран механизам „објави-претплати“ за прикажување информации за достапноста на мрежата (presence), и механизам „барање-одговор“ за поструктурирана размена на податоци (iq). Подетално овие механизми ќе бидат разгледани потоа.

2.4.3.3 Автентикација

Откако е воспоставена конекцијата и стримовите се поднесени, следниот чекор е автентикацијата. „XMPP“ ги автентификува клиентите користејќи го протоколот „SASL“, а може да се користат и други механизми за автентикација. Откако клиентот се автентификува, тој се поврзува со ресурс. За да испраќаме инстант-пораки, ние се најавуваме на некој „IM“ клиент (на пример, „Pidgin“) користејќи ја корисничката сметка. За да се најавиме на друг клиент од дома не ни е потребно да креираме нова корисничка сметка, туку може да се најавиме со

истата сметка. Ова е овозможено со користење ресурси. Секојпат кога ќе се најавуваме од друг клиент, секој од нив доделува ресурс (како идентификатор). Ова се обезбедува од серверот, или од клиентот. Откако клиентот еднаш ќе се автентификува и ќе му биде доделен ресурс, тој потоа може да започне со размена на структурирани „XML“-пораки.

2.4.3.4 Видови XML-строфи (Stanzas)

Како што споменавме и претходно, постојат три вида „XML“-строфи и тоа: „presence“, „message“ и „iq“ (info/query).

1. „Presence“ (<presence>)

Како што кажува и името, со овој елемент се прикажува присуството на ентитетот. Во основа се прикажува дали клиентот со кој сакаме да пишуваме е достапен или, пак, не е достапен („online“ или „offline“) [29]. Тој е специјализиран „broadcast“ или „publish-subscribe“ механизам, со кој повеќе клиенти примаат информации (во овој случај информации за достапноста на мрежата) за некој ентитет за кој се претплатени [27]. Така, ако „user1“ сака да добива информации за „user2“ и „user3“, во тој случај тој треба да се претплати за добивање присуство за „user2“ и „user3“ [29]. Претплатата се воспоставува користејќи го тагот <presence>. Пример за тоа како user1 се претплатува на user2 е даден со следниот код:

```
<presence
from='user1@example.com'
 to='user2@example.com'
   type='subscribe'/>
```

„User2“ ја прифаќа претплатата а тоа е дадено со следниот код:

```
<presence
from='user2@example.com'
 to='user1@example.com'
   type='subscribed'/>
```

Ако клиентот прати строфа за присуство до серверот, тогаш тој ја објавува достапноста на клиентот до сите претплатници на тој клиент (механизам „publish-subscribe“). На пример ако „user1@example.com“ прати <presence/> (наједноставна форма на „presence“) до серверот, тогаш сите корисници што се претплатени на „user1@example.com“ може да видат дека корисникот е достапен. Слично на ова, ако се испрати строфа „presence“ од типот <presence type='unavailable'> до серверот, во тој случај клиентите кои се претплатени на тој корисник ќе видат дека тој корисник не е присутен. Ова е начинот на кој

клиентите кои праќаат инстант-пораки ја прикажуваат достапноста на корисниците.

Овој тип строфа се состои уште од два елемента, кои се именувани како <status> и <show>. Елементот <status> може да го содржи кој било стринг на пример:

```
<status>XMPP protocol</status>
```

Елементот <show> се користи за да се прикаже достапноста на корисникот.Тој може да ги содржи вредностите: „away“, „chat“, „dnd“, и „xa“. На пример:

```
<show>away</show>
```

Користејќи ги горните примери, целата строфа за присуство ќе изгледа вака:

```
<presence>
<show>away</show>
<status>XMPP protocol</status>
</presence>
```

2. „Message“ (<message>)

Овој елемент е „push“ механизам, при што еден ентитет праќа пораки до друг ентитет, слично како кај комуникацијата која се остварува кај „email“ [27]. Пример:

```
<message from='user1@example.com'
to='user2@example.com'
type='chat'>
<body>I am learning about XMPP</body>
</message>
```

Во тагот <body> се наоѓа содржината на пораката. Атрибутот „to“ се користи за да се означи примателот на пораката, додека „type“ се користи да се означи типот, во овој случај тоа е „chat“.

3. „IQ“ (<iq>)

„IQ“ (Info/Query) е „барање-одговор“ механизам, многу сличен на „HTTP“ [27]. Семантиката на „IQ“ им овозможува на ентитетите да бараат информации, и да добиваат одговор назад од други ентитети.

Кога се најавуваме на „IM“ клиент може да ја видиме листата од сите корисници со кои може да разговараме. Оваа листа уште е позната како список (roster). За добивање на списокот со сите корисници, до серверот се испраќа следната „IQ“ строфа:

```
<iq from='user1@example.com'
type='get'
id='xyz123'>
```

```
<query xmlns='jabber:iq:roster'/>
</iq>
```

Кога погорната строфа е примена од серверот тој одговара со списокот на пријатели на „user1@example.com“:

```
<iq to='user1@example.com'
type='result'
id='xyz123'>
<query xmlns='jabber:iq:roster'>
<item jid='aleksandar@example.com' name='Aleksandar'/>
<item jid='darko@example.com' name='Darko'/>
</query>
</iq>
```

Атрибутите „id“ и „type“ кај „IQ“ се задолжителни. Атрибутот „type“ може да има неколку вредности и тоа: „get“, „set“, „result“ и „error“. Вредноста „get“ претставува барање за информација. Вредноста „set“ се користи за обезбедување соодветни податоци, за поставување нови вредности или за замена на постојни вредности. Вредноста „result“ е одговор на успешно „get“ или „set“ барање. Во случај на грешка, се јавува вредноста „error“ како резултат на обработката или на доставата на претходно испратено „get“ или „set“.

2.4.3.5 Исклучување (Disconnection)

Корисниците се одјавуваат од „IM“ клиентите со кликање на копчето за одјава. Во суштина, тие праќаат строфа „<presence>“ за присуство до серверот, со што му кажуваат на серверот дека веќе не се достапни. Така, другите корисници на чиј список сме ќе бидат известени дека не сме достапни. По ова стримот се прекинува користејќи го тагот </stream:stream>.

3. Материјали и методи на истражување

Мерењата за потрошувачката на енергија ќе бидат извршени за трите протоколи кои се наоѓаат на апликациско ниво кај Интернетот на нештата и тоа: „CoAP“, „MQTT“ и „XMPP“. Овие протоколи имаат различна архитектура која поединечно беше разгледана во претходните поглавја.

Во нашите примери ќе го користиме оперативниот систем „Contiki“. Тоа е оперативен систем за „IoT“ [30]. Развиен е од Адам Данкелс (Adam Dunkels), а понатаму бил доразвиван од големи светски тимови на развивачи од „Texas Instruments“, „Atmel“, „Cisco“, „Sensinode“ и други. Тој е имплементиран во програмскиот јазик „C“ и е со отворен код. Соодветен е на голем број архитектури на микроконтролери вклучувајќи ги овде „Texas Instruments MSP430“ и „Atmel AVR“ [31]. Наменет е за микроконтролери и сензорски уреди што имаат ограничена меморија, моќ, перформанси и пропусен опсег. Тоа се уреди кои најчесто се со осумбитни микроконтролери и имаат околу 100 kB ROM и 20 kB RAM меморија [31]. „Contiki“ се состои од јадро кое е управувано од настани, на чиј врв се наоѓаат апликативните програми кои се вчитуваат и се превчитуваат со тек на време [32]. Процесите кај „Contiki“ користат едноставни нишки наречени „protothreads“, кои обезбедуваат линеарен програмирачки стил со нишки на врвот на јадрото кое е управувано од настани. „Contiki“ исто така поддржува и опционални, превентивни процеси со повеќе нишки (multi-threading) и комуникација помеѓу процесите. Типичниот „Contiki“ систем има меморија од редот на килобајти (kB), потрошувачка од редот на миливати (mW), брзина на обработка која се мери во мегахерци (MHz) и комуникациски пропусен опсег од редот на стотици килобити во секунда (kbit/s). Ваквите системи вклучуваат многу видови вградени системи и стари осумбитни компјутери. Неколку „Contiki“ механизми биле објавени како посебни пакети со отворен код и имале големо влијание врз индустријата. Во 2001 година бил објавен вградениот „IP“-стек „uIP“ и тој денес се користи од стотици компании во системи како товарни бродови, сателити, опрема за дупчење нафта и слично. Првата библиотека за програмирање на „Contiki“ била објавена во 2005 година и најпрво се користела кај дигиталните телевизиски декодери и кај безжичните сензори. Покрај „TCP/IP“-стекот „uIP“ кој го користи протоколот „IPv4“, „Contiki“ обезбедува уште два мрежни механизми како: „uIPv6“, кој го користи протоколот „IPv6“ и „Rime“ стекот

кој е збир на едноставни мрежни протоколи кои се наменети за безжични мрежи со мала моќ. „IPv6“ стекот бил развиен од „Cisco“. Тој ги содржи: „RPL“ рутирачкиот протокол за „IPv6“ мрежи со мала моќ и големи загуби, „6LoWPAN“ компресија на хедерот и адаптациски слој за „IEEE 802.15.4“ линкови. Комуникацискиот стек „Rime“ обезбедува збир од едноставни комуникациски примитиви кои се движат од анонимни локални емитувања па сè до сигурни проточни мрежи [33]. Овој комуникациски стек се користи кога „IPv4“ и „IPv6“ обезбедуваат големи преоптоварувања на мрежата. Протоколите кај овој комуникациски стек се подредени во слоеви, при што посложените протоколи се имплементирани со користење на помалку сложените протоколи. Слоевите се дизајнирани така за да бидат екстремно едноставни и во смисла на интерфејс, и во смисла на имплементација. Секој слој додава посебно заглавје (header) на појдовните пораки. Индивидуалните заглавја кај протоколот „Rime“ се многу мали, обично по неколку бајти. Овој комуникациски стек најмногу се користи кај безжичните сензорски мрежни уреди. Апликациите или протоколите кои се наоѓаат на врвот на овој стек ги користат комуникациските примитиви. „Rime“ ги поддржува комуникациските примитиви „single-hop“ и „multi-hop“. Примитивите од типот „multi-hop“ не определуваат како се рутираат пакетите преку мрежата. Наместо тоа, кога се испраќа пакет низ мрежата, се повикува апликацискиот или горниот слој на секој јазол за да го избере следниот чекор до соседот. Со ова се отвора можноста за имплементација на произволни протоколи за рутирање на врвот на овие примитиви.

За истражувањето за потрошувачката на енергија во оваа магистерска работа беше користен комуникацискиот стек „Rime“. Тој беше користен бидејќи обезбедува архитектура во слоеви која ја поедноставува имплементацијата на комуникациските протоколи со мало зголемување на барањата за ресурси и е доста користен кај безжичните сензорски мрежи.

За истражувањата во оваа магистерска работа беше користен оперативниот систем „Contiki 3.0“ за Интернет на нештата (развојна околина Instant Contiki 3.0). Тоа е последната верзија на овој оперативен систем која официјално беше претставена на 25.8.2015 година [30]. Оваа верзија е голем исчекор за разлика од претходните верзии на овој оперативен систем нумерирани под „2.X“ (2.7 е верзијата пред 3.0). Таа има поддршка за: нов

хардверски збир на неколку нови протоколи, подобрувања кај мрежните протоколи, заедно со голем број општи подобрувања на стабилноста [34]. Нови хардверски елементи кои се додадени се: „Texas Instruments Sensortag“ и „Zolertia ReMote“. Додаден е збирот протоколи „IP64/NAT64/DNS64“, со што се овозможува поврзување на „Contiki IPv6“ мрежи директно со „IPv4“ мрежи, како што е Интернет, без потреба за додавање специјални прокси за преведување на протоколите. Протоколите работат директно на „IP“ слојот за да создадат преведувања помеѓу мрежата и Интернет. Заедно со тоа, додаден е и драјвер за поддршка на чипот „ENC26j80 Ethernet“ за да може лесно да се поврзат „Contiki“ мрежи со „Ethernet“ мрежи. За поврзување на апликациско ниво обезбедена е поддршка за протоколот „MQTT“ и ажурирана е верзијата на модулот „CoAP“ (протоколи кои ќе бидат тема на истражување во оваа магистерска работа). Додадена е поддршка и за нов „HTTP socket“ модул. Новите „API“ за „TCP“ и „UDP“ сокети овозможуваат значително полесно мрежно програмирање отколку во минатото.

Новиот системски модул обезбедува полесно и селективно користење на делови од кодот. Наместо да се користат макроа „C“ како и „#defines“ за да се избере дали да се користи „IPv4“ или „IPv6“ стек, новиот модул овозможува користење на „Makefile[32]“ нагодувања, што го прави компајлирањето поефикасно, а кодот полесен за читање.

Радио „API“ е надградено така што обезбедува подобро користење на радиото од протоколите. Во претходната верзија недостаток било тоа што „API“ не обезбедувало поддршка за поставување радиоканал. Тоа е сега дел од новото „API“.

Во последната верзија избришани биле и повеќе хардверски платформи кои веќе не се користеле, како и голем број примери кои не биле релевантни. Додадени биле и нови регресиски тестови за обезбедување континуирана стабилност на системот [34].

„Instant Contiki 3.0“ обезбедува развојна околина која ги содржи сите алатки, компајлери и симулатори за развој на апликации за „IoT“ [34]. Таа е

[32] „Makefile“ – фајл кај „Contiki“ програмите со кој се дефинираат основни параметри што се користат при компајлирање, http://anrg.usc.edu/contiki/index.php/Contiki_build_system

„Ubuntu Linux“ виртуелна машина со големина од 3.2 GB [35]. За целите на ова истражување, виртуелната машина е извршена во „VMWare 10 Virtual Machine Player“.

За истражувањата за енергетската потрошувачка на протоколите на апликациско ниво кај „IoT“ беше користен мрежниот симулатор „Cooja“. Тоа е апликација базирана на „Java“, со графички кориснички интерфејс („GUI“ базирано на стандардната алатка „Java Swing“) [36]. „Cooja“ исто така поддржува симулација на радиомедиум и интеграција со надворешни алатки за да се обезбедат дополнителни особини на апликацијата. „Cooja“ овозможува да бидат симулирани големи и мали мрежи на „Contiki“ сензорни модули. Модулите може да бидат емулирани на хардверско ниво, што е побавно, но секогаш прецизно испитување на особините на системот или на помалку деталното ниво, кое е побрзо и овозможува симулација на поголеми мрежи. Оваа алатка има два софтверски емулаторски пакети и тоа: „Avrora“ и „MSPSim“. „Cooja“ ја користи „Avrora“ за емулирање на уреди базирани на „Atmel-AVR“ и „MSPSim“ за емулирање на уреди базирани на „TI MSP430“. Повеќето платформи кај „IoT“ имаат микроконтролери од типот „MSP430“ [36]. Тоа е причината поради која „MSPSim“ е најчесто користениот софтверски пакет за симулација на безжични сензорски мрежи. Во ова истражување е користен емулаторскиот пакет „MSPSim“ бидејќи платформата која се користи е „MSP430“ базиран уред (Z1). „Cooja“ може да емулира повеќе платформи како: „TelosB/SkyMote“, „Zolertia Z1“, „Wismote“, „ESB“, „Micaz“ и други. Таа е многу корисна алатка за развојот на апликации за оперативниот систем „Contiki“ и за нивно дебагирање. „Cooja“ им овозможува на развивачите на софтвер да може да ги тестираат нивните кодови пред да бидат извршени на реални уреди. Оваа алатка овозможува и проценка на потрошувачката на енергија кај јазлите во симулацијата. Тоа е основната цел на оваа магистерска работа. „Cooja“ овозможува прикажување и на радио трансмисиите, како и на примањата.

