Izvedba i testiranje komunikacijskog sustava baziranog na MQTT protokolu

Košutar, Dario

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:041187

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-12**

Repository / Repozitorij:

University North Digital Repository

Sveučilište
Sjever

Završni rad br. 420/EL/2017

Izvedba i testiranje komunikacijskog sustava baziranog na MQTT protokolu

Dario Košutar, 3284/601

Varaždin, lipanj 2018. godine

Sveučilište
Sjever

Odjel za Elektrotehniku

Završni rad br. 420/EL/2017

Izvedba i testiranje komunikacijskog sustava baziranog na MQTT protokolu

Student

Dario Košutar, 3284/601

Mentor

mr.sc Matija Mikac, dipl.ing.

Varaždin, lipanj 2018. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za elektrotehniku

PRISTUPNIK: Dario Košutar | MATIČNI BROJ: 3284/601

DATUM: 13.12.2017. | KOLEGIJ: Računalne mreže

NASLOV RADA: Izvedba i testiranje komunikacijskog sustava baziranog na MQTT protokolu

NASLOV RADA NA ENGL. JEZIKU: Development and testing of MQTT based communication system

MENTOR: Matija Mikac | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. Miroslav Horvatić, dipl.ing.
2. mr.sc. Ivan Šumiga, dipl.ing.
3. mr.sc. Matija Mikac, dipl.ing.
4. Stanko Vincek, struč.spec.ing.el. (zamjenski član)
5.

Zadatak završnog rada

BROJ: 420/EL/2017

OPIS:

Jedno od rješenja za efikasnu komunikaciju u IoT sustavima, bazirano na pretplati/objavi principu, jest korištenje MQTT protokola. Naravno, primjena tog protokola moguća je i u raznim drugim sustavima. Radi se o komunikacijskom protokolu definiranom na aplikacijskom sloju TCP/IP modela koji definira koncept i mehanizme dovoljne za izvedbu pretplati/objavi komunikacijskih sustava. Preduvjet korištenja je podrška na pojedinim uređajima (klijentima i brokerima) - obzirom na dostupnost podrške za razne platforme, nužno je u okviru implementacije MQTT komunikacijskog sustava donijeti odluke o korištenju pojedinih rješenja, kao i mogućnosti primjene na ciljanim platformama (npr. instalacija softvera za MQTT broker na računalo ili korištenje dostupnih online rješenja, instalacija MQTT klijenata na računala, mobilne uređaje, mikroupravljače itd.)

U radu je potrebno:

* dati kratki osvrt na IoT i mogućnosti primjene MQTT protokola,
* dati pregled mogućnosti MQTT protokola,
* dati pregled dostupnih izvedbi MQTT brokera (softver) i mogućnosti primjene na različitim klijentima i platformama,
* kao praktični dio rada realizirati kompletni sustav koji koristi MQTT za komunikaciju, provesti analizu mogućnost i testiranje,
  * instalirati i konfigurirati MQTT broker korištenjem dostupnih softverskih rješenja,
  * razviti mobilnu aplikaciju za nadzor sustava, instalirati i konfigurirati MQTT klijent unutar aplikacije za mobilni uređaj (Android),
  * implementirati i provesti testiranje MQTT klijenta na mikroupravljaču ESP 8266-12F (za potrebe nadzora i upravljanja),
* komentirati postupak implementacije,
* dati analizu performansi (mrežni promet, vrijeme odziva i slično) realiziranog sustava.

ZADATAK URUČEN: 21.12.2017 | POTPIS MENTORA: A. Mikac

REPUBLIKA HRVATSKA SVEUČILIŠTE SJEVER

Predgovor

U današnje vrijeme sve se više govori o međusobnom povezivanju različitih stvari putem Interneta, takozvanom „Internetu stvari“ (eng. *Internet of Things,* IoT). Da bi te stvari (uređaji, senzori i slično) imale sposobnost međusobne komunikacije moraju postati „pametni“ objekti na mreži. U IoT projektima to se vrši kombinacijom mikroupravljača i različitih vrsta senzora. Novije verzije mikroupravljača imaju sposobnost povezivanja na mrežu. Što znači da se kombinacijom različitih senzora i mikroupravljača može bilo koju stvar pretvoriti u „pametan“ objekt koji može preko senzora nešto mjeriti i te informacije dijeliti sa ostalim stvarima („pametnim“ objektima). Da bi ti „pametni“ objekti mogli međusobno komunicirati potreban je protokol koji to omogućava. Budući da su mikroupravljači ograničeni resursima, taj protokol treba biti „lagan“, a isto tako mora biti efikasan i siguran. Jedan od protokola koji se danas sve više koristi za takve projekte je MQTT (eng. *Message Queue Telemetry Transport*) protokol. Ovaj završni rad najvećim dijelom bavi se upravo MQTT protokolom i njegovom implementacijom na resursno ograničene uređaje kao što su mikroupravljači.

Sažetak

U ovom radu detaljno je opisan MQTT (eng. *Message Queue Telemetry Transport*) protokol, njegove funkcionalnosti i primjena. Rad u svojem prilogu sadrži opsežnu dokumentaciju MQTT protokola. U praktičnom dijelu rada provedena je i opisana implementacija studijskih primjera korištenja MQTT protokola. Opisan je postupak instalacije, konfiguriranja i korištenja MQTT poslužitelja (u skladu s nazivljem koje MQTT koristi, brokera) bilo da se radi o računalu na lokalnoj mreži ili javno dostupnim resursima (MQTT broker na poslužitelju javno dostupnom na Internetu). Odnosno demonstrirana je instalacija MQTT brokera na virtualnom privatnom poslužitelju. Obzirom na primjenu, koja može zahtijevati osiguranje sigurnosne komunikacije, obrađuje se i korištenje sigurnosnog podsloja (TLS/SSL) u komunikaciji. Da bi se demonstrirao rad MQTT protokola na resursno ograničenim uređajima korišten je ESP 8266-12F mikroupravljač. Također kao segment praktičnog dijela rada, napravljena je Android mobilna aplikacija koja koristi MQTT klijentsku programsku knjižnicu, pomoću koje može komunicirati s MQTT brokerom i na taj način kontrolirati ili nadzirati određene ulaze/izlaze mikroupravljača. Uz to Android aplikacijom demonstrirana je mogućnost vizualizacije podataka sa senzora mikroupravljača dobivenih korištenjem MQTT protokola. U aplikaciju je implementirana vlastita metoda mjerenja odaziva MQTT brokera.

Ključne riječi: MQTT protokol, M2M komunikacija, IoT, MQTT broker, MQTT klijent, Android aplikacija, ESP 8266-12F.

Abstract

In this paper MQTT (eng. *Message Queue Telemetry Transport*) protocol is described in detail, as well as its functionality and its application. The paper in its attachment contains extensive documentation of the MQTT protocol. Implementation of study examples using MQTT protocol has been implemented and described in the practical part of the work. The procedure of installing, configuring and using MQTT server (according to the term used by MQTT, broker) is described, whether it is a computer on a local area network or publicly available resources (MQTT broker on a server, publicly available on the Internet). For example it is demonstrated installation of MQTT broker on the virtual private server. Due to the application, which may require secure communication use of the TLS/SSL is also handled. To demonstrate the work of MQTT protocol on resource limited devices ESP 8266-12F microcontroller was used. Also as a handy part of the work, an Android mobile application that uses MQTT client software library is used to communicate with MQTT broker and that way can control or monitor specific inputs/outputs of microcontroller. In addition with this Android application it is also possible to visualize data from the microcontroller sensor obtained using MQTT protocol. The application has implemented method of measuring MQTT broker time response.

Key words: MQTT protocol, M2M communication, IoT, MQTT broker, MQTT client, Android application, ESP 8266-12F.

Popis korištenih kratica

MQTT	(eng. *Message Queue Telemetry Transport*), komunikacijski protokol aplikacijskog sloja baziran na objavi/pretplati modelu
IoT	(eng. *Internet of Things*), Internet stvari
HTML	(eng. *HyperText Markup Language*), jezik za opisivanje hipertekstualnih dokumenata
TLS/SSL	(eng. *Transport Layer Security/Secure Sockets Layer*), skupina protokola za realizaciju sigurnosne komunikacije (kriptografija podataka i drugi sigurnosni mehanizmi)
TCP/IP	(eng. *Transmission Control Protocol/Internet Protocol*), standardni protokoli (protokolarni složaj) pomoću kojih se vrši razmjena podataka koristeći Internet vezu
QoS	(eng. *Quality of Service*), kvaliteta usluge
DUP	(eng. *Duplicate Flag*), zastavica za dupliciranje
HTTP	(eng. *Hyper Text Transfer Protocol*), protokol za prijenos (komunikacija između HTTP klijenta i poslužitelja, standardni upit-odgovor način rada) podataka na webu
VPS	(eng. *Virtual Private Server*), virtualni privatni poslužitelj
HTTPS	(eng. *Hyper Text Transfer Protocol Secure*), protokol za prijenos podataka na webu, sa aktiviranom sigurnosnom komunikacijom (uz korištenje TLS/SSL protokola)
ARP	(eng. *Address Resolution Protocol*), komunikacijski protokol kojim se dobiva fizička adresa na lokalnoj mreži iz poznate mrežne adrese
IEEE	(eng. *Institute of Electrical and Electronics Engineers*), međunarodna neprofitna organizacija za uznapredovanje tehnologije vezane sa elektricitetom
UART	(eng. *Universal Asynchronous Receiver/Transmitter*), asinkrona serijska komunikacija
USB	(eng. *Universal Serial Bus*), univerzalna serijska sabirnica
GPIO	(eng. *General Pourpose Input/Output*), ulaz/izlaz generalne namjene
MSB	(eng. *Most Significant Bit*), najznačajniji bit
LSB	(eng. *Least Significant Bit*), najmanje značajan bit

Sadržaj

1. Uvod 1
2. MQTT protokol 3
   2.1. MQTT klijent 4
   2.2. MQTT broker 4
   2.3. Uspostava MQTT veze između MQTT klijenta i brokera 5
   2.4. Usporedba između HTTP i MQTT protokola 6
3. Osnovni koncept MQTT protokola 8
   3.1. Objavi/pretplati 8
   3.2. Teme i pretplate 8
   3.3. Kvaliteta usluge (QoS) 9
      3.3.1. Razmjena poruka na razini kvalitete usluge QoS 0 9
      3.3.2. Razmjena poruka na razini kvalitete usluge QoS 1 10
      3.3.3. Razmjena poruka na razini kvalitete usluge QoS 2 10
   3.4. Zadržane poruke 11
   3.5. Čista sesija 11
   3.6. Oporuke 11
4. Struktura MQTT paketa 13
   4.1. Fiksno zaglavlje MQTT kontrolnog paketa 13
      4.1.1. Tip MQTT paketa 13
      4.1.2. Zastavice 14
      4.1.3. Princip formatiranja MQTT paketa 15
   4.2. Varijabilno zaglavlje MQTT paketa 15
      4.2.1. Zastavica čiste sesije 17
      4.2.2. Zastavica oporuke 17
      4.2.3. Zastavica kvalitete usluge oporuke 18
      4.2.4. Zastavica zadržavanja oporuke 18
      4.2.5. Zastavice za korisničko ime i lozinku 18
   4.3. Teret 19
5. Standardni paketi u MQTT protokolu 20
   5.1. `CONNECT` i `CONNACK` paketi 20
   5.2. `PUBLISH` i `PUBACK` paketi 20
      5.2.1. `PUBREC`, `PUBREL`, `PUBCOMP` paketi 20
   5.3. `SUBSCRIBE` i `SUBACK` paketi 21
   5.4. `UNSUBSCRIBE` i `UNSUBBACK` paketi 21
   5.5. `PINGREQ` i PINGRESP paketi 22
   5.6. `DISCONNECT` paket 22
6. Demonstracija MQTT protokola 23
   6.1. Instalacija MQTT brokera 23
   6.2. Postavljanje u lokalnoj mreži 24
      6.2.1. Analiza MQTT paketa na lokalnoj mreži pomoću programa Wireshark 27
      6.2.2. Korištenje sigurnosnog transportnog sloja SSL/TLS u lokalnoj mreži 29
   6.3. Instalacija MQTT brokera na udaljeni poslužitelj 31
      6.3.1. Instalacija Mosquitto MQTT brokera na udaljeni poslužitelj 32
      6.3.2. Testiranje povezivanja na Mosquitto MQTT broker instaliran na udaljenom poslužitelju 34
      6.3.3. Funkcioniranje TLS/SSL protokola za uspostavu sigurnosne veze 36

6.3.4. Demonstracija napada na Mosquito MQTT broker 36
7. ESP 8266-12F mikroupravljač kao MQTT klijent 41
7.1. ESP 8266-12F mikroupravljač 41
7.2. Elektronički sklop za rad s ESP 8266-12F mikroupravljačem 43
7.3. Programiranje ESP 8266-12F mikroupravljača 46
7.4. Programski kod ESP 8266-12F mikroupravljača 46
8. Android aplikacija kao MQTT klijent 49
8.1. Dizajn Android aplikacije 49
8.2. Programiranje Android aplikacije 50
8.3. Metoda mjerenja odaziva MQTT protokola korištena u Android aplikaciji 53
8.4. Postavljanje oporuke 55
9. Analiza i mjerenja u implementiranom sustavu 57
9.1. Sinkronizacija između Android aplikacije i mikroupravljača 57
9.2. Rezultati odaziva MQTT brokera 62
9.3. Skalabilnost MQTT brokera 66
10. Zaključak 69
11. Literatura 70
12. Prilozi 76
12.1. Struktura MQTT paketa (MQTT verzija 3.1.1.) 76
12.1.1. Struktura fiksnog zaglavlja CONNECT paketa 76
12.1.2. Struktura varijabilnog zaglavlja CONNECT paketa 76
12.1.3. Struktura fiksnog zaglavlja CONNACK paketa 77
12.1.4. Struktura varijabilnog zaglavlja CONNACK paketa 77
12.1.5. Struktura fiksnog zaglavlja PUBLISH paketa 77
12.1.6. Struktura varijabilnog zaglavlja PUBLISH paketa 77
12.1.7. Struktura fiksnog zaglavlja PUBACK paketa 78
12.1.8. Struktura varijabilnog zaglavlja PUBACK paketa 78
12.1.9. Struktura fiksnog zaglavlja PUBREC paketa 78
12.1.10. Struktura varijabilnog zaglavlja PUBREC paketa 78
12.1.11. Struktura fiksnog zaglavlja PUBREL paketa 78
12.1.12. Struktura varijabilnog zaglavlja PUBREL paketa 79
12.1.13. Struktura fiksnog zaglavlja PUBCOMP paketa 79
12.1.14. Struktura varijabilnog zaglavlja PUBCOMP paketa 79
12.1.15. Struktura fiksnog zaglavlja SUBSCRIBE paketa 79
12.1.16. Struktura varijabilnog zaglavlja SUBSCRIBE paketa 79
12.1.17. Struktura tereta SUBSCRIBE paketa 80
12.1.18. Struktura fiksnog zaglavlja SUBACK paketa 80
12.1.19. Struktura varijabilnog zaglavlja SUBACK paketa 81
12.1.20. Struktura tereta SUBACK paketa 81
12.1.21. Struktura fiksnog zaglavlja UNSUBSCRIBE paketa 81
12.1.22. Struktura varijabilnog zaglavlja UNSUBSCRIBE paketa 82
12.1.23. Struktura tereta UNSUBSCRIBE paketa 82
12.1.24. Struktura fiksnog zaglavlja UNSUBACK paketa 82
12.1.25. Struktura varijabilnog zaglavlja UNSUBACK paketa 82
12.1.26. Struktura fiksnog zaglavlja PINGREQ paketa 83
12.1.27. Struktura fiksnog zaglavlja PINGRESP paketa 83
12.1.28. Struktura fiksnog zaglavlja DISCONNECT paketa 83
12.2. Elektronički sklop za rad sa ESP 8266-12F mikroupravljačem 84