Во „Instant Contiki“ алатката се наоѓа во директориумот:

```
/home/user/contiki-3.0/tools
```

„Cooja“ се стартува преку терминал со следната команда:

```
cd  /home/user/contiki-3.0/tools/cooja
```
(промена на работниот директориум)

```
ant run cooja
```
(стартување на „Cooja“)

При првичното стартување на „Cooja“ може да се јави грешка која укажува на тоа дека директориумот „Mspsim“ е празен. „Mspsim“ е емулатор на сензорните јазли кои може да бидат од повеќе типови. За надминување на проблемот, најпрвин е потребно преку терминал да го промениме работниот директориум со следната команда:

```
cd /home/user/contiki-3.0/tools/mspsim,
```

а потоа е потребно да ги извршиме и следните команди со кои ќе се овозможи копирање на потребните „mspsim“ фајлови од основниот директориум на „ Contiki OS“:

```
git submodule init
git submodule update - -recursive
git clone https://github.com/contiki-os/mspsim.git
```

Со повторно стартување на „Cooja“, ќе забележиме дека сега нема никаков проблем при нејзиното стартување. Потоа може да се продолжи со изведување на симулацијата.

За изведување на симулациите во „Cooja“ беше користен безжичниот сензорен модул „Z1 Zolertia“ [37]. Овој модул им овозможува на развивачите на апликации за безжичните сензорски мрежи да ги развиваат и да ги тестираат апликациите со најдобар сооднос помеѓу времето на развивање и хардверската флексибилност. Тој се користи за извршување разни видови на апликации како: здравствен надзор, детектори за итни случаи, безбедносни и направи за спасување, мерење на енергијата, следење на состојбата во земјоделството и други „IoT“ апликации. „Z1“ е опремен со втората генерација на ниско моќен микроконтролер „MSP430F2617“, кој поседува „16-bit RISC CPU“ со 16 MHz брзина на часовникот, 8 KB RAM и 92 KB флеш-меморија. При изведување на симулацијата во „Cooja“, сензорните јазли имаат 8 MHz брзина на часовникот, што го земаме во предвид во пресметките за потрошувачката на енергија. Во овој модул е вклучен и „CC2420 transceiver“, кој е „802.15.4“ соодветен и оперира на 2.4 GHz со ефективна податочна рата од 250Kbps. Тој има 52-пински конектор за проширување. Поседува и дигитален акцелерометар со 3 оски и дигитален температурен сензор со мала моќ со точност од ± 0.5 (во рангот на температури од 25 ºC до 85 ºC) [37]. Уште има и опционална надворешна антена преку „U.FL“ конектор и „Micro-USB“ конектор за напојување и дебагирање. Модулот „Z1“ може да се користи во индустриски опсег на температури (-40ºC-85ºC). Тој треба

Табела 6. Приближна потрошувачка на струја на интегрирани кола кај „Z1“

Интегрирано коло	**Работен опсег**	**Потрошувачка на струја**	**Забелешки**
MSP430f2617	1.8V до 3.6V	0.1µA	OFF Mode
		0.5µA	Standby Mode
		0.5mA	Active Mode @1MHz
		< 10mA	Active Mode @16MHz
CC2420	2.1V до 3.6V	<1µA	OFF Mode
		20µA	Power Down
		426µA	IDLE Mode
		18.8mA	RX Mode
		17.4mA	TX Mode @ 0dBm
ADXL345	1.8V до 3.6V	0.1µA	Standby
		40uA до 145uA	Active Mode
M25P16	2.7V до 3.6V	1µA	Deep Power Down
		4mA до 15mA	Active Mode
TMP102	1.4V до 3.6V	1µA	Shutdown Mode
		15µA	Active Mode

да биде приклучен на напојување од 3V. Во Табела 6 е даден краток опис и приближната потрошувачка на струја на интегрираните кола во модулот „Z1“ [37]. Овие вредности се користени за пресметка на просечната потрошувачка на енергија во зависност од тоа колку долго секое интегрирано коло работи во секој режим (даден во колоната за забелешки во Табела 6). Од сите податоци за интегрираните кола кои се дадени во Табела 6 најважни се „MSP430f2617“ и „CC2420“ односно најважна е потрошувачката на енергија на микроконтролер единицата и на радио предавателот.

Модулот „Z1“ може да се напојува на неколку начини, во зависност од потребната апликација. Двата најважни начини на напојувања се [37]:

- Батерии: Користење на „2xAA“ батерии или „AAA“ ќелии или „1xCR2032 ќелија“ во сместувачот за батерии.
- „USB“: Поврзување на „Z1“ со помош на „MicroUSB“ кабелот.

Двата типа на напојување може да постојат во исто време, но се препорачува батериите да бидат отстранети од сместувачот ако модулот „Z1“ се користи подолго време (На пример, во случајот кога се врши развивање и тестирање на апликациите директно на модулот „Z1“). Податоците за батеријата ќе бидат

користени за пресметките за тоа колкав ќе биде животниот век на батеријата на модулот „Z1“ во случаите кога кај дадените примери на апликации се користат протоколите на апликациско ниво кај „IoT“.

За протоколот „CoAP“, „Z1“ ќе го користиме и на клиентската и на серверската страна. Ќе ја мериме потрошувачката на енергија и на клиентската, и на серверската страна, и тоа во случаи кога имаме од еден до пет клиенти кои пристапуваат до серверскиот сензорен модул, или кога немаме ниту еден клиент. Притоа ќе утврдиме како се менува потрошувачката во овој случај. За другите два протокола „MQTT“ и „XMPP“, „Z1“ ќе го користиме само на клиентската страна бидејќи имплементацијата на брокерот кај „MQTT“ и серверот кај „XMPP“ е ориентирана кон веб. Комуникацијата со нив се остварува со користење на „RPL border“ рутер, чија основна цел е да поврзе една мрежа со друга [38].

Во анализата на протоколот „CoAP“, комуникацијата се остварува помеѓу клиент и сервер. Најпрво се креира врска со серверот а потоа периодично се праќаат податоци до кој било од предефинираните „URL“ за откривање сервиси. Селектирањето се врши по случаен избор со користење на „random“ функција. Во „Instant Contiki 3.0“ оваа апликација се наоѓа во директориумот со примери:

```
<CONTIKI_HOME>/examples/rest-example/
```

Во овој случај „CONTIKI_HOME“ е домашниот директориум на Contiki или во овој случај тоа е директориумот со патека:

```
/home/user/contiki-3.0/examples/rest-example
```

Со „rest-server-example.c“ е имплементиран „RESTful“ сервер кој покажува како треба да се користи „REST“ слојот за да се развиваат апликации на серверската страна (кои е можно да се извршуваат преку „CoAP“ или преку „HTTP“). За да се користи „CoAP“ како основен протокол на апликациско ниво, потребно е во апликацискиот „Makefile“ со патека „rest-example/Makefile“ да се додаде следниот код:

```
WITH_COAP = 1
```

Ако не се додаде овој код, тогаш како протокол на апликациско ниво ќе се користи „HTTP“. „REST“ серверот ги има следните ресурси: „light“, „led“, „toggle“ и „discover“ [39]. Секој ресурс си има функција која управува со него (resource handler function) која се повикува секогаш кога клиентот го бара тој ресурс. Оваа

функција го враќа одговорот назад до клиентот. Пример за функција која управува со ресурсот „light“ е даден со следниот код:

```
RESOURCE(light, METHOD_GET, "light");
void
light_handler(REQUEST* request, RESPONSE* response)
{
  read_light_sensor(&light_photosynthetic, &light_solar);
  sprintf(temp,"%u;%u", light_photosynthetic, light_solar);
  char etag[4] = "ABCD";
  rest_set_header_content_type(response, TEXT_PLAIN);
  rest_set_header_etag(response, etag, sizeof(etag));
  rest_set_response_payload(response, temp, strlen(temp));
}
```

Со „coap-client-example.c“ имплементиран е „CoAP“ клиент. „CoAP“ клиентот воспоставува врска со серверот на „CoAP“ портата „61616“ и го поставува тајмерот (e_timer) на одредена вредност [39]. Секогаш кога вредноста на тајмерот истекува, се повикува функцијата „send_data(void)“. Кога ќе се добие одговорот на барањето од серверот, се повикува функцијата „handle_incoming_data()“. Тоа е дадено со следниот код:

```
etimer_set(&et, 5 * CLOCK_SECOND);
  while(1)
{
    PROCESS_YIELD();
    if (etimer_expired(&et)) {
      send_data();
      etimer_reset(&et);
    } else if (ev == tcpip_event) {
      handle_incoming_data();
    }
  }
```

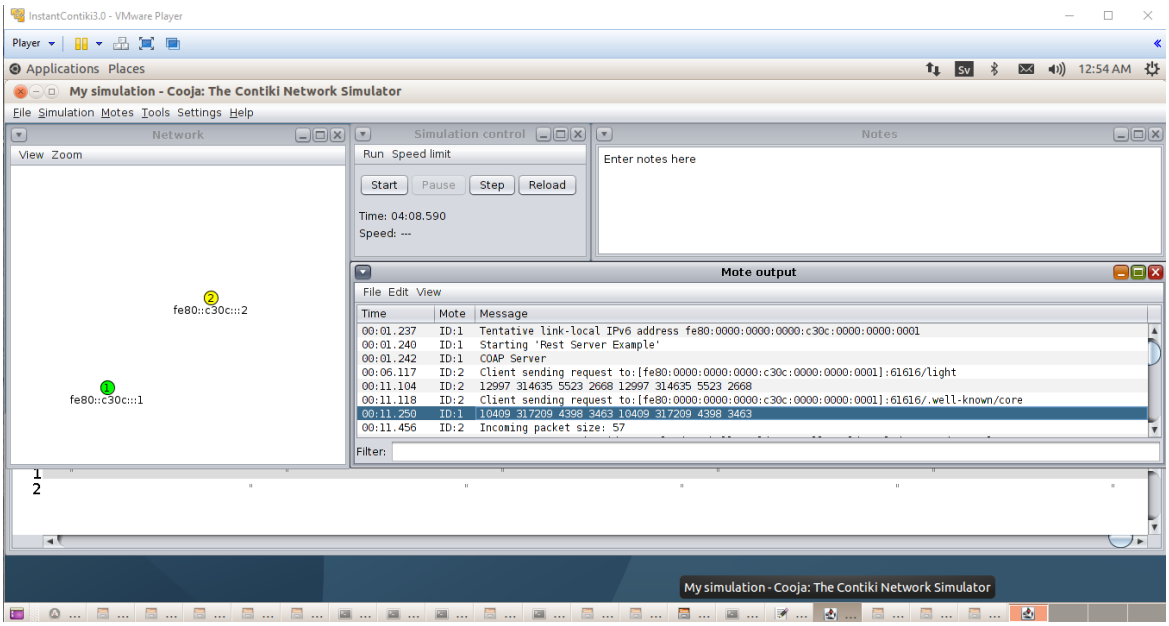
„CoAP“ клиентот извршува тајмер кој кога се ресетира, клиентот случајно селектира „service_id (resource)“ користејќи ја функцијата „random_rand()“ и го

праќа барањето до „REST“ серверот користејќи ја функцијата „send_data(void)“. Тоа е дадено со следниот код:

```
int data_size = 0;
    int service_id = random_rand() % NUMBER_OF_URLS;
    coap_packet_t* request =
(coap_packet_t*)allocate_buffer(sizeof(coap_packet_t));
    init_packet(request);
    coap_set_method(request, COAP_GET);
    request->tid = xact_id++;
    request->type = MESSAGE_TYPE_CON;
    coap_set_header_uri(request,
service_urls[service_id]);
```

Кога серверот го враќа одговорот назад до клиентот, се извршува функцијата „handle_incoming_data()“ која го зема пакетот, ја парсира пораката и го печати вратениот одговор (payload). Тоа е прикажано со следниот код:

```
static void
handle_incoming_data()
{
  PRINTF("Incoming packet size: %u \n", (uint16_t)uip_datalen());
  if (init_buffer(COAP_DATA_BUFF_SIZE)) {
    if (uip_newdata()) {
      coap_packet_t* response =
(coap_packet_t*)allocate_buffer(sizeof(coap_packet_t));
      if (response) {
        parse_message(response, uip_appdata, uip_datalen());
        response_handler(response);
      }
    }
    delete_buffer();
  }
}
```



Слика 19. Симулација на „CoAP“ сервер со 1 клиент во „Cooja“

Од функцијата може да се види дека се печати и големината на пакетот кој е вратен со делот од кодот:

```
PRINTF("Incoming packet size: %u \n", (uint16_t)uip_datalen());
```

Овие излезни вредности се печатат во делот за излез на модулот („Z1“ во нашите примери) кој се користи при извршување на симулацијата во симулаторот „Cooja“ за безжични сензорски мрежи.

На Слика 19 е прикажана симулација во „Cooja“ на „CoAP“ клиент со „CoAP“ сервер. Клиентот периодично пристапува до ресурсите на серверот и ја печати вратената вредност во зависност од тоа кој ресурс е селектиран. Може да се види дека апликацијата е извршена, а резултатите од апликацијата се прикажани во прозорецот „Mote output“ на „Cooja“. Во овој прозорец се прикажуваат и вредностите за отчукувањата на часовникот на микроконтролерот во различни фази. Овие вредности потоа ќе бидат користени за пресметка на просечната потрошувачка на енергија на протоколите на апликациско ниво. На сликата вредностите во „Mote output“ се селектирани со сина линија.

За протоколот „CoAP“ симулацијата ќе биде изведена во неколку сценарија. Првото сценарио е како што е дадено на Слика 19 (1 клиент пристапува на 1 сервер). Во другите сценарија ќе бидат користени 2-5 клиенти

или, пак, состојба во која нема ниту еден клиент. Основната цел е да се утврди просечната потрошувачка на енергија на серверот во случаи кога има само 1 клиент и во случаи кога има повеќе клиенти. Со тоа треба да се утврди менувањето на потрошувачката со зголемувањето на клиентите.

За сценариото кај протоколот „MQTT“ е користена клиентска библиотека на „MQTT“ за оперативниот систем „Contiki“ [40]. Во директориумот со апликации во „Instant Contiki“ (/home/user/contiki-3.0/apps) потребно е во новокреиран директориум (во нашиот пример под име „mqtt-service“) да ги додадеме сите фајлови од користената клиентска библиотека, освен фајлот „example.c“ кој што треба да го додадеме во новокреиран директориум (во овој случај под име „mqtt“) во директориумот со примери (examples/mqtt). Во овој директориум потребно е уште да го додадеме „Makefile“, кој ги содржи основните поставки на апликацијата кои се користат при нејзиното компајлирање. Како поставките во овој дел се дефинираат: името на проектот, потребните апликации што се користат, се дефинира основниот директориум на „Contiki“ и на „Makefile.include“. Во ова сценарио „Makefile“ го содржи следниот код:

```
CONTIKI_PROJECT = mqtt
all: $(CONTIKI_PROJECT)
CONTIKI = ../..
APPS = mqtt-service powertrace
TARGET = z1
include ../../Makefile.include
```

Од кодот може да се види дека се врши лоцирање на претходно креираниот директориум под име „mqtt-service“, кој содржи помошни библиотеки со основните функции на протоколот „MQTT“ кои ни се потребни при извршување на апликацијата. Во овој дел е додаден и друг апликациски дел именуван како „powertrace“. Тој ги содржи потребните функции кои ни се потребни за проценка на просечната потрошувачка на енергија на сите протоколи на апликациско ниво. Тој ќе биде користен во сите сценарија како помошна алатка за добивање податоци кои ќе бидат користени во пресметките за потрошувачката. Во делот од кодот означен како цел (TARGET) за извршување на апликацијата е додаден сензорскиот модул „Z1 Zolertia“. Овој

сензорски модул исто така е користен во сите сценарија како што беше напоменато и претходно.