12.3. Java programski kod za Android aplikaciju 84
12.3.1. Programski kod Login Activity 84
12.3.2. Programski kod Main Activity 87
12.4. C programski kod korišten za ESP 8266-12F mikroupravljač 95

1. Uvod

Cilj ovog završnog rada je upoznati se s MQTT (eng. *Message Queue Telemetry Transport*) protokolom, upoznati njegovu arhitekturu i funkcionalnosti. MQTT je komunikacijski protokol koji funkcionira na principu objavi/pretplati (eng. *Publish/Subscribe*). Protokol je „lagan“, otvoren i jednostavan za korištenje [1]. Baš te karakteristike čine ga idealnim za korištenje u mnogo situacija, uključujući ograničena okruženja kao što je komunikacija „stroja sa strojem“ (eng. *Machine to Machine*). Također je pogodan za mnogobrojne projekte u kontekstu „Interneta stvari“ (eng. *Internet of Things*, IoT), gdje je za rad potrebno što jednostavnije rješenje, uz često, ograničene procesne resurse, a propusnost mreže je velika. Telemetrijska tehnologija omogućuje da različite vrste procesa budu nadzirane, mjerene te kontrolirane iz daljine. Današnja poboljšanja u telemetrijskoj tehnologiji omogućavaju povezivanje različitih mjernih uređaja, na način da komuniciraju jedan s drugim na različitim geografskim lokacijama. U današnje vrijeme ljudi, tvrtke pa čak i vlade, sve se više okreću „pametnim“ uređajima i telemetrijskoj tehnologiji da bi se na što „pametniji“ način povezali sa svijetom oko sebe. Smisao korištenja kombinacije telemetrije i Interneta može se prikazati na primjeru čovjeka koji ide u kupovinu namirnica te želi znati što mu je trenutno doma u skladištu. Informacija koja bi u tome pomogla mogla bi doći koristeći raznovrsne mjerne uređaje. Izazov je u transportu (prijenosu) informacija s mjernih uređaja prema korisnicima koji ih zahtijevaju. Na sreću taj izazov moguće je svladati uporabom poboljšane telemetrijske tehnologije, isto kao i komunikacijskih protokola koji omogućuju slanje i primanje tih informacija preko Interneta. Ako je mreža nestabilna ili krajnji uređaji imaju slabu procesorsku moć kao što je slučaj s mikroupravljačima, MQTT protokol svojim mehanizmima nudi u većini slučajeva prihvatljivo rješenje prijenosa potrebnih informacija. MQTT komunikacijski protokol stvoren je krajem devedesetih godina prošlog stoljeća (1999. god.). Protokol su razvili Andy Stanford - Clark (IBM) i Arlen Nipper (Arcom, Cirrus, Link) [2], kada su trebali osmisliti protokol za minimalnu potrošnju baterije i rad na nisko propusnoj mreži nadzora naftovoda koristeći satelitsku vezu [3]. Njihovi ciljevi za budući protokol bili su:

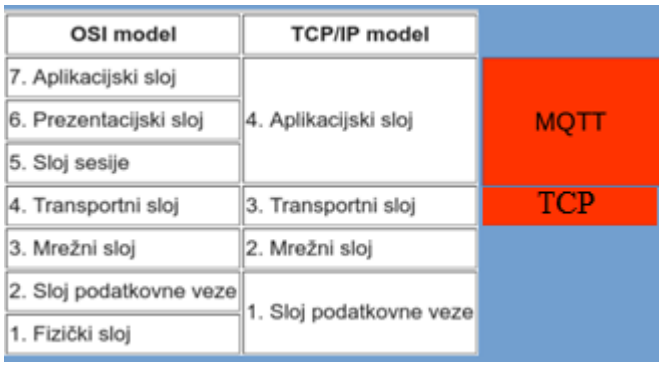
- jednostavno za implementirati,
- osigurati kvalitetu usluge isporuke podataka,
- osigurati lagan i propusno efikasan protokol,
- osigurati svijest o kontinuiranoj sesiji.

Ti ciljevi još su uvijek temelj MQTT protokola, a promijenio se samo fokus sa ciljanog, ograničenog i zatvorenog sustava na sustav koji je otvoren za primjenu, kao što je primjer sa IoT aplikacijama. U početnim poglavljima ovog završnog rada daje se pregled MQTT protokola.

Objašnjeni su pojmovi MQTT klijenta te MQTT brokera, kao i sam princip uspostave veze između klijenta i brokera. Detaljno je razrađen koncept MQTT protokola, kao i format MQTT paketa. U praktičnom dijelu završnog rada demonstriran je MQTT protokol koristeći Android mobilnu aplikaciju preko koje se vrši povezivanje sa MQTT brokerom, kao i kontrola te vizualizacija stanja mikroupravljača. Da bi se utvrdilo i kvalitetno vrednovalo koliko je vremenski potrebno MQTT poruci da dođe sa uređaja na uređaj, u Android aplikaciju implementirana je metoda mjerenja odaziva MQTT brokera.

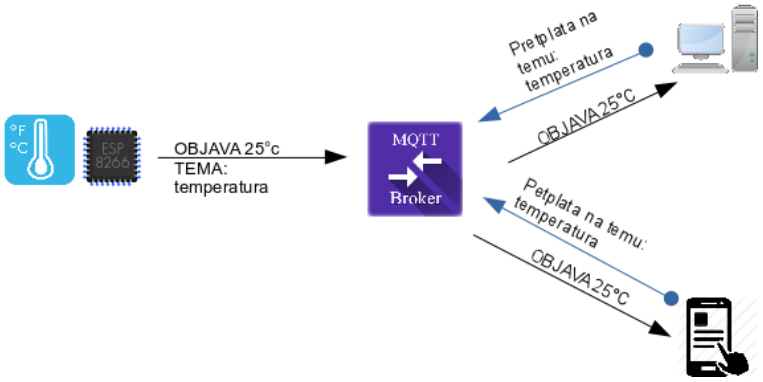
2. MQTT protokol

Autori prve verzije MQTT protokola (1999. god.) su Andy Stanford-Clark i Arlen Nipper. Kasnije 2013. god. IBM izdaje sljedeću verziju (MQTT v3.1), a zatim i verziju (MQTT v3.1.1) koja postaje OASIS (eng. *Organization for the Advancement of Structured Information Standards*) standard. MQTT protokol standardiziran je od strane ISO-a (2016. god. ISO/IEC PRF 20922) [4]. Osim izvornog MQTT protokola nastaje i MQTT-SN verzija koja kao transportni protokol koristi UDP. Prema OSI mrežnom modelu MQTT protokol koristi sloj sesije, prezentacijski i aplikacijski sloj, što odgovara aplikacijskom sloju TCP/IP mrežnog modela, prikazano na slici 2.1. Zapravo, MQTT je protokol aplikacijskog sloja koji se za prijenos podataka oslanja na postojeći TCP transportni protokol.



Slika 2.1 Prikaz MQTT protokola u OSI i TCP/IP mrežnom modelu

Princip rada MQTT protokola prikazan je na slici 2.2 gdje je prikazano da svi uređaji (klijenti) komuniciraju s MQTT brokerom. S jedne strane su klijent (senzor) koji mjeri temperaturu, a s druge strane su klijenti na kojima se vrši vizualizacija tih mjerenja. MQTT protokol koristi metodu objavi/pretplati (eng. *Publish/Subscribe*). Uređaj koji mjeri temperaturu objavljuje (eng. *Publish*) rezultate mjerenja na određenu temu (eng. *Topic*). Klijenti kojima je potrebno dostaviti informacije su pretplaćeni (eng. *Subscribe*) na tu istu temu. Kada je objavljena poruka na temu „temperatura“, klijenti koji su pretplaćeni na temu „temperatura“ dobivaju poruku.



Slika 2.2 Princip rada MQTT protokola

2.1. MQTT klijent

Pod pojmom MQTT klijent podrazumijeva se pretplatnika ili objavitelja, a obojica su MQTT klijenti koji objavljuju ili se pretplaćuju na teme koje su njima „interesantne“ [5]. MQTT klijent može u isto vrijeme biti pretplatnik (pretplatiti se na neku temu) i objavitelj (objavljivati poruke na neku temu). U osnovi MQTT klijentom se može smatrati svaki uređaj od mikroupravljača do računala, odnosno svaki uređaj koji ima implementiranu podršku za rad s MQTT protokolom (MQTT knjižnice i programska podrška) i povezuje se s MQTT brokerom. MQTT klijentske knjižnice su dostupne za velik broj programskih jezika C, C++, C#, Java, Java Script, kao i za platforme iOS i .NET.

2.2. MQTT broker

Posrednik između klijenata je MQTT broker, koji je uglavnom i „srce“ cijelog objavi/pretplati protokola [5]. Ovisno o konkretnoj implementaciji, kao i samim performansama uređaja na kojem je MQTT broker instaliran, MQTT broker može imati mnogo istovremeno aktivnih veza. Takav primjer je Emqtt broker koji može imati i do 1.3 milijuna istovremeno aktivnih veza [6]. Primarne funkcije MQTT brokera su: primanje poruka od svih klijenata koji komuniciraju s njim, „filtriranje“ istih, odlučivanje o interesu za određene poruke te slanje istih „zainteresiranim“ klijentima. Sljedeća važna uloga MQTT brokera je ovjeravanje autentičnosti. U principu MQTT broker može biti svako računalo specifikacijama dovoljno moćno da podrži program MQTT brokera. Danas postoji mnogo takvih programa. Neki od njih su otvorenog koda (eng. *Open Source*), a neki imaju razna ograničenja. Neki od najčešće korištenih i javno dostupnih MQTT brokera prikazani su u tabeli 2.1.

Ime MQTT brokera	**Otvoreni kod**	**Programski jezik**	**Opis**
Mosquitto	+	C	MQTT broker otvorenog koda sa C i C++ klijentskim knjižnicama
Moquette	+	Java	Java implantacija MQTT brokera, jednostavan za instalirat i ugradit, format konfiguracijske datoteke kao i kod Mosquitto MQTT brokera
Emqtt	+	Erlang/OTP	Pruža mnogo proširenja, kao i velik broj aktivnih veza
Hive MQ	-	Java	MQTT broker sa maksimalnom proširivošću, ograničenja kod besplatne licence tipa moguće samo 25 spojenih uređaja, Hive MQ nije dozvoljeno koristiti u komercijalne svrhe.
Mosca	+	Node.js	MQTT broker otvorenog koda, može se koristiti kao samostalan servis, a postoji i mogućnost ugradnje u Node.js aplikaciju

Tabela 2.1 Neki od MQTT brokera s kratkim opisom [7]

Na računalu je moguće instalirati MQTT brokere iz tabele 2.1, no postoje i već „gotova“ rješenja, odnosno web servisi bazirani na MQTT brokerima iz tabele 2.1, koji nude mogućnost spajanja na već instalirane i javno dostupne MQTT brokere, što je pogodno za IoT projekte. Većim dijelom „gotova“ rješenja se naplaćuju, no postoje i besplatni servisi, ali većinom uz razna ograničenja. Tabela 2.2 prikazuje neka od „gotovih“ web rješenja.

Polje	**Vrijednost**
Adresa	**iot.eclipse.org**
Port	1883, 80 (WebSockets), 443 (Websockets+SSL)
Broker	Mosquitto
Info	Web stranica, Xively statistika, teme

Adresa	**test.mosquitto.org**
Port	1883, 8883 (SSL), 8884 (SSL), 80 (WebSockets)
Broker	Mosquitto
Info	Administratorska kontrolna ploča

Adresa	**broker.mqttdashboard.com**
Port	1883, 8000 (WebSockets)
Broker	HiveMQ
Info	Informacijska stranica, prikupljanje statistike te administratorska kontrolna ploča

Adresa	**www.cloudmqtt.com**
Port	18443, 28443 (SSL)
Broker	Mosquitto
Info	Zahtjeva registraciju, korisničko ime/lozinka, cijena (mogućnost besplatnog paketa ali uz ograničenje aktivnih veza)

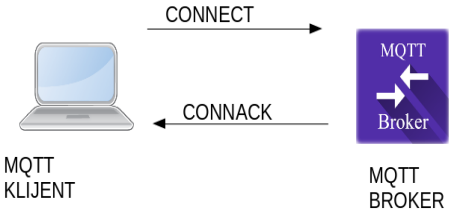
Adresa	**mqtt.dioty.co**
Port	1883 (MQTT), 8883 (MQTT+SSL), 8080 (WebSockets), 8880 (WebSockets+SSL)
Broker	Mosca
Info	Besplatan-zahtjeva registraciju, dokumentacija, sadržava i gotove IoT aplikacije za iOS i Android platforme

Tabela 2.2 Prikaz „gotovih“ web rješenja MQTT brokera [8]

2.3. Uspostava MQTT veze između MQTT klijenta i brokera

Budući da je MQTT protokol baziran na aplikacijskom sloju TCP/IP mrežnog modela, klijenti kao i MQTT broker trebaju imati TCP/IP podršku. MQTT veza uspostavlja se između klijenta i brokera. Ne postoji slučaj u kojem bi klijenti bili povezani bez posrednika (MQTT brokera). Vezu uspostavlja klijent koji šalje **`CONNECT`** MQTT paket MQTT brokeru, koji odgovara s **`CONNACK`** MQTT paketom i statusnim kodom. Kada je veza uspostavljena klijent će ju održavati

otvorenom, na način da će periodično slati **PINGREQ** MQTT pakete MQTT brokeru na koje će broker odgovarati s **PINGRESP** MQTT paketima. Veza će ostati otvorena do trenutka kada klijent ne pošalje naredbu (paket) za prekidom veze, što znači da je do tog trenutka TCP veza otvorena.