Следно, потребно е да креираме нова симулација. Тоа се прави со следната команда во симулаторот „Cooja“:

```
File -> New Simulation
```

Со тоа ќе се отвори прозорец во кој треба да го напишеме името на симулацијата. Следно, потребно е да додадеме нов јазол во симулацијата со помош на командата:

```
Motes->Add Motes->Create new mote types->Z1 mote
```

Во делот „Contiki process/firmware“ го избираме фајлот „example.c“ што се наоѓа во директориумот со примери кој беше креиран претходно (examples/mqtt). Со кликнување на „Compile“ се врши компајлирање на дадениот код. Откако ќе заврши компајлирањето, се оди на „Create“, со што јазолот се додава во „Network“ делот на симулаторот.

За поврзување на јазолот „Z1“ со брокер (во нашето сценарио користен е брокер „Mosquitto“) кој е инсталиран, потребен е „RPL border router“, кој се користи за поврзување на „WSN“ со други мрежи, а во овој случај со брокер кој е инсталиран на локалната машина.
Кодот за „RPL border“ рутерот се наоѓа на следната локација:

```
/home/user/contiki-3.0/examples/ipv6/rpl-border-router
```

Потребно е со терминал да го смениме основниот директориум на дадената патека и да ја извршиме командата:

```
make TARGET=z1
```

По извршување на командата во истиот директориум се креира нов фајл под име „border-router.z1“. Креираме нов јазол во „Cooja“ исто од тип „Z1“ и како „Process/Firmware“ се избира претходно креираниот фајл (border-router.z1). Со тоа безжичната сензорска мрежа е комплетирана.

На локалната машина потребно е да има инсталирано брокер кој ќе се користи за примање и праќање на пораките кај „MQTT“. За сценариото на „MQTT“ е користен брокер „Mosquitto“ иако може да се користат и други кои беа споменати претходно [24]. Тој е брокер со отворен код кој го има имплементирано механизамот на протоколот „MQTT“ со „TLS“ поддршка и може да работи на „Windows“, „Linux“ и „MacOS“. Тој може да се инсталира преку

„Ubuntu Software Center“ во „Instant Contiki 3.0“, со едноставно внесување текст „Mosquitto“ во полето за пребарување. По пронаоѓањето, тој се инсталира со кликнување на „Install“. Постои и можност брокерот да се симне од неговата официјална страница, при што има можност да се избере инсталациски фајл според оперативниот систем на кој ќе биде користен [24]. За ова сценарио е користена верзијата „mosquitto 0.15-2ubuntu1“ на „Mosquitto“ која обезбедува поддршка за верзијата „3.1“ на овој протокол. Сите акции за овој брокер (објавување/претплата) се извршуваат преку терминал.

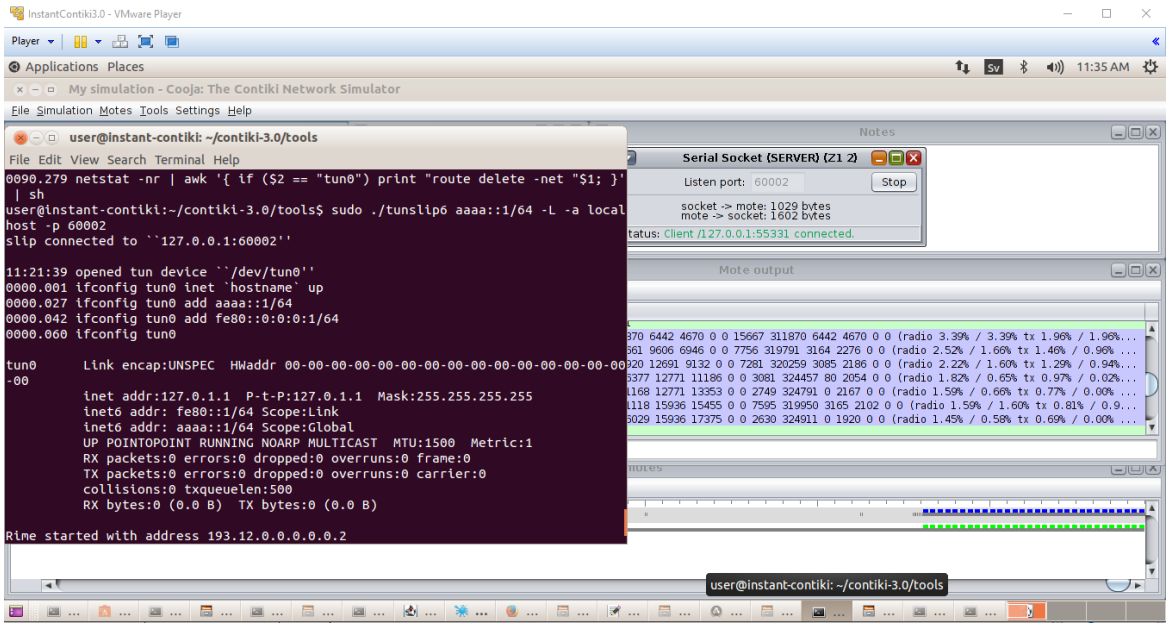
Следно, потребно е да ја овозможиме комуникацијата помеѓу „border-router“ и брокерот „Mosquitto“ кој е локално инсталиран. Тоа се прави на следниот начин:

- Се кликнува на јазолот со „border-router“ и се избира „Mote tools for Z1 2“, а од следното мени се избира „Serial Socket (SERVER)“. Се избира портата 60002 и се кликнува на „Start“.
- За остварување на комуникацијата, се користи алатката „tunslip6“. Меѓутоа, претходно треба да се изврши нејзиниот код. Преку терминал го променуваме основниот директориум во:

  `/home/user/contiki-3.0/tools`

  Потоа ја извршуваме командата:

  `make tunslip6`



Слика 20. Воспоставена конекција со помош на „tunslip6 bridge“ на порта 60002

- Сега веќе може да ја воспоставиме врската. Додека сè уште се наоѓаме во директориумот со алатки, преку терминалот ја извршуваме командата:

```
sudo ./tunslip6 aaaa::1/64 -L -a localhost -p 60002
```

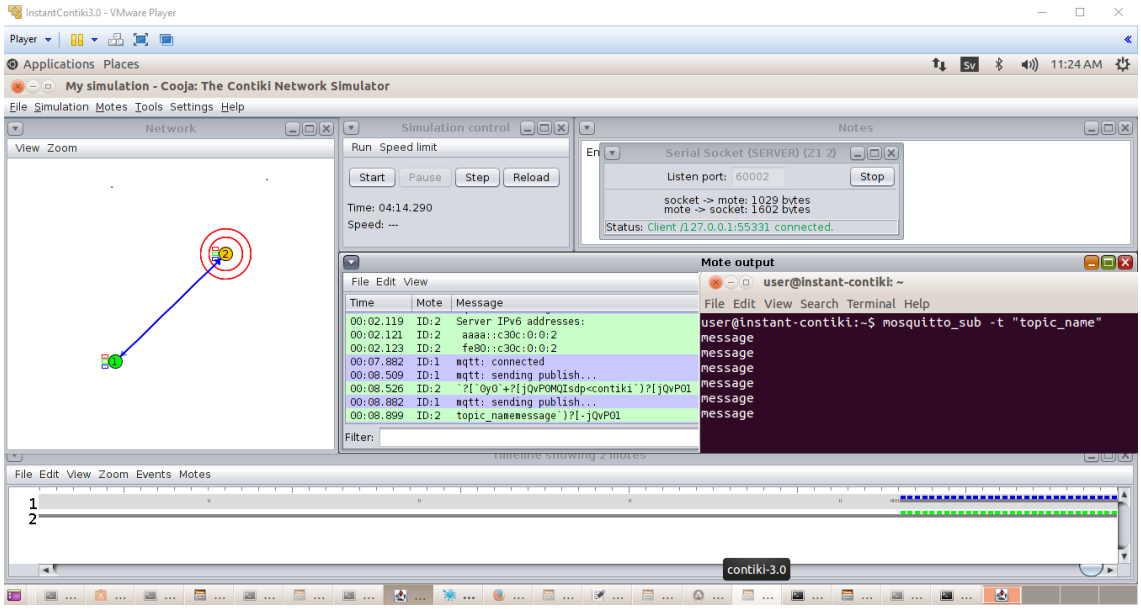
Ако врската е успешна, во терминалот ќе се прикаже код како на Слика 20, а додека во „Cooja“ во прозорецот „Serial Socket (SERVER)“ треба да стои следниот статус „Client/127.0.0.1 connected“.

Следно, потребно е да го стартуваме брокерот „Mosquitto“ и да се претплатиме на некоја тема (topic). Во ова сценарио се претплатуваме на темата „topic_name“ (Слика 21). Тоа го правиме од терминал со извршување на следната команда:

```
mosquitto_sub –t “topic_name”
```

Со ова брокерот ќе слуша на оваа тема и сите пораки кои ќе бидат објавени на неа ќе бидат препратени до „MQTT“ клиентите кои се имаат претплатено на таа тема (topic_name). Во овој случај сите пораки кои ќе бидат пратени до брокерот од објавувачите ќе бидат испечатени во терминалот на претплатниците.

Програмата која ќе се користи како пример за „MQTT“ (example.c) е напишана во програмскиот јазик „C“ како и сите други програми за „Contiki OS“. Како што беше споменато претходно, оваа програма се извршува на јазолот во „Cooja“ кој беше креиран со нејзино избирање. Со нејзиното извршување, односно со извршување на симулацијата во Cooja се врши објавување пораки на темата „topic_name“.



Слика 21. Извршување на симулацијата (конектирање на „MQTT“) и објавување пораки во терминалот

За објавување пораки во пример-програмата се користи следната функција:

```
mqtt_publish("topic_name", "message", 0, 1);
```

Значи извршениот код на јазолот „Z1“ најпрво се конектира на брокерот „Mosquitto“, кој работи локално на машината (localhost) и на дадената тема со име „topic_name“ објавува порака „message“. Излезните вредности може да се видат во терминалот со кој се врши претплата на дадената тема. Во овој случај 6 пораки со текст „message“ се прикажуваат во терминалот со кој сме ја извршиле претплатата на темата (Слика 21). Може да се види уште и извршувањето на симулацијата. Во излезот за јазолот во „Cooja“ може да се види како клиентот се поврзува на брокерот и објавува пораки. Во овој дел исто така се печатат вредностите за отчукувањата на часовникот за сите состојби во кои може да се најде јазолот „Z1“. Овие вредности потоа ги користиме за пресметување на потрошувачката на енергија.

За протоколот „XMPP“ се користи сценарио во кое се утврдува потрошувачката на енергија само на „XMPP“ клиент. За таа цел ја користиме имплементацијата на „XMPP“ клиент од страна на Клаук (Klauck) и Кирш (Kirsche) [41]. Нивниот код е базиран на прототипот „Contiki-XMPP 0.1“, кој бил развиен од Бејл (Bail) и Хорнсби (Hornsby) [42]. Клаук и Кирш ја зголемиле мемориската ефикасност на протоколот „XMPP“, а додале и некои други особини како: поддршката за „IPv6“, оптимизација на протокот на пораките, намалена дефиниција на функциите, проширено „API“ и делумна поддршка за „MUC“. Со нивниот код имплементирале само „XMPP“ клиент за оперативниот систем „Contiki“ за Интернет на нештата. Потребен е надворешен „XMPP“ сервер со кој се овозможува комуникацијата кај „XMPP“. Во сценариото за „XMPP“ е користен „Prosody XMPP 0.9.10“ како надворешен сервер [43]. Како надворешен „IM“ клиент се користи „Pidgin“ [44]. Серверот „Prosody“ и „IM“-клиентот „Pidgin“ се инсталирани користејќи ги стандардните пакети на „Instant Contiki 3.0“ преку делот „Ubuntu Software Centre“. Сите истражувања и тестирања се правени користејќи ги токму овие апликации кои претходно мора да бидат инсталирани пред да се изврши симулацијата во „Cooja“.

За ова сценарио на „XMPP“ серверот најпрво потребно е да се креираат кориснички сметки („123@localhost“ и „node_xmpp@localhost“) заедно со лозинки. За да може да се креираат нови кориснички сметки потребни се одредени

нагодувања на серверот во конфигурацискиот документ на „Prosody“ (со патека „/etc/prosody/prosody.cfg.lua“). Ако документот не може да се отвори, потребно е да се обезбеди дозвола за негово менување (користејќи ја функцијата „chmod“). За да се овозможи регистрација на нови корисници, потребни се следните редови код да бидат додадени во конфигурацискиот документ:

```
VirtualHost "localhost"
allow_registration = true;
```

Постојат и други нагодувања кои може да се направат преку овој документ како: нагодувања за криптирање и безбедност, сесии и ресурси, генерални нагодувања на серверот, мрежни нагодувања и други.

„Prosody“ има алатка која се извршува од командна линија наречена „prosodyctl“. Оваа алатка се користи за регистрација на нови корисници преку командна линија со извршување на следниот код за додавање корисник „123@localhost“:

```
prosodyctl adduser 123@example.com
```

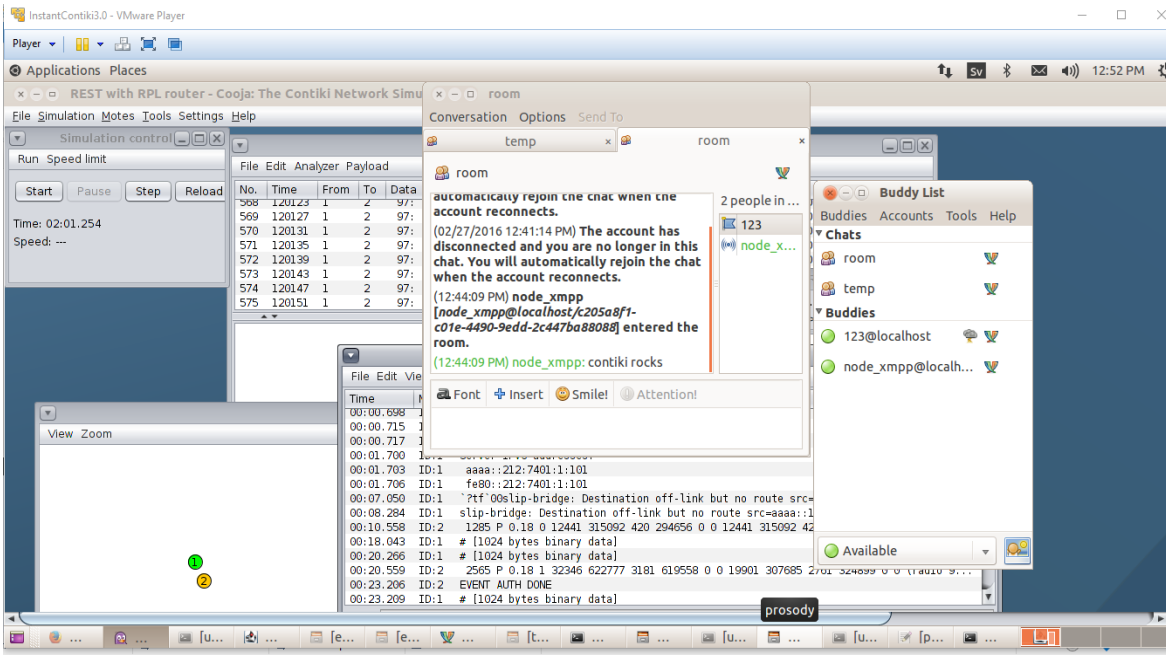
По извршување на командата серверот ќе побара од нас да внесеме и лозинка за најава. Понатаму потребно е да се најавиме со овие кориснички сметки на клиентот „Pidgin“. Корисничката сметка под името „node_xmpp@localhost“ ја правиме недостапна, а со корисничката сметка „123@localhost“ се придружуваме во соби за разговор под име „temp“ и „room“.