Slika 2.3 Uspostava veze između MQTT klijenta i brokera

S obzirom da MQTT koristi TCP na transportnom sloju, a znamo da je TCP specifičan po uspostavi logičke veze između klijenta i poslužitelja, TCP veza između klijenta i MQTT brokera je uvijek otvorena. To omogućuje klijentima da u bilo kojem trenutku objavljuju ili primaju poruke na koje su pretplaćeni. Ako nema razmjene podataka (ništa se ne objavljuje) u određenom vremenskom periodu, klijent će generirati **PINGREQ** (zahtjev za odazivom) na kojega će očekivati odgovor **PINGRESP** od strane MQTT brokera. Razmjena **PINGREQ** i **PINGRESP** paketa znači da je veza otvorena. Vremenski period u kojem se razmjena tih poruka mora dogoditi definiran je vremenskim brojačem „održi na životu“ (eng. *Kepp Alive Timer*). Ovaj princip razmjene **PINGREQ/PINGRESP** paketa u određenom vremenskom intervalu ima još jednu važnu ulogu, a to je izbjegavanje polu-otvorenih veza u slučaju da je došlo do pogreške na mreži.

2.4. Usporedba između HTTP i MQTT protokola

Obzirom na primjenu MQTT, postavlja se pitanje dostupnosti protokola slične namjene. Tehnološki, postoji niz rješenja koja nisu tema ovog rada. S druge strane, obzirom na široku zastupljenost HTTP protokola, u nastavku je dana usporedba ta dva protokola (iako se njihove primjene primarno razlikuju, HTTP se često koristi za dohvat podataka istog tipa). HTTP ima najširu uporabu te je najzastupljeniji protokol. Gotovo sva računala i uređaji koji imaju TCP/IP protokol ga koriste. HTTP i MQTT protokoli koriste TCP/IP grupu protokola na nižim slojevima. HTTP protokol koristi zahtjev/odgovor (eng. *Request/Response*) model, koji je trenutno najčešći protokol za razmjenu poruka. Za razliku od HTTP-a MQTT protokol koristi objavi/pretplati model.

	MQTT	HTTP
Orijentacija na temelju dizajna	Orijentiran prema podacima	Orijentiran prema dokumentima
Metoda funkcioniranja	Objavi/Pretplati	Zahtjev/Odgovor
Veličina poruke	Malene poruke sa kompaktnim binarnim zaglavljem. Najmanja samo 2 bajta.	Velika, djelomično zbog toga što su poruke bazirane na tekstualnom zapisu.
Nivoi usluge	Tri nivoa kvalitete usluge	Sve poruke su na istom nivou
Kompleksnost	Jednostavan	Složen (kompleksan)
Dodatne knjižnice	Knjižnice za C (30KB) i javu (100) KB	Ovisno o aplikaciji (JSON, XML) tipično nisu male
Distribucija podataka	Podržava 0:1; 1:1; 1:n	Samo 1:1

Tabela 2.3 Usporedba MQTT i HTTP protokola [1]

Iz tabele 2.3 primjećuje se da je MQTT protokol orijentiran prema podacima odnosno podaci u MQTT protokolu transportiraju se kao red (eng. *Array*) bajtova. Za razliku od MQTT protokola HTTP je centriran prema dokumentima odnosno podržava MIME standard da bi definirao tip sadržaja, ali ograničeni uređaji tipično ne trebaju tu naprednu značajku. Bitna razlika zapaža se i u metodi slanja i prihvata poruka (njihovom obrascu). Naime MQTT koristi objavi/pretplati obrazac što znači da klijenti ne trebaju biti svjesni postojanja drugih uređaja. Jedino o čemu trebaju voditi brigu je sadržaj koji će biti poslan ili primljen. HTTP koristi zahtjev/odgovor model. To je osnovni i moćan obrazac razmjene poruka. Što se kompleksnosti tiče MQTT je dosta jednostavan. Za razvojne programere bitno je da razumiju samo par MQTT paketa kao što su: **`CONNECT`**, **`PUBLISH`**, **`SUBSCRIBE`**, **`UNSUBSCRIBE`** i **`DISCONNECT`**. HTTP protokol je složeniji. Koristi mnogo povratnih kodova i metoda kao što su POST, PUT, GET, DELETE, HEAD, TRACE, CONNECT itd… Veličina poruka je još jedna razlika između ova dva protokola. Poruke u MQTT protokolu imaju kratko zaglavlje, najmanji paket ima svega 2 bajta. Za razliku od MQTT protokola koji koristi binaran format, HTTP korist tekstualan, što omogućuje duga zaglavlja i poruke. Na primjer osnovna poruka za dohvat nekog sadržaja HTTP-om GET (tekst) je već 3 bajta. HTTP protokol ne nudi kvalitetu usluge, ako je potrebna garancija dostave poruke, razvojni programeri trebaju sami implementirati tu funkciju. Razlike su također vidljive i u distribuciji podataka kao i u dodatnim programskim knjižnicama. Osnovna razlika i samim time prednost u primjeni MQTT protokola je u tome što kod HTTP protokola klijent mora poslati zahtjev uvijek kad želi očitati (dohvatiti) neki podatak, te čekati odgovor. Kod MQTT protokola MQTT klijent ne šalje zahtjev već mu broker servira sve na što je pretplaćen (ne mora pojedinačno slati zahtjeve svaki put kad želi dohvatiti neki podatak).

3. Osnovni koncept MQTT protokola

Cijeli protokol napravljen je tako da koristi nekoliko osnovnih koncepata. Svi su usmjereni prema osiguravanju isporuke poruke.

3.1. Objavi/pretplati

MQTT protokol je baziran na principu objave poruke i pretplate na temu, što se tipično odnosi na objavi/pretplati model. Klijenti se mogu pretplatiti na teme od interesa i time primati sve poruke koje su objavljene odnosno „adresirane“ na te teme. To omogućava da je poruka „adresirana“ na jednu temu dostupna za sve klijente koji su pretplaćeni na tu temu.

3.2. Teme i pretplate

Svaka poruka objavljena korištenjem MQTT protokola „adresirana“ je na određenu temu. Klijenti se pretplaćuju na teme, a zauzvrat primaju poruke koje su „adresirane“ na teme na koje se klijent pretplatio. Pretplate mogu biti eksplicitne što ograničava poruke koje se primaju na određenoj temi. U MQTT brokeru nije potrebno konfigurirati teme prilikom pretplate. Ako tema ne postoji, ona će biti kreirana naknadno prilikom objave poruke. Teme nisu ništa drugo nego nizovi znakova (eng. *string*) kodirani u UTF 8 formatu, pa se mogu koristiti i hrvatska slova. Isto tako teme mogu biti hijerarhijski organizirane. Najbolji primjer za to bio bi posjedovanje više senzora u kući, primjerice da svaka prostorija ima senzor ili više njih koji nešto mjere. Primjer takve teme bi bio:

- **/kuća/soba1/temperatura** ili
- **/kuća/soba2/vlažnost zraka**

Mogu se koristiti zamjenski znakovi (eng. *wildcard*) kao što je npr. brojčani znak (#) što znači da se klijent pretplaćuje na sve teme. Zamjenski znakovi se koriste kod pretplata, a nikako ne kod objava poruka. MQTT protokol koristi dva zamjenska znaka. Jedan od njih je već spomenuti brojčani znak (#), a drugi je plus simbol (+), koji se koristi za jednu razinu hijerarhije, za razliku od (#) koji se koristi za više razina hijerarhije. Primjer takve pretplate bio bi npr. kad se klijent želi pretplatiti na sve senzore vlažnosti zraka u kući:

- **/kuća/+/vlažnost zraka**

Ili možda se klijent želi pretplatiti na senzor temperature u sobi 1:

- **/kuća/soba1/temperatura**

Ako se klijent želi pretplatiti na sve senzore u cijeloj kući definira pretplatu na sljedeći način:

- **/kuća/#**

3.3. Kvaliteta usluge (QoS)

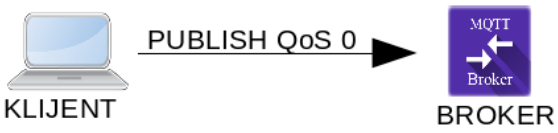
MQTT protokol ima definirane tri razine kvalitete usluge (eng. *Quality of Service,* QoS). Kvaliteta usluge definira koliko će se klijent/broker „truditi" da poruka bude dostavljena. Poruka može biti poslana na bilo kojem nivou kvalitete, isto tako klijent se može pretplatiti na temu bez obzira na kvalitetu usluge. Na primjer, ako je poruka objavljena uz kvalitetu usluge QoS 2, a klijent je pretplaćen s kvalitetom usluge QoS 0, poruka će biti dostavljena klijentu, ali uz QoS 0. Ako postoji klijent koji je pretplaćen na tu istu temu, ali s QoS 2, poruka će biti dostavljena uz QoS 2. Drugi primjer je, ako je klijent pretplaćen sa QoS 2, a poruka je objavljena uz QoS 0, klijent će primiti poruku, ali uz QoS 0. Što je veći nivo usluge, veća je i vjerojatnost dostave poruke. Korištenjem viših nivoa kvalitete usluge zbog povećanja složenosti postupka dostave podataka, očekuje se povećanje kašnjenja u isporuci podataka, a isto tako povećava se i količina podataka koji se prenose (veći zahtjevi vezani uz propusnost mreže). U tabeli 3.1 prikazan je popis razina kvalitete usluge te opis istih.

Nivo kvalitete usluge (QoS)	Opis
0	Broker/klijent će jednom poslati poruku bez potvrde
1	Broker/klijent će poslati poruku barem jednom sa potrebom potvrde
2	Broker/klijent će poslati poruku točno jednom ali koristeći četverostruko rukovanje (eng. *four step handshake*)

Tabela 3.1 Opis kvalitete usluge (QoS) u MQTT protokolu [9]

3.3.1. Razmjena poruka na razini kvalitete usluge QoS 0

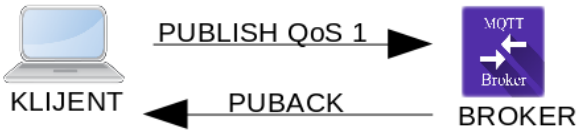
Minimalna kvaliteta usluge je 0 (QoS 0). Na ovoj razini poruka neće biti potvrđena od strane primatelja, kao ni spremljena te natrag poslana od strane pošiljatelja. Ova razina kvalitete usluge još se naziva „Pošalji i zaboravi" (eng. *Fire and Forget*). Tipično ovo je najbrža metoda ali i nepouzdana. Poruka će biti poslana samo jednom, a primljena jednom ili nijednom Primjer razmjene poruke na razini QoS 0 prikazan je na slici 3.1, gdje klijent šalje **`PUBLISH`** paket MQTT brokeru.



Slika 3.1 Razmjena poruke između MQTT klijenta i brokera sa kvalitetom usluge 0

3.3.2. Razmjena poruka na razini kvalitete usluge QoS 1

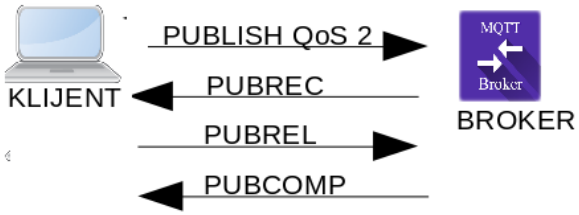
Korištenje kvalitete usluge 1 (QoS 1) je garancija da će poruka biti dostavljena primatelju barem jednom, ali isto tako može se dogoditi da poruka bude primljena više puta. Klijent šalje poruku i čeka potvrdu (eng. *acknowledgement*) **PUBACK** paket. Kada klijent koji je poslao poruku primi potvrdu, on može obrisati poruku koju je poslao (pošiljatelj prije samog slanja pohranjuje poruku), zbog toga da u slučaju ako ne primi potvrdu da je poruka primljena može ponovno poslati poruku.) Ako pošiljatelj ne primi potvrdu on će ponovno poslati poruku, s tim da će postaviti zastavicu dupliciranja (eng. *DUP flag*). Pošiljatelj će poruku slati tako dugo dok ne primi potvrdu **PUBACK**. Kada broker zaprimi poruku on će ju proslijediti svim klijentima koji su pretplaćeni na temu na koju je poruka objavljena, čak i onda kada je zastavica dupliciranja postavljena. Zbog toga se može dogoditi da ista poruka bude poslana više puta.



Slika 3.2 Razmjena poruke između MQTT klijenta i brokera sa kvalitetom usluge 1

3.3.3. Razmjena poruka na razini kvalitete usluge QoS 2

Ovo je najviša kvaliteta usluge koju MQTT protokol podržava, ona garantira da će svaka poruka biti poslana samo jednom. Ovo je najsigurnija kvaliteta usluge, ali ujedno i najsporija.



Slika 3.3 Razmjena poruke između MQTT klijenta i brokera sa kvalitetom usluge 2

Ova metoda se odvija u četiri koraka. Prvo pošiljatelj šalje poruku, sprema referencu na nju i čeka potvrdu, odnosno **PUBREC** MQTT paket. Kada primatelj primi poruku šalje **PUBREC** paket pošiljatelju. Ako pošiljatelj ne primi **PUBREC** od primatelja, pošiljatelj je dužan ponovno poslati poruku sa postavljenom zastavicom dupliciranja. Kada pošiljatelj zaprimi potvrdu **PUBREC** odmah šalje **PUBREL**. Ako se dogodi da primatelj ne primi **PUBREL** on će ponovno poslati **PUBREC** MQTT paket. Kada primatelj primi **PUBREL** paket to znači da može proslijediti tu poruku drugim pretplaćenim klijentima. Time je osigurano da ista poruka ne bude poslana više puta, kao što je slučaj kod QoS 1. Nakon što primatelj zaprimi **PUBREL** paket, pošiljatelju šalje **PUBCOMP** paket što je četvrti i zadnji paket na najvišoj razini kvalitete usluge. Kada pošiljatelj zaprimi **PUBCOMP** on može izbrisati prethodno spremljenu referencu na poruku.

3.4. Zadržane poruke

Zadržana poruka je normalna MQTT poruka sa postavljenom zastavicom zadržavanja. MQTT broker će spremiti zadnju zadržanu poruku. Svaki klijent koji se pretplati na temu u kojoj postoji zadržana poruka odmah će ju primiti. Za svaku temu samo jedna zadržana poruka biti će spremljena od strane brokera. U suštini zadržavanje poruke može pomoći novo pretplaćenim klijentima da se brže ažuriraju, u slučaju da postoji poruka koja se rijetko objavljuje. Jednom kada MQTT broker pohrani zadržanu poruku, jedini način da se ona ukloni je da se objavi nova zadržana poruka, ili da se objavi prazna poruka sa postavljenom zastavicom zadržavanja.