По креирањето на корисничките сметки и најавата на клиентот „Pidgin“ со нив, следен чекор е извршување на симулацијата. Претходно е потребно да направиме поврзување со „border-router“, кој е даден како втор јазол во симулацијата, со користење на следната команда од терминал:

`make connect-router-cooja`, или

```
contiki/tools$ sudo ./tunslip6 -a 127.0.0.1 -p 60001 aaaa::1/64
```

Со ова се врши пренасочување на сите пакети од симулацијата во „Cooja“ преку „Serial Line Internet Protocol“ (SLIP) до мрежата на хост компјутерот на кој што се извршува „XMPP“ серверот (Prosody) и обратно [41]. Ова овозможува поврзување на Cooja со реалниот свет, овозможува проширување на пробите со виртуелни јазли и овозможува користење на податоците кои се добиваат во излезот на Cooja за детална анализа во ова истражување. Во овој случај рутерот кој е користен како друг „Z1“ јазол ја овозможува комуникацијата на јазолот во



Слика 22. Извршување на симулацијата со „XMPP“ и излез во „Pidgin“

„Cooja“ симулацијата со надворешниот клиент „Pidgin“ на кој е извршено логирање со корисничките сметки креирани на „Prosody“ серверот.

Следен чекор е стартување на „Prosody“ серверот ако веќе претходно не е стартуван. На крај треба да се стартува симулацијата во „Cooja“. Кога симулацијата се рестартира многу е важно да се рестартира и „Prosody“ серверот, бидејќи старите сесии за „IP“ мора да бидат избришани, инаку статусот на јазлите во симулацијата ќе биде „недостапен“ и нема да се случи најава.

„XMPP“ клиентот пристапува до соба за чат (room) во „Pidgin“ и во неа испишува текст кој е претходно предефиниран во програмата која е инсталирана како „firmware“ на безжичниот сензорен модул во „Cooja“. Во програмата се користи следниот код:

```
xmpp_send_muc_msg("contiki rocks", "'room@conference.localhost'");
```

Во овој случај во клиентот „Pidgin“ се испишува текстот „contiki rocks“ (Слика 22).

Во сценариото за протоколот „XMPP“ се мери потрошувачката на енергија само на клиентската страна како кај протоколот „MQTT“. Не се прават мерења на серверската страна бидејќи се користи надворешен сервер за комуникација (Prosody). Како и во другите сценарија, така и овде потрошувачката на енергија се прикажува во делот на излезот на јазолот во „Cooja“. Се користи модулот „Z1“ како и во претходните сценарија со извршен „XMPP“ клиент на него.

3.1 Пресметка за просечната потрошувачка на енергија

За пресметување на потрошувачката на енергија ќе биде користена алатката „Powertrace“. Тоа е механизам за одредување на потрошувачката на енергија што го користи следењето на состојбата за потрошувачката на секој јазол. Со него се врши разделување на потрошувачката според состојбата и се пропишува на активности на повисоко ниво [45]. Примери за такви активности се: индивидуалните апликации, индивидуалните протоколи или механизмите на протоколите како што е контролата на сообраќајот, препраќањето пакети или, пак, ретрансмисиите. „Powertrace“ овозможува испитување на однесувањето на енергијата на ниво на јазли или, пак, испитување на енергијата на мрежно ниво.

„Powertrace“ користи линеарен модел со кој моменталната енергија се проценува како збир од сите активни сосотојби. Системската енергија произлегува од времето кое го поминува системот во секоја состојба. Моменталната системска потрошувачка на енергија $P_{system}(t)$ во време t е дадена со равенката (1) [45]:

$$P_{system}(t) = \sum_{m,n} P_{m,n}\, s_{m,n}(t) \tag{1}$$

$P_{m,n}$ е потрошувачка на енергија на компонентата m во состојба n, додека $s_{m,n}$ е или 0 или 1 во зависност од тоа дали состојбата е влезена во време t или не [46]. Примери за компоненти се „CPU“, радио предавателот, вградените флеш-мемории и сензорите. Примери за состојби се „CPU“ во активна (active mode) или во мирувачка состојба (sleep mode) и радиото во состојба на слушање (listening mode) и во состојба на пренос (transmission mode). Енергетскиот модел е даден со равенката (2):

$$E_{system} = \sum_{n,m} P_{m,n}\, T_{m,n} \tag{2}$$

$T_{m,n}$ го претставува времето кое компонентата m го минува во состојба n. Постојаните фактори $P_{m,n}$ може да бидат претходно калибрирани или калибрирани во време на извршување [45].

„Powertrace“ ги следи состојбите (фазите на потрошувачка на енергија) со мерење на времето на компонентите во секоја состојба. Драјверите на уредите имаат можност да го снимаат временскиот печат кога компонентата влегува во

нова состојба. Кога компонентата ја напушта нејзината состојба, се пресметува временската разлика и се додава на соодветните $T_{m,n}$.

Овој механизам за следење на состојбата во која се наоѓа уредот заради одредување на потрошувачката се користи и кај компјутерите, преносните уреди како „Android“ и „Windows“ телефоните, мрежните вградени системи и слично. Следењето на состојбата се прави целосно во софтверот и не е потребен дополнителен хардвер. Софтверски-базираните решенија не се засегнати од фактори на влијанија од средината кои може да влијаат врз потрошувачката на енергија на системот, како што се температурата или влажноста. Покрај тоа, овие техники го даваат истиот резултат на различни серии од истиот хардвер, но тоа не мора да е точно за хардверски-базираните решенија. Како и да е и хардверски-базираните решенија за мерење на потрошувачката на енергија си имаат свои предности и недостатоци.

„Powertrace“ обезбедува точност од 94% во утврдувањето на потрошувачката на енергија за многу функции како за време на употребата на процесорот (CPU), радиопреносот, радиослушањето и слично [45].

Со извршување на апликацијата во „Cooja“ во излезот на јазолот „Z1“ се печатат вредностите за вкупниот број отчукувања на часовникот кои ги поминува „Z1“ во одредена фаза. Овие вредности се означени како: „ALL_CPU“, „ALL_LPM“, „ALL_TX“ и „ALL_RX“. „ALL_CPU“ го претставува вкупниот број отчукувања во активна состојба на централната процесорска единица. „ALL_LPM“ е вкупниот број отчукувања во фазата „LPM“ (Low Power Mode). „ALL_TX“ е вкупниот број отчукувања на часовникот во фазата „TX“, односно тоа е фазата што ја поминува модулот во состојба кога праќа податоци (Transmit). „ALL_RX“ е вкупниот број отчукувања на часовникот во фаза „RX“, односно тоа е фазата во која модулот прима податоци. Овие вредности ќе ги користиме за пресметка на просечната потрошувачка на енергија при користење на протоколите на апликациско ниво кај „IoT“. Печатењето на овие вредности во излезот на јазолот во „Cooja“, како што и претходно споменавме е со помош на алатката „Powertrace“, која вообичаено е составен дел на оперативниот систем „Instant Contiki“ и се наоѓа во директориумот со апликации [46]. „Powertrace“ во „Instant Contiki“ се состои од две датотеки именувани како „powertrace.c“ и

„powertrace.h“. Тие ги содржат сите функции кои се потребни за да се овозможи печатење на вредностите за отчукувањата на часовникот. Постои уште една датотека што е именувана како „Makefile.powertrace“, која ги содржи основните нагодувања за оваа алатка.

За мерење на потрошувачката на енергија се користи алатката „powertrace“, која се додава во делот „APPS“ во „Makefile“:

```
APP += powertrace
```

За да може да ги користиме функциите на „powertrace“, потребно е во кодот на програмата да ја додадеме следната библиотека:

```
#include "powertrace.h"
```

Бројот на отчукувања во секунда за „rtime“ кај дадениот модул може да се добие со командата за печатење:

```
printf("Ticks per second: %u\n", RTIMER_SECOND);
```

Притоа командата врши печатење на вредноста за „RTIMER_SECOND“ во излезот на јазолот во симулаторот „Cooja“. Кај модулот „Z1“, „RTIMER_SECOND = 32768“.

За да се овозможи печатење на вредностите за поминатото време на модулот во одредена сосотојба на секои 10 секунди, се користи командата:

```
powertrace_start(CLOCK_SECOND * 10);
```

Вредностите се печатат во редоследот како што се прикажани во Табела 7. Овде се дадени и вредности како пример од извршувањето на апликацијата за протоколот „MQTT“. Како што може да се види од првиот ред од табелата од времето на извршување на симулацијата, се врши печатење на вредностите на приближно 10 секунди како што беше поставено во кодот на програмата (конкретно во примерот за „MQTT“, временската разлика, 00:11.280-00:21.279, е приближно 10 секунди). Вториот параметар е стринг-вредност која го претставува идентификацискиот број (ID) на јазолот во симулацијата (во конкретниот пример ID:1). Третиот параметар (clock_time) го прикажува времето на часовникот. Четвртиот параметар секогаш е константа (P) и претставува навестување на следниот параметар (Rimeaddr_node_addr) кој ја прикажува „Rime“ адресата на јазолот во симулацијата. Следната вредност (seqno) е

Табела 7. Значење на вредностите на „Powertrace“ во „Cooja“ и вредности како пример

0	simulation time	00:11.280	00:21.279
1	str	ID:1	ID:1
2	clock_time()	1383	2663
3	P	P	P
4	Rimeaddr_node_addr	193.12	193.12
5	seqno	0	1
6	all_cpu	15649	23394
7	all_lpm	311888	631694
8	all_transmit	6447	9612
9	all_listen	4789	7074
10	all_idle_transmit	0	0
11	all_idle_listen	0	0
12	cpu	15649	7745
13	lpm	311888	319806
14	transmit	6447	3165
15	listen	4789	2285
16	idle_transmit	0	0
17	idle_listen	0	0

секвенциски број. Вредностите за параметрите 6-9 (ALL_CPU, ALL_LPM, ALL_TX, ALL_RX) се вредности кои беа претходно објаснети и нив ќе ги користиме во нашите пресметки за потрошувачката на енергија. Параметрите 10 и 11 (all_idle_transmit, all_idle_listen) го претставуваат времето кое го поминува модулот во неактивна состојба до дадена временска рамка до која се извршува симулацијата. Параметрите 12-15 (cpu, lpm, transmit, listen) ги претставуваат времињата кои ги поминува модулот помеѓу две временски рамки. Поточно, тоа е разликата помеѓу две последователни времиња. Ако го погледнеме примерот во Табела 7, во вториот временски период, ќе забележиме дека вредноста за „cpu“ изнесува 7745. Оваа вредност е добиена како разлика помеѓу вредностите за „ALL_CPU“ во вториот временски период (00:21.279) и првиот временски период (00:11.280) на модулот (23394-15649=7745). Ова истото се однесува и за вредностите на другите параметри кои се добиени на истиот начин. Последните два параметри 16-17 (idle_transmit, idle_listen) ги претставуваат разликите помеѓу вкупното време кое модулот го поминува во даден временски интервал во неактивна состојба (all_idle_transmit или all_idle_listen).

Постои и скратена верзија на „Powertrace“ со која не се печатат сите вредности, туку се печатат само вредностите на параметрите кои ни се потребни во пресметките за потрошувачката на енергија (all_cpu, all_lpm, all_transmit, all_listen, cpu, lpm, transmit, listen) [47]. Со оваа верзија се олеснува работата при пронаоѓањето на вредностите во излезот од „Cooja“ за одреден јазол, бидејќи наместо 18 параметри колку што се печатат со користењето на оригиналната верзија на „Powertrace“, со оваа верзија се печатат само 8 параметри.

По извршувањето на симулацијата и по печатењето на вредностите кои се прикажани во Табела 7, понатаму може да се пресмета потрошувачката на енергија во сите временски интервали. Од излезот во „Cooja“ најпрво потребно е да ги селектираме вредностите за параметрите бидејќи во излезот за јазолот се печатат и други информации. Ако постојат повеќе јазли, во тој случај се печатат излезите за сите. Затоа е многу важно да се погледне идентификацискиот број (ID) на јазолот кој е единствениот идентификатор.

За пресметување на потрошувачката на енергија во даден временски интервал се користи равенката (3) [46]:

$$Power\_consumption = \frac{Energest\_Value\ x\ Current\ x\ Voltage}{RTIMER\_SECOND\ x\ Runtime} \qquad (3)$$

Во овој случај со равенката (3) се добива вредност изразена во mW[33]. Ако се исфрли „Runtime“ од равенката (3), тогаш се добива потрошената енергија во mJ[34].

Во равенката (3) значењето на променливите е следно:

- „Energest_Value“ е разликата помеѓу вредностите во два последователни временски периоди кои ги поминува модулот во одредена состојба (ALL_CPU, ALL_LPM, ALL_RX, ALL_TX);
- „Current“ е потрошувачката на енергија на „CPU“ која може да се прочита од податоците за дадениот модул (истата е различна за секој модул и затоа треба да се провери);
- „Voltage“ е напонот на кој работи модулот;

[33] mW – единица за моќност, 1 mW = 0.001W, 1W = 1 J/s (Joule per second)

[34] mJ – единица за енергија, 1 mJ = 0.001 Joules

- „RTIMER_SECOND“ е бројот на отчукувања во секунда за кој е објаснето претходно како може се провери;
- „Runtime“ е времето на печатење на вредностите во „Cooja“ (во овој случај тоа е 10 секунди);

Пример за вредностите кои се дадени во Табела 7, потрошувачката за централната процесорска единица (CPU) во временскиот интервал од 11 до 21 секунда од извршувањето на симулацијата ќе биде:

(23394-15649)*5*3/32768*10 ≈ 0.354 mW

Овде разликата 23394-15649 е разликата помеѓу „ALL_CPU“ во двата временски интервала дадени како пример. Вредноста 5 се однесува на електричната струја на „CPU“ во активна состојба. Од Табела 6. може да се види дека оваа вредност за „CPU“ во активна состојба (Active Mode @16MHz) е околу 10 mA[35]. Во симулацијата модулот „Z1“ работи на 8MHz. Тоа е причината поради која во горната пресметка за струјата е користена вредност 5. Вредноста 3 го претставува напонот на кој работи модулот „Z1 Zolertia“. 32768 е „RTIMER_SECOND“. Вредноста 10 е „Runtime“, односно тоа е времето на извршување кое се нагодува во командата со која се обезбедува печатење на вредностите во „Contiki“ програмата. Вкупниот временски интервал на извршување на симулациите во „Cooja“ е 110 секунди (00:00.000 до 01:50.000). Времето се прикажува во делот за контрола на симулацијата во „Cooja“ (Simulation Control), каде уште се наоѓаат и командите за стартување на симулацијата, превчитување и контрола на брзината.