3.5. Čista sesija

Prilikom povezivanja klijent može postaviti zastavicu za čistu sesiju (eng. *Clean Session*), koja je još poznata kao zastavica koja označuje čisti start. Ako ta zastavica nije postavljena veza se tretira kao trajna. To znači da će nakon što klijent prekine vezu, sve njegove pretplate biti spremljene od strane MQTT brokera. Osim samih pretplata MQTT broker će pohranjivati poruke objavljene na razini usluge Qos 1 ili 2, da bi mogle biti ponovno poslane nakon ponovnog spajanja klijenta sa MQTT brokerom. U slučaju da je zastavica postavljena, nakon što klijent prekine vezu, sve pretplate će biti uklonjene [1].

3.6. Oporuke

Mogućnost postavljanja oporuka (eng. *Will*) je još jedna važna funkcija MQTT brokera. Prilikom spajanja klijenta s MQTT brokerom, klijent može postaviti zastavicu oporuke. Za MQTT brokera to znači da klijent ima definiranu poruku koju će broker automatski objaviti ako dođe do neočekivanog ispada (prekida veze) između klijenta i brokera. Poruka sa oporukom ima

temu, kvalitetu usluge, status zadržavanja isto kao i svaka druga poruka. Prilikom postavljana oporuke klijent mora definirati temu oporuke kao i poruku oporuke, odnosno parametre kao i za svaku normalnu poruku koja se objavljuje. Poruka oporuke se jedino razlikuje po tome što MQTT broker prepoznaje prekinutu vezu, te automatski objavljuje poruku koju je klijent postavio. Tu poruku primit će svi klijenti koji su pretplaćeni na temu oporuke odnosno (temu koju definira klijent koji postavlja oporuku). Ova funkcija je korisna u slučaju ispada klijenta iz mreže (neočekivanog prekida veze) jer pruža mogućnost da ostali klijenti budu obaviješteni prilikom ispada nekog klijenta.

4. Struktura MQTT paketa

MQTT protokol funkcionira tako da razmjenjuje niz MQTT paketa na predefiniran način. U osnovi MQTT paket sastoji se od tri dijela: fiksnog zaglavlja, varijabilnog zaglavlja i tereta (eng. *Payload*), odnosno sadržaja MQTT poruke. Prikaz strukture (formata) MQTT paketa prikazan je u tabeli 4.1.

Fiksno zaglavlje, prisutno u svim MQTT paketima
Varijabilno zaglavlje, prisutno u nekim MQTT paketima
Teret, prisutan u nekim MQTT paketima

Tabela 4.1 Struktura MQTT paketa [10]

4.1. Fiksno zaglavlje MQTT kontrolnog paketa

Fiksno zaglavlje prikazano je u tabeli 4.2

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip MQTT paketa				Specifične zastavice za svaki tip MQTT paketa			
Bajt 2	Duljina ostatka paketa							

Tabela 4.2 Struktura fiksnog zaglavlja MQTT paketa [10]

Iz tabele 4.2 vidi se da fiksno zaglavlje MQTT paketa sadržava tip MQTT paketa za čiju se identifikaciju koriste 4 bita te polje koje sadržava specifične zastavice MQTT paketa za što se isto koriste 4 bita.

4.1.1. Tip MQTT paketa

Tip MQTT paketa definiraju prva četiri najznačajnija bita prvog bajta fiksnog zaglavlja. Tip MQTT paketa je reprezentiran kao vrijednosti od 4 bita. U tabeli 4.3 prikazan je opis svih tipova MQTT paketa.

Ime	**Vrijednost**		**Smjer**	**Opis**
	Decimalno	Binarno		
Rezervirano	0	0000	Zabrana	Rezervirano
CONNECT	1	0001	Klijet => broker	Zahtjev za spajanje na broker
CONNACK	2	0010	Broker => klijent	Potvrda spajanja
PUBLISH	3	0011	Dvosmjerno	Objava poruke
PUBACK	4	0100	Dvosmjerno	Potvrda objave
PUBREC	5	0101	Dvosmjerno	Objavljivanje primljeno (osigurana isporuka dio 1)

PUBREL	6	0110	Dvosmjerno	Puštanje objave (osigurana isporuka dio 2)
PUBCOMP	7	0111	Dvosmjerno	Objava je kompletna (osigurana isporuka dio 3)
SUBSCRIBE	8	1000	Klijent => broker	Zahtjev klijenta za pretplatu
SUBBACK	9	1001	Broker => klijent	Potvrda pretplate
UNSUBSCRIBE	10	1010	Klijent => broker	Zahtjev za otkazivanje pretplate
UNSUBACK	11	1011	Broker => klijent	Potvrda za otkaz pretplate
PINGREQ	12	1100	Klijent => broker	Zahtjev za odazivom
PINGRESP	13	1101	Broker => klijent	Odgovor na odaziv
DISCONNECT	14	1110	Klijent => broker	Klijent prekida vezu
Rezervirano	15	1111	zabrana	Rezervirano

Tabela 4.3 Tipovi MQTT paketa [10]

4.1.2. Zastavice

Tabela 4.2 prikazuje da četiri najmanje značajna bita prvog bajta u fiksnom zaglavlju MQTT paketa sadržavaju zastavice koje su specifične za svaki tip MQTT kontrolnog paketa. U tabeli 4.4 prikazani su bitovi zastavica (eng. *Flag_Bits)*. Bit koji je označen s „rezervirano“ znači da je rezerviran za buduću uporabu te mora biti postavljen na vrijednost kako je prikazano u tabeli 4.4. Ako je zastavica pogrešno postavljena, primatelj iste mora prekinuti mrežnu vezu.

MQTT paket	Zastavica fiksnog zaglavlja	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Rezervirano	0	0	0	0
CONNACK	Rezervirano	0	0	0	0
PUBLISH	Upotreba u MQTT verziji 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Rezervirano	0	0	0	0
PUBREC	Rezervirano	0	0	0	0
PUBREL	Rezervirano	0	0	1	0
PUBCOMP	Rezervirano	0	0	0	0
SUBSCRIBE	Rezervirano	0	0	1	0
SUBACK	Rezervirano	0	0	0	0
UNSUBSCRIBE	Rezervirano	0	0	1	0
UNSUBACK	Rezervirano	0	0	0	0
PINGREQ	Rezervirano	0	0	0	0
PINGRESP	Rezervirano	0	0	0	0
DISCONNECT	Rezervirano	0	0	0	0

Tabela 4.4 Bitovi zastavica u fiksnom zaglavlju MQTT paketa [10]

U tabeli 4.4 nalaze se oznake koje označavaju:

- DUP = dvostruka dostava **PUBLISH** kontrolnog paketa;
- QoS = **PUBLISH** kvaliteta usluge;
- RETAIN = **PUBLISH** zastavica zadržavanja (eng. *Retain Flag*).

4.1.3. Princip formatiranja MQTT paketa

Duljina ostatka paketa je broj bajta preostalih u paketu, uključujući podatke u varijabilnom zaglavlju i teretu. U tu duljinu nisu uključeni bajtovi korišteni za njeno kodiranje. Duljina ostatka paketa kodira se uporabom sheme varijabilne dužine, koja koristi jedan bajt za kodiranje vrijednost do 127. S većim vrijednostima postupa se na sljedeći način. Najmanje značajnih 7 bitova od svakog bajta kodiraju podatak, a najznačajniji bit se koristi kao indikator da ima još bitova koje treba kodirati. Tako svaki bajt kodira vrijednost do 128 i nastavni bit. Duljina ostatka paketa može maksimalno sadržavati 4 bajta. Takvi primjeri su:

- primjer kodiranja na dekadskoj vrijednosti 64:
  Za tu vrijednost koristi se samo jedan bajt;
- primjer kodiranja na dekadskoj vrijednosti 321:
  321 = (65+2*128) vrijednost je kodirana koristeći 2 bajta od kojih je najmanje značajan prvi. Prvi bajt 65+128=193. Najveći bit je postavljen kao nastavni bit koji indicira da je preostalo još bitova za kodiranje. Nakon njega slijedi drugi bit.
  To omogućuje aplikacijama slanje MQTT paketa do 256 MB.