За пресметување на просечната потрошувачка на енергија потребно е да се пресмета потрошувачката на модулот во сите состојби во кои може да се најде. Потоа тие потрошувачки треба да се соберат и збирот да се подели со вкупниот број временски интервали на модулот.

[35] mA – единица за електрична струја, 1mA = 0.001 Ampere

4. Резултати и дискусија

Во Глава 3-та од оваа магистерска работа беа објаснети сценаријата во кои беше изведена симулацијата во „Cooja“. За сите протоколи беа извршени мерења на потрошувачката на енергија на клиентската страна. Само за протоколот „CoAP“ беа извршени мерења и на серверската страна. Во ова сценарио со промена на клиентите беше анализирана промената на потрошувачката на енергија на серверот.

Во Поглавје 3.1 детално беа објаснети методите за пресметка на потрошувачката на енергија на протоколите на апликациско ниво кај Интернетот на нештата: „CoAP“, „MQTT“ и „XMPP“.

Во овој дел ќе ги прикажеме добиените резултати од сите изведени сценарија, како и просечната потрошувачка на енергија. Резултатите кои беа добиени со помош на алатката „Powertrace“ и беа испечатени во излезот на јазлите во симулаторот „Cooja“ се претставени во табели и графици. Кај табелите во заглавјата се наоѓаат состојбите во кои може да се најде модулот „Z1 Zolertia“ (ALL_CPU, ALL_LPM, ALL_TX и ALL_RX), додека останатите редови ги претставуваат отчукувањата на часовникот на модулот во определена состојба во која може да се најде. Кај графиците временските интервали се претставени на x-координатата, додека потрошувачката на енергија (во mW) е претставена на y-координатата. Поточно, на графиците се прикажани вредностите за потрошувачката на енергија кои се пресметани во секој временски интервал. Сите овие вредности се прикажани во табели кои се дадени на истата слика со графиците. Во овие табели колоните ги претставуваат потрошувачките од состојбите додека последната колона (Total) го претставува збирот од потрошувачките во сите состојби само во одреден временски интервал. Последната вредност во последната колона ја претставува просечната потрошувачка на енергија од сите временски интервали (вкупно 10 временски интервали). На графиците секоја од линиите ја претставува промената на потрошувачката во вкупен интервал од 110 секунди што претходно беше предефиниран интервал за изведување на симулацијата за сите сценарија. За подобра прегледност секоја од линиите е претставена со различна боја за да се прави разлика помеѓу состојбите за кои се однесуваат соодветните вредности за потрошувачката на енергија.

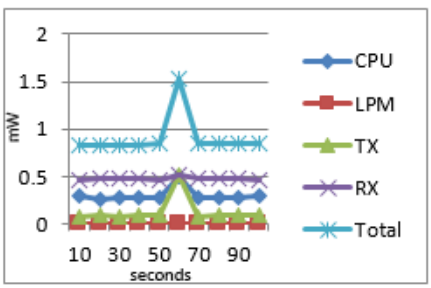
Табела 8. Податоци добиени со „Powertrace“ за „CoAP“ клиент

clock ticks (Rticks)			
ALL CPU	**ALL LPM**	**ALL TX**	**ALL RX**
12997	314635	5523	2668
19374	635910	6022	5370
25211	957724	6598	8112
31216	1279365	7080	10889
37180	1601053	7640	13629
43309	1922576	8263	16325
54037	2239497	11475	19345
60228	2560957	11976	22150
66330	2882507	12536	24898
72274	3204210	13096	27699
78540	3525595	13719	30386

4.1 Потрошувачка на енергија при употреба на протоколот „CoAP“

Во Табела 8 се прикажани податоците кои се добиваат со извршување на „Contiki“ апликацијата со примена на протоколот „CoAP“ во симулаторот „Cooja“. Во табелата е прикажано вкупното време што модулот „Z1“ го поминува во сите состојби, односно вкупниот број отчукувања на часовникот. Може да се види дека вредностите на податоците се зголемуваат како што се зголемува временскиот интервал (по колони), сè до 110-тата секунда од извршувањето на симулацијата. Овде се селектирани само податоците што ќе ги користиме во пресметката за потрошувачката, иако во излезот на јазолот во „Cooja“ се печатат и други податоци, како што беше објаснето претходно.

Energy Consumption(Power - mW)				
CPU	LPM	TX	RX	Total
0.29191589	0.00029414	0.07949158	0.46506592	0.83676752
0.26719666	0.00029463	0.09175781	0.47195068	0.83119978
0.27488708	0.00029447	0.07678345	0.47797485	0.82993986
0.27301025	0.00029451	0.08920898	0.47160645	0.8341202
0.28056335	0.00029436	0.099245	0.4640332	0.84413592
0.49108887	0.00029015	0.51167725	0.5197998	1.52285607
0.28340149	0.00029431	0.07981018	0.48279419	0.84630016
0.27932739	0.00029439	0.08920898	0.4729834	0.84181416
0.27209473	0.00029453	0.08920898	0.48210571	0.84370395
0.28683472	0.00029424	0.099245	0.46248413	0.84885808
				0.90796957



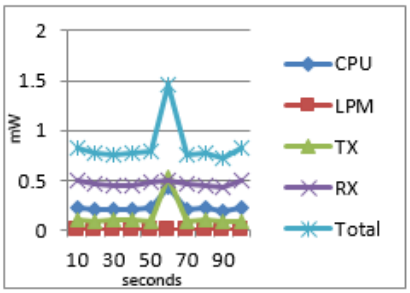
Слика 23. Потрошувачка на енергија кај „CoAP“ клиент

Со користење на податоците од Табела 8 и формулата (3) добиени се податоците за потрошувачката во сите временски интервали. Сите податоци за потрошувачката може да се погледнат на Слика 23. Овде тие се претставени според состојбата во која се наоѓа модулот „Z1“ (CPU, LPM, TX, RX). Во последната колона (Total) се наоѓа вкупната потрошувачка заедно со просечната потрошувачка како последна вредност. На оваа слика, од десна страна се наоѓа графичкиот приказ на потрошувачката кој беше споменат претходно.

Според добиената просечна потрошувачка, животниот век на батеријата на модулот „Z1“ изнесува 0.942943917 години. Целата пресметка е дадена подолу:

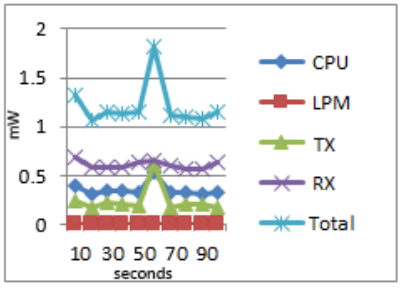
(2500mAh*1.5V*2) / (0.90796957mW*24h*365days) = 0.942943917 years (4)

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.23117065	0.00029535	0.09828918	0.48985107	0.81960627
0.21025085	0.00029577	0.09127991	0.46300049	0.76482702
0.20722961	0.00029583	0.10067871	0.4531897	0.76139385
0.21432495	0.00029569	0.1086438	0.44458374	0.76784818
0.21881104	0.0002956	0.08299622	0.48658081	0.78868366
0.42718506	0.00029143	0.53063416	0.50224365	1.4603543
0.20787048	0.00029582	0.08347412	0.46971313	0.76135356
0.21620178	0.00029565	0.1084845	0.44372314	0.76870508
0.19747925	0.00029602	0.09414734	0.42943726	0.72135987
0.22009277	0.00029558	0.09112061	0.50413696	0.81564592
				0.84297777



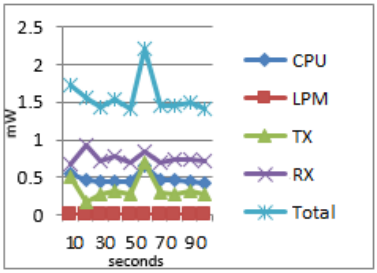
a) 1 клиент

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.3968811	0.00029202	0.23911194	0.68176392	1.31804898
0.30775452	0.00029381	0.16678894	0.58950806	1.06434532
0.34025574	0.00029316	0.21585388	0.58520508	1.14160785
0.33879089	0.00029319	0.20263184	0.58227905	1.12399497
0.33370972	0.00029329	0.18287842	0.63339844	1.15027986
0.5365448	0.00028923	0.62478149	0.65697876	1.81859429
0.32510376	0.00029346	0.1814447	0.59966309	1.10650501
0.32881165	0.00029339	0.20438416	0.56455078	1.09803997
0.3182373	0.00029359	0.20326904	0.56231323	1.08411317
0.33197021	0.00029332	0.17459473	0.64131592	1.14817418
				1.20537036



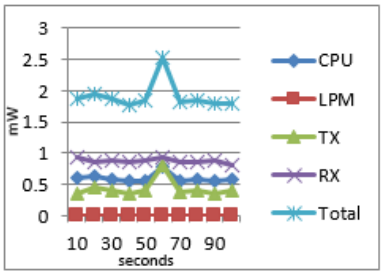
b) 2 клиента

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.54547119	0.00028906	0.50052612	0.67246948	1.71875586
0.46815491	0.00029061	0.17156799	0.92823853	1.56825204
0.43936157	0.00029119	0.26714905	0.71601563	1.42281743
0.45002747	0.00029098	0.31063843	0.77866699	1.53962386
0.43496704	0.00029127	0.27830017	0.68916504	1.40272352
0.65716553	0.00028683	0.70475098	0.84940796	2.21161129
0.46440125	0.00029068	0.29263733	0.69587769	1.45320694
0.45812988	0.00029081	0.27065369	0.73271118	1.46178556
0.45172119	0.00029094	0.3085675	0.73804688	1.49862651
0.42599487	0.00029146	0.26810486	0.72720337	1.42159456
				1.56989976



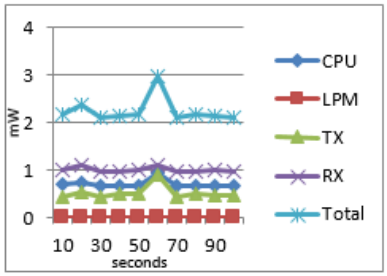
c) 3 клиенти

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.6002655	0.00028788	0.35062317	0.92686157	1.87803813
0.62644958	0.00028743	0.4530542	0.86541504	1.94520625
0.57971191	0.00028836	0.39650208	0.89192139	1.86842374
0.55279541	0.0002889	0.34839294	0.86042358	1.76190084
0.55773926	0.0002888	0.40717529	0.89002808	1.85523143
0.77554321	0.00028445	0.80479248	0.94613892	2.52675906
0.56707764	0.00028862	0.37611145	0.86489868	1.80837639
0.58277893	0.0002883	0.39968811	0.85164551	1.83440085
0.54977417	0.00028896	0.36527893	0.88159424	1.7969363
0.57165527	0.00028853	0.40637878	0.82169678	1.80001936
				1.90752924



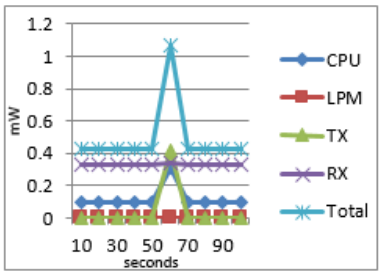
d) 4 клиенти

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.70628357	0.00028584	0.44763794	1.01240479	2.16661213
0.74789429	0.00028501	0.54624573	1.0943335	2.38875852
0.66508484	0.00028667	0.44620422	0.98211182	2.09368755
0.67991638	0.00028638	0.48889709	0.9810791	2.15017895
0.68453979	0.00028628	0.48873779	0.98882446	2.16238833
0.94633484	0.00028104	0.91678162	1.1168811	2.9802786
0.6787262	0.0002864	0.45130188	0.98125122	2.11156569
0.68692017	0.00028623	0.50084473	0.98125122	2.16930235
0.68330383	0.0002863	0.45687744	0.99192261	2.13239018
0.67025757	0.00028655	0.46850647	0.95870361	2.09775421
				2.24529165



e) 5 клиенти

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.09960938	0.00029796	0	0.33046875	0.43037609
0.0978241	0.000298	0	0.33046875	0.42859085
0.09786987	0.000298	0	0.33046875	0.42863662
0.09915161	0.00029797	0	0.33046875	0.42991833
0.09887695	0.00029798	0	0.33046875	0.42964368
0.30665588	0.00029382	0.42214966	0.33873047	1.06782983
0.10107422	0.00029793	0	0.33046875	0.4318409
0.09928894	0.00029797	0	0.33046875	0.43005566
0.0987854	0.00029798	0	0.33046875	0.42955213
0.0995636	0.00029796	0	0.33046875	0.43033031
				0.49367744



f) без клиенти

Слика 24. Потрошувачка на енергија на „CoAP“ сервер кој има 1-5 клиенти и состоіба во коіа нема клиенти

На Слика 24 е прикажана потрошувачката на енергија во случаите кога „CoAP“ серверот има од 1 до 5 клиенти или, пак, ако нема клиенти. Во сите овие сценарија е користена формулата (3) за пресметка на потрошувачката на енергија [46]. Во Табела 9 е прикажан животниот век на модулот во сите 6 сценарија. За пресметка на животниот век на батеријата на модулот „Z1“ е користена формулата (4). Како што може да се види од истражувањето, животниот век на батеријата се намалува како што се зголемува бројот на клиентите. Животниот век на батеријата е најголем во случајот кога ниту еден „CoAP“ клиент не пристапува до „CoAP“ серверот, додека животниот век на

Табела 9. Потрошувачка на енергија и животен век на батерија за „CoAP“ сервер (Z1 модул)

	Power consumption (mW)	**Lifetime (year)**
1 client	0.84297777	1.015642896
2 clients	1.205370361	0.710291551
3 clients	1.569899757	0.545362454
4 clients	1.907529235	0.448834213
5 clients	2.245291651	0.381315444
Without clients	0.493677441	1.734258674

батеријата е најмал кога имаме максимален број клиенти (односно 5 клиенти колку што е максимумот во ова сценарио) кои пристапуваат до „CoAP“ серверот. Така, просечната потрошувачка на енергија на „CoAP“ сервер што нема ниту еден клиент изнесува 0.494 mW. Според тоа животниот век на батеријата на модулот „Z1“ на кој е инсталиран „CoAP“ серверот ќе изнесува околу 1.73 години. Просечната потрошувачка на енергија во сценариото со 1 клиент изнесува околу 0.84 mW, со тоа животниот век на батеријата на модулот „Z1“ ќе биде околу 1 година. Просечната потрошувачка во второто сценарио со 2 клиента изнесува приближно 1.2 mW. Животниот век на батеријата со ова сценарио ќе биде приближно 0.71 години. Во третото сценарио со 3 клиенти просечната потрошувачка изнесува околу 1.57 mW, со тоа животниот век на батеријата ќе биде 0.54 години. Во сценариото со 4 клиенти просечната потрошувачка изнесува 1.9 mW, додека животниот век на батеријата ќе биде приближно 0.45 години. Просечната потрошувачка на енергија во сценариото со 5 клиенти изнесува околу 2.24 mW, што е за 1.746 mW повеќе од сценариото кога нема клиенти. Животниот век на батеријата на модул „Z1“ со инсталирани 5 клиенти на него ќе биде 0.38 години, што е за 1.35 години помалку од претходно споменатото сценарио без клиенти. Со тоа забележуваме дека драстично се намалува животниот век на батеријата на модулот со 5 клиенти. Ако ги погледнеме другите сценарија од Табела 9 кога се користат 1-4 клиенти кои пристапуваат до „CoAP“ серверот ќе се види како просечната потрошувачка на енергија се зголемува за околу 0.3 mW. Ако се погледне животниот век во сценариото кога нема клиенти и во случајот кога е додаден само 1 клиент, ќе се забележи дека животниот век со додавањето на клиентот се намалува за околу 0.7 години. Со додавањето уште еден клиент, животниот век се намалува за речиси 1 година споредено со сценариото без клиенти или за 0.3 години од сценариото со 1 клиент.

Како што беше објаснето, одредувањето на потрошувачката на енергија е многу важно за модули што имаат ограничени перформанси, а пред се ограничени капацитети на батерии. Тие често се поставени на непристапни предели заради добивање податоци за различни фактори во животната средина (температура, светлина). Освен, ова многу е важно да се направи преглед и за тоа колку клиенти ќе му пристапуваат, да речеме, на друг модул што ја игра

улогата на сервер (во ова сценарио „CoAP“ сервер). Од претходно разгледаните сценарија може да забележиме дека ако повеќе клиенти му пристапуваат на серверскиот модул, батеријата ќе се троши многу брзо. Со тоа многу е важна анализата за тоа колку ќе ни издржи батеријата во тие услови.

4.2 Потрошувачка на енергија при употребата на протоколот „MQTT“

Во Табела 10 се прикажани податоците кои се добиени од „Powertrace“ како излези за модулот „Z1“ во „Cooja“ симулацијата при употреба на протоколот „MQTT“. Во овој дел се прикажани само податоците за „MQTT“ клиент. Тој во овој случај се наоѓа на модул „Z1“. Како што беше објаснето во поглавјето 3, „MQTT“ клиентот праќа податоци до брокер „Mosquito“ што е инсталиран на локалната машина. Брокерот, пак, ги доставува овие податоци до друг клиент што е претплатен на дадената тема на која првиот клиент (објавувач) ги објавува пораките. Во овој дел немаме податоци за потрошувачката на серверот бидејќи тој е локално инсталиран и не постои можност за одредување на потрошувачката со користење на „Powertrace“ како алатка која помага при утврдувањето на потрошувачката.

Ако извршиме споредба со податоците добиени за „CoAP“ клиентот, ќе се види дека имаме одредени разлики во поглед на добиените вредности. Споредено со времето на модулот поминато во состојба „CPU“ во првите 10 секунди од симулацијата, „CoAP“ клиентот поминува помалку време во оваа

Табела 10. Податоци добиени со „Powertrace“ за „MQTT“ клиент

clock ticks (Rticks)			
ALL CPU	**ALL LPM**	**ALL TX**	**ALL RX**
15649	311888	6447	4789
23394	631694	9612	7074
30645	951993	12693	9267
33695	1276492	12773	11321
36411	1601327	12773	13477
44009	1921274	15936	15579
47607	2245216	16116	18057
50222	2570143	16116	19977
52826	2895083	16116	21897
55420	3220033	16116	23817
62558	3540437	19201	25785

состојба (12997), во споредба со „MQTT“ клиентот, кај кој бројот на отчукувањата на часовникот во првите 10 секунди изнесува 15649. Ако ја погледнеме крајната состојба (до 110 секунда од извршување на симулацијата) ситуацијата е обратна, односно „CoAP“ клиентот (модулот „Z1“) ја завршува симулацијата со поголем број на отчукувања во состојба „CPU“ (како вкупна состојба до тој период) во споредба со „MQTT“ клиентот.

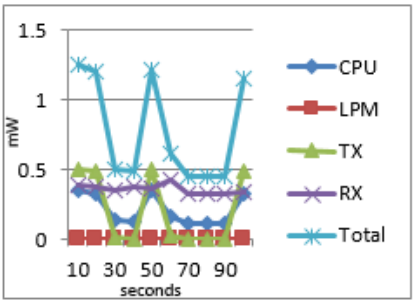
Во состојба „LPM“ до десетата секунда модулот „Z1“ („CoAP“ клиент) има повеќе отчукувања (314635) во споредба со модулот „Z1“ на „MQTT“ клиентот (311888). Во крајната состојба на симулацијата, ситуацијата со вредностите е обратна.

Во состојбата „TX“ која ја претставува состојбата кога се пренесуваат податоци (во овој случај до брокерот), бројот на отчукувањата на часовникот во првите десет секунди и на крајот од симулацијата за „CoAP“ клиенот е помал во споредба со „MQTT“ клиентот.

За последната состојба (RX) бројот на отчукувањата на модулот за „CoAP“ клиентот е поголем во споредба со „MQTT“ клиентот во првите 10 секунди, но на крајот на симулацијата (во 110-та секунда), ситуацијата е обратна.

На Слика 25 може да се погледне потрошувачката на енергија на „MQTT“ клиент во сите временски интервали од извршувањето на симулацијата, како и графикот на кој е претставена потрошувачката според состојбата во која се наоѓа модулот „Z1“. Најголемата потрошувачка модулот ја има во десеттата секунда (10 сек.), односно потрошувачката на енергија изнесува околу 1.25 mW.

Energy Consumption (Power - mW)				
CPU	LPM	TX	RX	Total
0.35453796	0.00029279	0.50419006	0.39329224	1.25231305
0.33192444	0.00029324	0.49080872	0.37745728	1.20048367
0.13961792	0.00029709	0.01274414	0.35353271	0.50619186
0.12432861	0.0002974	0	0.37108887	0.49571488
0.34780884	0.00029292	0.50387146	0.36179443	1.21376765
0.16470337	0.00029658	0.02867432	0.42651123	0.62018549
0.1197052	0.00029748	0	0.33046875	0.45047143
0.11920166	0.00029749	0	0.33046875	0.4499679
0.1187439	0.0002975	0	0.33046875	0.44951015
0.32675171	0.00029334	0.49144592	0.33873047	1.15722144
				0.77958275



Слика 25. Потрошувачка на енергија кај „MQTT“ клиент

Најмалата потрошувачка модулот ја има во деведесеттата секунда (90 сек.), односно во овој временски интервал таа изнесува околу 0.45 mW. Во споредба со „CoAP“ клиентот, кај кој минималната потрошувачка беше околу 0.8299 mW а максималната изнесуваше околу 1.523 mW, модулот „Z1“ кај „MQTT“ клиентот има помала максимална и минимална потрошувачка. Просечната потрошувачка на енергија, која ја добиваме кога вкупната потрошувачка во сите временски интервали ќе ја поделиме со вкупниот број временски интервали (во овој случај 10), изнесува 0.779582753 mW (Слика 25). Според ова, просечниот животен век на батеријата на модулот „Z1“ ќе биде приближно околу 1.098 години (според формулата (4)).

4.3 Потрошувачка на енергија кај „XMPP“ клиент

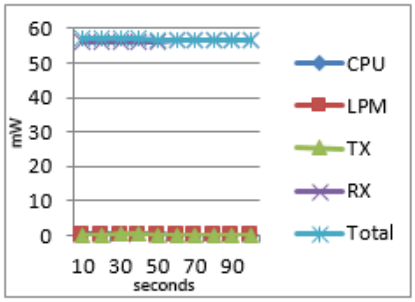
Во Табела 11 се прикажани податоците кои се добиени со помош на алатката „Powertrace“ за „XMPP“ клиент кој е имплементиран на модул „Z1“. Како во сценариото за протоколот „MQTT“ така и овде се врши испитување на потрошувачката само при користење на „XMPP“ клиент. Во ова сценарио не постои можност за испитување на потрошувачката за серверот бидејќи во овој случај се користи локално инсталиран сервер (Prosody XMPP server), кој се користи за регистрација на корисници и за другите функции како авторизација, автентикација и слично.

Со споредување на добиените податоци кај „CoAP“ и кај „MQTT“ клиент може да се воочат разликите во однос на добиените вредности со „Powertrace“.

Табела 11. Податоци добиени со „Powertrace“ за „XMPP“ клиент

clock ticks (Rticks)			
ALL CPU	**ALL LPM**	**ALL TX**	**ALL RX**
6842	320803	210	295123
18492	636738	771	622236
30372	952440	2037	948644
42660	1267730	4474	1273871
55981	1581976	6864	1599144
65398	1900144	7407	1926277
69093	2224021	7487	2253874
72536	2548149	7487	2581554
75983	2872272	7487	2909234
79419	3196405	7487	3236914
83082	3520316	7589	3564491

Energy Consumption (Power-mW)				
CPU	LPM	TX	RX	Total
0.53329468	0.00028925	0.08936829	56.3024084	56.9253607
0.54382324	0.00028903	0.20167603	56.1810645	56.9268528
0.5625	0.00028866	0.38821838	55.9777917	56.9287988
0.60978699	0.0002877	0.3807312	55.9857092	56.9765151
0.43107605	0.00029129	0.08650085	56.3058508	56.823719
0.16914368	0.00029652	0.01274414	56.3857141	56.5678984
0.15760803	0.00029675	0	56.4	56.5579048
0.15779114	0.00029674	0	56.4	56.5580879
0.1572876	0.00029675	0	56.4	56.5575844
0.16767883	0.00029655	0.01624878	56.3822717	56.5664959
				56.7389218



Слика 26. Потрошувачка на енергија кај „XMPP“ клиент

Така, „XMPP“ клиентот поминува многу повеќе време во состојба „CPU“ во споредба со клиентите на другите два протокола во првите 10 секунди од извршувањето на симулацијата во „Cooja“. Бројот на отчукувања во оваа состојба (6842) е речиси на половина во однос на „CoAP“ клиентот и „MQTT“ клиентот. Во последната секунда од извршувањето на симулацијата бројот на отчукувања кај „XMPP“ е поголем од вредностите кај двата протокола (83082).

Во втората состојба (LPM) бројот на отчукувања кај „XMPP“ клиентот (320803) во десеттата секунда е поголем во споредба на модулите кај другите 2 клиента, додека во крајниот временски интервал бројот на отчукувањата е помал (3520316).

Во третата состојба (TX) на модулот, бројот на отчукувањата е помал и во првите 10 секунди и на крајот од симулацијата, односно во 110-тата секунда.

Во последната состојба (RX) на модулот со „XMPP“ клиент бројот на отчукувањата е помал и во првите 10 секунди, и на крајот од извршувањето на симулацијата.

Потрошувачката на енергија на модул „Z1“, кој извршува „XMPP“ клиент може да се види на Слика 26. Ако се погледнат сите состојби, ќе се види дека најголемата потрошувачка на енергија се јавува во состојбата „RX“ (Receive). Во оваа состојба просечната потрошувачка на енергија достигнува дури 56 mW. Таа во основа ја дефинира и вкупната просечна потрошувачка во сите временски интервали на симулацијата. Вкупната просечна потрошувачка за модул „Z1“ кај „XMPP“ клиент изнесува приближно 56.74 mW.

Според добиените податоци за просечната потрошувачка на енергија за модул „Z1“ кај „XMPP“ клиент, животниот век на батеријата ќе биде приближно 0.015089543 години. Во споредба со животниот век на модулите „Z1“ кај другите протоколи, кај протоколот „XMPP“ тој е многу помал.

5. Развој на Андроид MQTT клиент

Во овој дел ќе го претставиме развојот на Андроид клиент за протоколот „MQTT“. Како што споменавме претходно „MQTT“ е протокол за кој основа е моделот „објави/претплати“ (publish/subscribe) за споделување пораки. Централната улога на моделот ја игра брокерот (broker). Неговата функција, како што беше споменато претходно, е да ги прима пораките од клиентите кои ги праќаат и да ги доставува до клиентите што сакаат да ги примаат. Според тоа, од двете страни на брокерот се наоѓаат два различни типа клиенти. Од едната страна се наоѓаат клиенти што праќаат пораки до брокерот и тие се наречени објавувачи на пораки (publishers). Од другата страна се наоѓаат клиенти кои се претплатени и сакаат да ги добиваат пораките и тие се наречени претплатници (subscribers). Во основа, објавувањето се прави на соодветни теми (topics) кои во суштина се хиерархиски подредени стрингови. Објавувачите ги праќаат пораките на тие теми, додека претплатниците се претплатуваат на темите и ги добиваат истите пораки.

5.1 Користена технологија

За развојот на Андроид „MQTT“ клиент е користено „Android Studio 2.0“ како една од основните алатки што денес се користи за развој на апликации за оперативниот систем за мобилни уреди „Андроид“. До овој период тоа е последната верзија на „Android Studio“ иако често се прават нови ажурирања, пред сè поради постојаните новини и технологии за овој оперативен систем. Оваа алатка овозможува брз начин за развивање квалитетни апликации за телефони, таблети, „Android Wear“, „Android TV“, „Android Auto“ и слично [48]. Таа е официјално „IDE“ (Integrated Development Environment) според „Google“ и ги нуди сите помошни алатки за развој, тестирање и интеграција на апликации за Андроид. Таа има уредувач за код, алатки за анализа на код, емулатори и друго. „Android Studio“ е работено според Андроид платформата и ги поддржува сите верзии на овој оперативен систем (има соодветни „API“). Последна верзија на овој оперативен систем до овој момент е „Android 6.0“, позната уште како „Marshmallow“.

Верзијата „Android Studio 2.0“ на овој оперативен систем нуди неколку подобрувања и нови функционалности во споредба со претходните верзии како [48]:

- „Instant Run“ – Со оваа функција се овозможува брз тек на процесот на развивање и тестирање на апликациите. Оваа функција ги анализира промените кои се направени во кодот на апликацијата и овозможува извршување на новиот код на најбрз начин. Со ова се врши внесување на промените на кодот во процесот на апликацијата која се извршува, со што се избегнува повторното извршување и инсталирање на апликацијата (APK). За некои апликации е потребно повторно стартување на активностите, односно повторно стартување на апликацијата. Оваа карактеристика ќе работи на сите уреди што имаат оперативен систем „Android“ или емулатор со „API 14“ (Ice Cream Sandwich) или поголемо.
- „Android Emulator“ – Новиот емулатор е три пати побрз во однос на „CPU“, „RAM“ и „I/O“ во споредба со претходните верзии на оваа развојна алатка за „Андроид“.
- „Cloud Test Lab“ – Нов сервис кој дозволува тестирање апликации на уреди во облакот.
- „App Indexing“ – Карактеристика со која корисниците ќе може побрзо да ги пронајдат нивните апликации со пребарување на „Google“.
- „GPU Debugger Preview“ – Ова е функционалност која е корисна за развивачите на игри и апликации и овозможува преглед на секоја рамка и „GL“ состојба со новиот „GPU“-дебагер.
- „IntelliJ 15 Update“ – Ажурирање на кодирачката платформа „IntelliJ“ врз основа на која е развиено „Android Studio“.

Клиентска библиотека која беше користена за развој на Андроид „MQTT“ клиент е „Eclipse Paho“ [49]. Оваа библиотека е достапна за повеќе програмски јазици. Во овој случај се користи библиотеката за „Java“ која е соодветна за развој на апликации за „Андроид“. Овој проект на „Eclipse“ бил еден од првите проекти со отворен код за „MQTT“ клиенти.

За развивање на апликацијата беше користен сервисот „Paho Android“. Тој е интерфејс на клиентската библиотека „Paho Java MQTT“ за „Андроид“ платформата. „MQTT“-врската е воспоставена во „Андроид“ сервисот кој што работи во позадина на „Андроид“ апликацијата, чувајќи ја активна додека апликацијата се префрла помеѓу различни активности.

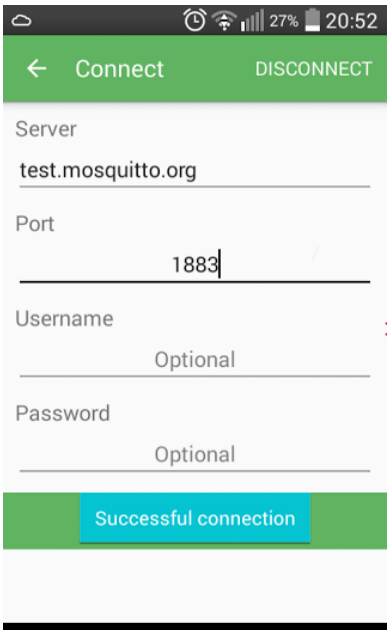
Сервисот „Paho Android“ поседува голем дел од карактеристиките на „MQTT“. Оваа библиотека има поддршка за верзиите „MQTT 3.1“ и „MQTT 3.1.1“. Таа има поддршка и за „LWT“ (LastWillTopic). Ги подржува сите три нивоа на квалитет на сервис (QoS 0, QoS 1 и QoS 2) и има можност за автентикација.

За да може да ги користиме функционалностите кои што ги нуди сервисот „Paho“, како и клиентските функционалности, потребно е во папката со библиотеки во „Android Studio“ да се додадат клиентската библиотека и библиотеката на сервисот („org.eclipse.paho.android.service.jar“ и „ org.eclipse.paho.client.mqttv3-1.0.2.jar“). Овие библиотеки е потребно да бидат додадени и во делот „Gradle“ во „Android Studio“ означен како зависности (dependencies) на следниот начин:

```
compile files('libs/org.eclipse.paho.client.mqttv3-1.0.2.jar')
compile files('libs/org.eclipse.paho.android.service.jar')
```

5.2 Андроид апликација „LearningNotes“

Поради применливост на „MQTT“ клиентот, развиена е една едноставна апликација со назив „LearningNotes“. Тоа е апликација која го применува моделот „објави/претплати“ на „MQTT“. Основната цел на апликацијата е да се користи во едукативни цели. Знаеме дека кога наставниците одржуваат часови, учениците најчесто ги запишуваат најважните работи од лекциите. Со тоа тие губат време, а и концентрација. Идејата е да се обезбеди услуга со која наставниците ќе можат да ги споделат најважните забелешки од лекцијата преку апликацијата. Во овој случај наставниците ја имаат улогата на објавувачи. Потребно е тие да ги достават забелешките до учениците преку апликацијата така што учениците нема да мора да ги запишуваат. За да се воспостави оваа врска помеѓу наставниците и учениците, потребен е брокер кој ја има централната улога кај „MQTT“. Во апликацијата во лентата за акции се наоѓаат три икони со чие кликнување се извршуваат функциите на апликацијата. Првата икона ја имплементира функцијата за објава кај „MQTT“. При кликнувањето на неа, се отвора нов прозорец преку кој можеме да објавуваме пораки на дадена тема. Втората икона ја има функцијата на претплата на дадена тема. По претплатата пораките се појавуваат во главниот дел на апликацијата филтрирани според темата на која се објавени. Последната икона ја има функцијата за остварување врска со брокерот. Со кликнувањето на неа, се



Слика 27. Воспоставување на врска

отвора нов прозорец во кој треба да ги дефинираме параметрите на врската (брокер, порта).

5.2.1 Воспоставување на врска

Сервисот „Paho Android“ има поддршка за „MQTT“ врска, и нуди „API“ за тоа. За да се овозможи користење на овој сервис, потребно е тој најпрво да биде деклариран во „AndroidManifest.xml“. За таа цел е потребно да бидат додадени следните линии код:

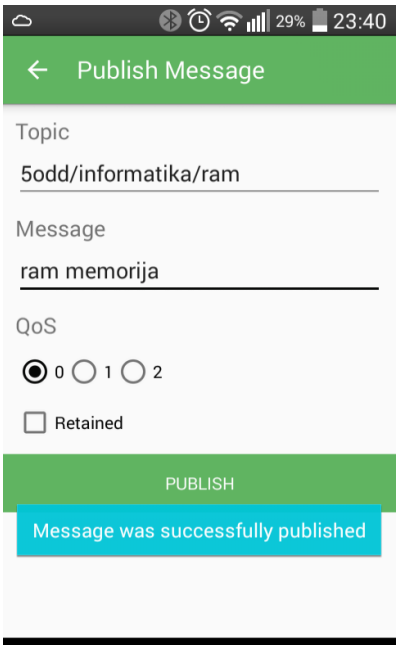
```
<service android:name="org.eclipse.paho.android.service.MqttService" >
</service>
```

Клиентскиот интерфејс „MQTTAndroidClient“ се поврзува на сервисот откако ќе биде креиран. На Слика 27 е прикажан екранот за воспоставување врска. Притоа, овде имаме 4 полиња за влез. Првите две полиња (Server, Port) се задолжителни додека полињата за корисничко име и лозинка, кои се користат за автентикација, не се задолжителни. Делот за сервер во основа го претставува брокерот. Во овој пример за воспоставување врска се користи брокер „Mosquitto“ на порта 1883. Може да се види дека врската е воспоставена успешно. Кодот за воспоставување врска е даден во Прилог А.

5.2.2 Објава

„MQTT“ Андроид клиент може да објавува пораки со помош на методот:

```
publish(topic, MqttMessage)
```



Слика 28. Објавување порака

Кај овој метод параметарот „topic“ претставува име на темата на која ќе биде објавена пораката што е дадена како втор параметар во методот (MqttMessage). Во основа, клиентот не ги става во редица пораките ако не е поврзан. Во тој случај ќе се јави грешка ако се обидеме да пратиме порака кога клиентот нема обезбедено врска.

За да се утврди дали клиентот е поврзан, се користи функцијата „isConnected()“ на клиентот (на пр., Client.isConnected() == true).

Може да се објавуваат и таканаречени „задржани“ пораки (retained messages). Кај овие пораки се зачувува последната вредност како најнова. Брокерот ја зачувува оваа вредност. Секогаш кога клиентите (учениците) ќе се претплатат на таа тема, тие ќе ја добијат пораката веднаш.

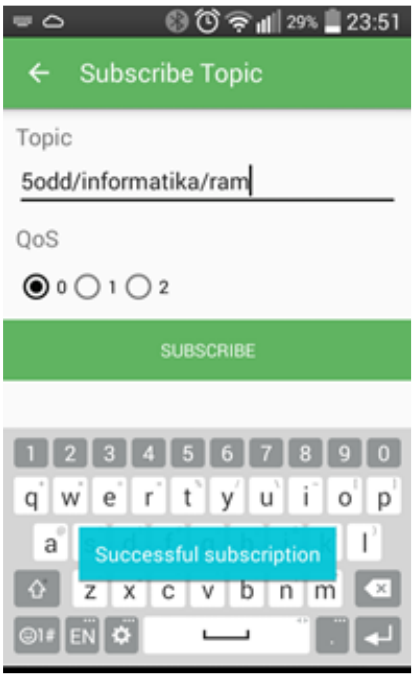
На Слика 28 е прикажано едно успешно испраќање порака со текст „ram memorija“ на тема „5odd/informatika/ram“. Во Прилог Б е прикажан кодот за објава.

5.2.3 Претплата

Претплатата на одредена тема се врши со помош на следната функција на клиентот:

```
client.subscribe(topic, qos);
```

Оваа функција има 2 параметра и тоа: „topic“ и „qos“. Првиот параметар ја претставува темата на која клиентот се претплатува, додека вториот параметар



Слика 29. Претплата на дадена тема

го претставува квалитетот на сервисот (со можни вредности 0,1 и 2). Методот „subscribe()“ враќа „IMqttToken“. Овој токен се користи за се одреди дали претплатата може да се воспостави успешно или, пак, поради одредени причини да не успее. Кодот за претплата напишан во „Java“ е прикажан во Прилог В.

На Слика 29 е направена успешна претплата на темата „5odd/informatika/ram“, со претходно поставена вредност за „QoS 0“. Откако ќе биде извршена

Слика 30. Објавени пораки на дадена тема

претплатата, тогаш сите пораки што ќе бидат испратени од објавувачите до брокерот на дадената тема брокерот ќе ги препрати до овој клиент. На Слика 30 се прикажани две последователно испратени пораки што се појавуваат кај претплатениот клиент на темата „5odd/informatika/ram“. Едната порака е со текст „16GB“, додека другата порака е со текст „ram memorija“.

Заклучок

Интернетот на нештата претставува концепт со кој во иднина ќе бидат опфатени голем број уреди. Најчесто, тоа се уреди што имаат ограничени перформанси, меморија и капацитет на батерија. Токму поради тоа е потребно да се обезбедат механизми со кои ќе се овозможи следење на овие карактеристики.

Во оваа магистерска работа беше разгледана енергетската потрошувачка на протоколите на апликациско ниво кај Интернетот на нештата: „CoAP“, „MQTT“ и „XMPP“. Испитувањето на потрошувачката беше извршено софтверски. За таа цел беше користен симулаторот „Cooja“. Тој е составен дел на оперативниот систем за Интернет на нештата познат како „Contiki“, и овозможува симулирање на безжични, сензорски модули. Во сценаријата за истражувањата беше користен безжичниот сензорски модул „Z1“. За мерење на потрошувачката беше користена алатката „Powertrace“, која овозможува печатење на потрошувачката во различни состојби на модулот (CPU, LPM, TX, RX).

Од анализата на протоколите на апликациско ниво кај Интернетот на нештата може да заклучиме дека просечната потрошувачка на енергија се разликува кај „CoAP“, „MQTT“ и „XMPP“. Според извршените истражувања, најголеми заштеди на енергија се обезбедуваат со користење на протоколите „CoAP“ и „MQTT“, додека „XMPP“ има значително поголема енергетска потрошувачка. Според тоа, животниот век на батеријата на модулот „Z1“ во овој случај ќе трае многу подолго со користење на протоколите „MQTT“ и „CoAP“ во споредба со протоколот „XMPP“. За сите протоколи најмногу енергија се троши во состојба „RX“ на модулот, додека најмалку енергија се троши во состојба „LPM“.

Протоколот „MQTT“ има значителна примена кај системите што применуваат архитектура „објави/претплати“ (publish/subscribe) за размена на пораки. Кај овие системи клиентите што праќаат пораки се нарекуваат објавувачи, додека оние што ги примаат пораките се нарекуваат претплатници. Централната улога кај овие системи ја има брокерот, кој е задолжен за примање и за препраќање на пораките до претплатниците. За целите на оваа магистерска работа е развиена „MQTT“ Андроид апликација, која го применува овој модел.

Користена литература

[1] Rose, K., Eldridge, S., Chapin, L. “The Internet of things: An overview”, The Internet Society (ISOC), 2015.

[2] Vermesan, O., Friess, P. “Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems”, River Publishers, 2013.

[3] Internet of Things, Strategic Research Roadmap. [Online]. Available: http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2009.pdf. (Access Date: 14.04.2016)

[4] Tschofenig, H., Arkko, J., Thaler, D., McPherson, D., “Architectural Considerations in Smart Object Networking”, Internet Architecture Board (IAB), 2015.

[5] The Internet of Things Reference Model. [Online]. Available: http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf. (Access Date: 25.04.2016)

[6] Emerging Open and Standard Protocol Stack for IoT. [Online]. Available: http://www.slideshare.net/aniruddha.chakrabarti/mphasis-digital-pov-emerging-open-standard-protocol-stack-for-iot (Access Date: 27.04.2016)

[7] Zigbee Wireless Standard. [Online]. Available: http://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard (Access Date: 27.04.2016)

[8] Software Design Specification, Z-Wave Protocol Overview. [Online]. Available: https://wiki.ase.tut.fi/courseWiki/images/9/94/SDS10243_2_Z_Wave_Protocol_Overview.pdf (Access Date: 27.04.2016)

[9] Palattella, M.R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L.A., Boggia, G., Dohler, M., “Standardized Protocol Stack for the Internet of (Important) Things,” *IEEE Communications Surveys & Tutorials*, vol.15, pp. 1389 – 1406, Jul.2013.

[10] Naagesh, S.B, “Design and Implementation of IEEE 802.15.4 Mac Protocol on FPGA,” in *Innovative Conference on Embedded Systems, Mobile Communication and Computing (ICEMC2)*, pp. 1-5, 2011.

[11] Ee, G.K., Ng, C.K., Noordin, N.K., Ali, B.M., “A Review of 6LoWPAN Routing Protocols,” *Asia-Pacific Advanced Network,* vol.30, pp. 71-81, 2010.

[12] Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. [Online]. Available: https://tools.ietf.org/html/rfc6282 (Access Date: 28.04.2016)

[13] TCP/IP Transport Layer, Transmission Control Protocol, TCP, User Datagram Protocol, UDP. [Online]. Available: http://www.omnisecu.com/tcpip/transport-layer.php (Access Date: 28.04.2016)

[14] The User Datagram Protocol (UDP). [Online]. Available: http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html (Access Date: 28.04.2016)

[15] Gallego, F. V., Zarate, J. A., Chatzimisios, P., Karagiannis, V. “A Survey on Application Layer Protocols for the Internet of Things,” *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11-17, 2015.

[16] The Constrained Application Protocol (CoAP), Internet Engineering Task Force (IETF), Universitaet Bremen TZI, Germany. [Online]. Available: https://tools.ietf.org/html/rfc7252 (Access Date: 28.04.2016)

[17] CoAP: The Web of Things Protocol, Zach Shelby. [Online]. Available: http://www.slideshare.net/zdshelby/coap-tutorial (Access Date: 29.04.2016)

[18] MQTT Version 3.1.1. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (Access Date: 29.04.2016)

[19] Fuqaha, A.A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M. “Internet of Things: A Survey on Enabling Technologies, Protocols and Applications,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347 - 2376, 2015.

[20] MQTT Essentials. [Online]. Available: http://www.hivemq.com/blog/mqtt-essentials/page/2/ (Access Date: 29.04.2016)

[21] Tang, K., Wang, Y., Liu, H., Sheng, Y., Wang, X., Wei, Z., “Design and Implementation of Push Notification System Based on the MQTT Protocol,” in *International Conference on Information Science and Computer Applications*, Changsha, Hu Nan, China, pp. 1-5, 2013.

[22] MQTT libraries for different programming languages. [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/libraries (Access Date: 30.04.2016)

[23] Getting Connected – Free MQTT brokers for ThingStudio. [Online]. Available: http://blog.thingstud.io/getting-started/free-mqtt-brokers-for-thingstudio/ (Access Date: 30.04.2016)

[24] Mosquitto An Open Source MQTT v3.1/v3.1.1 Broker. [Online]. Available: http://mosquitto.org/ (Access Date: 30.04.2016)

[25] Mosca broker. [Online]. Available: https://github.com/mcollina/mosca/wiki (Access Date: 30.04.2016)

[26] SYS Topics. [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics (Access Date: 03.05.2016)

[27] Extensible Messaging and Presence Protocol (XMPP): Core. [Online]. Available: https://tools.ietf.org/html/rfc6120 (Access Date: 04.05.2016)

[28] XMPP for Dummies - Creating Your Own Chat Application from Scratch - Part1. [Online]. Available: http://blog.blazeclan.com/xmpp-beginners-guide-creating-chat-application-xmpp-protocol/ (Access Date: 04.05.2016)

[29] XMPP for Dummies - Part 4 - Diving Deep into Stanzas. [Online]. Available: http://blog.blazeclan.com/xmpp-dummies-part-4-diving-deep-stanzas/ (Access Date: 05.05.2016)

[30] Contiki: The Open Source OS for the Internet of Things [Online]. Available: http://www.contiki-os.org/. (Access Date: 06.05.2016)

[31] Dunkels, A., Gronvall, B., Voigt, T., “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *Local Computer Networks, 2004. 29th Annual IEEE International Conference*, pp. 455-462, 2004.

[32] Contiki: Bringing IP to Sensor Networks [Online]. Available: http://ercim-news.ercim.eu/en76/rd/contiki-bringing-ip-to-sensor-networks. (Access Date: 07.05.2016)

[33] Contiki 2.6, The Rime communication stack [Online]. Available: http://contiki.sourceforge.net/docs/2.6/a01798.html#_details. (Access Date: 07.05.2016)

[34] The Official Contiki OS Blog [Online]. Available: http://contiki-os.blogspot.mk/. (Access Date: 08.05.2016)

[35] The Contiki Operating System, Instant Contiki 3.0 [Online]. Available: https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/. (Access Date: 08.05.2016)

[36] Roussel, K., Song, Y.Q., Zendra, O., “Using Cooja for WSN Simulations: Some New Uses and Limits,” in Proceedings of the 2016 International Conference on Embedded Wireless Systems and Network - EWSN ’16 - NextMote workshop, Graz, Austria, pp.319-324, 2016.

[37] Z1 Zolertia Wireless sensor module Datasheet [Online]. Available: http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf .(Access Date: 08.05.2016)

[38] RPL (Routing Protocol) Border Router [Online]. Available: http://anrg.usc.edu/contiki/index.php/RPL_Border_Router. (Access Date: 08.05.2016)

[39] REST example running on Cooja and Sky motes [Online]. Available: http://anrg.usc.edu/contiki/index.php/REST_example_running_on_Cooja_and_Sky_motes. (Access Date: 08.05.2016)

[40] MQTT client library for Contiki [Online]. Available: https://github.com/esar/contiki-mqtt. (Access Date: 09.05.2016)

[41] Klauck, R., Kirsche, M., “Chatty things - Making the Internet of Things readily usable for the masses with XMPP,“ in 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012), Pittsburgh, Pennsylvania, USA, pp.60-69, 2012.

[42] Hornsby, A., Bail, E., µXMPP: Lightweight implementation for low power operating system Contiki,“ in 2009 International Conference on Ultra Modern Telecommunications & Workshops, St. Petersburg, Russia, pp.1-5, 2009.

[43] Prosody IM [Online]. Available: https://prosody.im/. (Access Date: 10.05.2016)

[44] Pidgin IM Client [Online]. Available: https://www.pidgin.im/download/linux/. (Access Date: 10.05.2016)

[45] Dunkels, A., Eriksson, J., Finne, N., Tsiftes, N. “Powertrace: Network-level power profiling for low-power wireless networks”. Technical Report T2011:05, Swedish Institute of Computer Science, Mar. 2011.

[46] Contiki OS: Using Powertrace and Energest power profile to estimate power consumption [Online]. Available: http://thingschat.blogspot.mk/2015/04/contiki-os-using-powertrace-and.html. (Access Date: 10.05.2016)

[47] Contiki Powertrace update, Son Han repository [Online]. Available: https://github.com/sonhan/contiki/blob/master/apps/powertrace-sonhan/powertrace.c. (Access Date: 11.05.2016)

[48] Android Studio 2.0, Android Developers Blog [Online]. Available: http://android-developers.blogspot.mk/2016/04/android-studio-2-0.html. (Access Date: 14.05.2016)

[49] MQTT Client Library Encyclopedia – Eclipse Paho Java [Online]. Available: http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-eclipse-paho-java. (Access Date: 14.05.2016)

Прилог А

Код на активноста за воспоставување врска (ConnectActivity.java)

```
public class ConnectActivity extends AppCompatActivity {

    String uri, port1, username, password;
    EditText server_edit, port_edit, username_edit, password_edit;
    Button connect;
    MqttConnectOptions options;
    public static final String MYPREFERENCES = "MyPrefs";
    SharedPreferences sharedpreferences;
    public MqttAndroidClient client;
    public String clientId;
    MenuItem item;
    Menu menu;
    MenuInflater inflater;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_connect);

        ActionBar actionbar = getSupportActionBar();
        actionbar.setDisplayHomeAsUpEnabled(true);
        actionbar.show();


        connect = (Button)findViewById(R.id.connect);
        connect.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {


                    server_edit =
(EditText)findViewById(R.id.edit_server);
                    port_edit =
(EditText)findViewById(R.id.edit_port);

                    uri = server_edit.getText().toString();
                    port1 = port_edit.getText().toString();

                clientId = MqttClient.generateClientId();
                client = new
MqttAndroidClient(getApplicationContext(), "tcp://"+uri+":"+port1,
                        clientId);

                try {
                    MqttConnectOptions options = new
MqttConnectOptions();
                    options.setCleanSession(false);
```

```
                    IMqttToken token = client.connect(options);
                    token.setActionCallback(new
IMqttActionListener() {
                        @Override
                        public void onSuccess(IMqttToken
asyncActionToken) {
                            // We are connected
                            Toast.makeText(getBaseContext(),
"Successful connection", Toast.LENGTH_SHORT).show();
                            sharedpreferences =
getSharedPreferences(MYPREFERENCES, Context.MODE_PRIVATE);
                            SharedPreferences.Editor editor=
sharedpreferences.edit();

                            editor.putString("uri", uri);
                            editor.putString("port", port1);
                            editor.putString("clientId",clientId);
                            editor.commit();

                            item.setVisible(true);

                        }

                        @Override
                        public void onFailure(IMqttToken
asyncActionToken, Throwable exception) {
                            // Something went wrong e.g. connection
timeout or firewall problems
                            Toast.makeText(getBaseContext(),
"Connection failed", Toast.LENGTH_SHORT).show();
                            item.setVisible(false);

                        }
                    });
                } catch (MqttException e) {
                    e.printStackTrace();
                }



            }
        });

}

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        inflater = getMenuInflater();
        inflater.inflate(R.menu.connect_menu, menu);
```

```
        item = menu.findItem(R.id.disconnect_menu);

        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch(item.getItemId())
        {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
                return true;
            case R.id.disconnect_menu:
            {
                try {
                    IMqttToken disconToken = client.disconnect();
                    disconToken.setActionCallback(new
IMqttActionListener() {
                        @Override
                        public void onSuccess(IMqttToken
asyncActionToken) {
                            // we are now successfully disconnected

Toast.makeText(getApplicationContext(),"Successful
disconnection",Toast.LENGTH_SHORT).show();
                        }

                        @Override
                        public void onFailure(IMqttToken
asyncActionToken,
                                              Throwable exception) {
                            // something went wrong, but probably we
are disconnected anyway
                        }
                    });
                } catch (MqttException e) {
                    e.printStackTrace();
                }
            }
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    public MqttAndroidClient GetClient()
    {
        return this.client;
    }
}
```

Прилог Б

Код за активноста за објавување (PublishActivity.java):

```
public class PublishActivity extends AppCompatActivity {

    EditText pub, mess;
    Button pubButton;
    public static final String MYPREFERENCES = "MyPrefs";
    SharedPreferences sharedpreferences;
    MqttAndroidClient client;
    String topic, payload, clientId;
    byte[] encodedPayload;
    CheckBox retained_checkBox;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_publish);

        pub = (EditText)findViewById(R.id.publish_topic);
        mess = (EditText)findViewById(R.id.publish_message);
        pubButton = (Button)findViewById(R.id.button_publish);
        retained_checkBox =
(CheckBox)findViewById(R.id.retained_checkBox);
        android.support.v7.app.ActionBar actionBar =
getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
        actionBar.show();

        final ConnectActivity connectActivity = new
ConnectActivity();

        pubButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String string_pub = pub.getText().toString();
                String string_mess = mess.getText().toString();

                sharedpreferences =
getSharedPreferences(MYPREFERENCES, Context.MODE_PRIVATE);
                String uri = sharedpreferences.getString("uri",
"uri");
                String port = sharedpreferences.getString("port",
"1883");
                clientId = sharedpreferences.getString("clientId",
"0");

                client = new
```

```
MqttAndroidClient(getApplicationContext(), "tcp://"+uri+":"+port,
                        MqttClient.generateClientId());

                topic = string_pub;
                payload = string_mess;
                encodedPayload = new byte[0];

                try {
                    IMqttToken token = client.connect();
                    token.setActionCallback(new
IMqttActionListener() {
                        @Override
                        public void onSuccess(IMqttToken
asyncActionToken) {
                            // We are connected
                            client.setCallback(new
ExampleCallback());

                            try {
                                encodedPayload =
payload.getBytes("UTF-8");
                                MqttMessage message = new
MqttMessage(encodedPayload);
                                if (retained_checkBox.isChecked())
                                {
                                    message.setRetained(true);
                                }
                                client.publish(topic, message);

                                Toast.makeText(getBaseContext(),
"The message was successfully published!",
Toast.LENGTH_SHORT).show();

                            } catch (UnsupportedEncodingException |
MqttException e) {
                                e.printStackTrace();
                            }

                        }

                        @Override
                        public void onFailure(IMqttToken
asyncActionToken, Throwable exception) {
                            // Something went wrong e.g. connection
timeout or firewall problems
                            Toast.makeText(getBaseContext(),
"Unsuccessful publish", Toast.LENGTH_SHORT).show();

                        }
                    });
                } catch (MqttException e) {
```

```
                    e.printStackTrace();
                }
            }
        });

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId())
        {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
            default:
                return super.onOptionsItemSelected(item);
        }

    }
}
```

Прилог В

Код за активноста за претплата (SubscribeActivity.java):

```
public class SubscribeActivity extends AppCompatActivity{

    ContentValues values;
    String topic;
    EditText sub_edit;
    String currentDateTimeString;
    SharedPreferences mSharedPreferences;
    SharedPreferences sharedpreferences;
    String MYPREFERENCES="MyPrefs";
    MqttAndroidClient client;
    String clientId;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_subscribe);


        Button subscribe =
(Button)findViewById(R.id.button_subscribe);

        currentDateTimeString =
DateFormat.getDateTimeInstance().format(new Date());

        sharedpreferences = getSharedPreferences(MYPREFERENCES,
Context.MODE_PRIVATE);
        String uri = sharedpreferences.getString("uri", "uri");
        String port = sharedpreferences.getString("port", "1883");
        String clientId = sharedpreferences.getString("clientId",
"id");


        client = new MqttAndroidClient(getApplicationContext(),
"tcp://"+uri+":"+port,
                clientId);
        Toast.makeText(getApplicationContext(),"url: " +uri+" port
"+port+ " clientId "+ clientId,Toast.LENGTH_LONG).show();
        subscribe.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sub_edit =
(EditText)findViewById(R.id.subscribe_topic);
                topic = sub_edit.getText().toString();

                Cursor c =
getContentResolver().query(MyTodoContentProvider.CONTENT_URI, null,
TodoTable.COLUMN_TOPIC + " = " +
DatabaseUtils.sqlEscapeString(topic), null, null);
```

```
                SubscribeTodoDatabaseHelper db = new
SubscribeTodoDatabaseHelper(getApplicationContext());
                if(c.getCount() == 0) {

                    try {
                        MqttConnectOptions options = new
MqttConnectOptions();
                        options.setCleanSession(false);

                        IMqttToken subToken =
client.connect(options);
                        subToken.setActionCallback(new
IMqttActionListener() {
                            @Override
                            public void onSuccess(IMqttToken
asyncActionToken) {
                                // The message was published
                                try {
                                    IMqttToken subToken =
client.subscribe(topic, 1);
                                    subToken.setActionCallback(new
IMqttActionListener() {
                                        @Override
                                        public void
onSuccess(IMqttToken asyncActionToken) {
                                            // The message was
published

Toast.makeText(getApplicationContext(), "Successful subscription",
Toast.LENGTH_SHORT).show();
                                            values = new
ContentValues();

values.put(TodoTable.COLUMN_TOPIC, topic);

values.put(TodoTable.COLUMN_DATE, currentDateTimeString);

getContentResolver().insert(MyTodoContentProvider.CONTENT_URI,
values);
                                            client.setCallback(new
MqttCallback() {
                                                @Override
                                                public void
connectionLost(Throwable throwable) {

                                                }

                                                @Override
                                                public void
messageArrived(String s, MqttMessage mqttMessage) throws Exception {
```

```
String date;
ContentValues values;
String messageBody = new String(mqttMessage.getPayload());

Log.d("message", messageBody);
date = DateFormat.getDateTimeInstance().format(new Date());

values = new ContentValues();

values.put(SubscribeTodoTable.COLUMN_TOPIC, s);

values.put(SubscribeTodoTable.COLUMN_MESSAGE, messageBody);

values.put(SubscribeTodoTable.COLUMN_DATE, date);

getContentResolver().insert(SubscribeTodoContentProvider.CONTENT
_URI, values);


SharedPreferences sharedpreferences =
getSharedPreferences("Prefs", Context.MODE_PRIVATE);

SharedPreferences.Editor editor = sharedpreferences.edit();

editor.putString("message", messageBody);

editor.commit();

}

@Override
public void deliveryComplete(IMqttDeliveryToken
iMqttDeliveryToken) {

}
});
}

@Override

public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {

// The subscription could not be performed, maybe the user was
not
// authorized to subscribe on the specified topic e.g. using
wildcards

Toast.makeText(getApplicationContext(), "Unsuccessful
subscription", Toast.LENGTH_SHORT).show();

}
```

```
});
} catch (MqttException e) {
          e.printStackTrace();
}

}

@Override
public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
// The subscription could not be performed, maybe the user was
not

// authorized to subscribe on the specified topic e.g. using
wildcards

Toast.makeText(getApplicationContext(), "Please connect to the
broker!", Toast.LENGTH_SHORT).show();

                                }
                            });
                        } catch (MqttException e) {
                            e.printStackTrace();
                        }

                    }
                    else
                    {
                        Toast.makeText(getApplicationContext(),"You
are already subscribed on topic "+topic,
Toast.LENGTH_SHORT).show();
                    }
                }
            });
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId())
        {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);

            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```