U tabeli 4.5 prikazano je kodiranje duljine ostatka paketa.

Bajt	Od	Do
1	0(0x00)	127(0x7F)
2	128 (0x80, 0x01)	16383 (0xFF, 0x7F)
3	16384 (0x80, 0x80, 0x01)	2 097151 (0xFF, 0xFF, 0x7F)
4	2097152 (0x80, 0x80, 0x80, 0x01)	256435455 (0xFF, 0xFF, 0xFF, 0x7F)

Tabela 4.5 Kodiranje duljine ostatka paketa [10]

4.2. Varijabilno zaglavlje MQTT paketa

Neki tipovi MQTT paketa uz fiksno sadrže i varijabilno zaglavlje. Varijabilno zaglavlje nalazi se između fiksnog zaglavlja i tereta. To zaglavlje ne sadrži polje za preostalu dužinu.
Varijabilno zaglavlje sadrži sljedeća polja:

- **ime protokola** – prisutno u varijabilnom zaglavlju prilikom uspostave veze (**`CONNECT`** paket);
- **verzija protokola –** prisutno prilikom uspostave veze (**`CONNECT`** paket);
- **vremenski brojač: „**održi na životu“ (eng. *Keep Alive Timer*) – prisutan u varijabilnom zaglavlju prilikom povezivanja. Brojač mjeri vrijeme u sekundama te definira maksimalan vremenski interval, odnosno dopušteno vrijeme između slanja i primanja MQTT paketa. To omogućava MQTT brokeru da otkrije dali je mrežna veza s klijentom prekinuta bez čekanja relativno duge TCP/IP pauze (eng. *TCP/IP Timeout*). Klijent ima odgovornost za slanje poruke unutar svakog održi na životu intervala. Ako poruka ne dođe u tom intervalu, klijent šalje **`PINGREQ`** paket, na koji broker odgovara s **`PINGRESP`** paketom. Ako broker ne dobije taj paket od klijenta u vremenu 1.5 puta većem od perioda održi na životu, isključuje klijenta odnosno detektira polu otvorenu vezu, čime se aktivira oporuka. Ako klijent ne primi **`PINGRESP`** (odgovor na **`PINGREQ`**) paket u vremenskom intervalu održi na životu, klijent zatvara TCP/IP utičnicu (eng. *TCP/IP socket*);
- **povratni kod prilikom povezivanja:** kod poslan prilikom povezivanja, točnije to polje je prisutno u varijabilnom zaglavlju **`CONNACK`** kontrolnog paketa. Polje je definirano kao vrijednost od jednog bajta, koja definira povratni kod. Značenje vrijednosti povratnog koda prikazana su u tabeli 4.6;

Vrijednost	Heksadecimalna vrijednost	Opis
0	0x00	Veza prihvaćena
1	0x01	Veza nije prihvaćena: neprihvatljiva verzija protokola
2	0x02	Veza nije prihvaćena: identifikator je odbijen
3	0x03	Veza nije prihvaćena: server nije dostupan
4	0x04	Veza nije prihvaćena: pogrešno korisničko ime ili lozinka
5	0x05	Veza nije prihvaćena: nije odobreno
6-255		Rezervirano za buduću uporabu

Tabela 4.6 Opis vrijednosti povratnog koda [10]

- **naziv teme –** polje je prisutno u varijabilnom zaglavlju **`PUBLISH`** kontrolnog paketa. Naziv teme je ključ koji identificira kanal informacije prema kojemu je sadržaj poruke objavljen. Pretplatnici koriste taj ključ da bi identificirali kanal sa kojega žele primati objavljene poruke. U varijabilnom zaglavlju također su prisutne zastavice za povezivanje (eng. *Connect Flags*). Ovdje spadaju zastavice kao što su: čista sesija, oporuke, oporuke QoS, retain te zastavice za korisničko ime/lozinku. Sve ove zastavice prisutne su u varijabilnom zaglavlju **`CONNECT`** kontrolnog MQTT paketa.

4.2.1. Zastavica čiste sesije

Ako zastavica čiste sesije (eng. *Clean Session Flag*) nije postavljena, broker mora pohraniti pretplate klijenta nakon što klijent prekine vezu. To uključuje i nastavak pohranjivanja poruka s kvalitetom usluge QoS 1, QoS 2 za pretplaćene teme kako bi se poruke mogle isporučiti kada se klijent ponovno poveže. MQTT broker također treba održavati takozvano slanje poruke „u letu“ (eng. *In-flight*) koje se isporučuju na mjesta gdje je veza prekinuta. Te se informacije moraju čuvati do trenutka kada se klijent ponovno poveže. U slučaju da je zastavica postavljena MQTT broker mora odbaciti sve prijašnje održavane informacije o klijentu i tretirati vezu kao „čista“. Obično, klijent će raditi u jednom od dva načina rada i neće se mijenjati. Izbor naravno ovisi o aplikaciji (primjeni MQTT protokola). Klijent s postavljenom zastavicom čiste sesije neće dobivati stalne podatke i mora se ponovno pretplatiti svaki put kada se povezuje. Poruke koje su objavljene sa QoS 0 nikada se ne spremaju. Tabela 4.7 prikazuje format zastavice čiste sesije.

Bit	7	6	5	4	3	2	1	0
	User Name zastavica	*Password* zastavica	*Will* Retain	*Will QoS*		*Will* zastavica	*Clean session*	Rezervirano
	x	x	x	x	x	x		X

Tabela 4.7 Zastavica čiste sesije u varijabilnom zaglavlju MQTT paketa (polje koje nije označeno sa „X“) [10]

4.2.2. Zastavica oporuke

Zastavica oporuke (eng. *Will Flag*) definira dali je poruka objavljena u korist klijenta od strane MQTT brokera kada je naišao na ulaznu/izlaznu grešku prilikom komunikacije s klijentom ili je klijent zakazao komunicirati u intervalu kojeg je definirao vremenski brojač „održi na životu“. Slanje poruke s oporukom nije izazvano od strane brokera prilikom primanja poruke o prekidu veze (**`DISCONNECT`** paket) od strane klijenta. Ako je zastavica postavljena, polje kvalitete usluge oporuke (eng. *Will QoS*) i zadržavanje oporuke mora biti prisutno prilikom povezivanja te polje „tema oporuke“ mora biti prisutno u teretu (sadržaju) poruke. Format ove zastavice prikazan je u tabeli 4.8.

Bit	7	6	5	4	3	2	1	0
	User Name zastavica	*Password* zastavica	*Will Retain*	*Will QoS*		*Will* zastavica	*Clean session*	Rezervirano
	x	x	x	x	x		x	X

Tabela 4.8 Zastavica oporuke u varijabilnom zaglavlju MQTT paketa (polje koje nije označeno sa „X“) [10]

4.2.3. Zastavica kvalitete usluge oporuke

Klijent prilikom povezivanja s MQTT brokerom navodi nivo kvalitete usluge u polju kvalitete usluge oporuke (eng. *Will QoS*), namijenjenim za poruku s oporukom koja se šalje u slučaju dolaska neočekivanog prekida veze klijenta. Poruka s oporukom je definirana u sadržaju **CONNECT** MQTT paketa. Ako je zastavica "oporuke" (eng. *Will flag*) postavljena, onda je polje kvaliteta usluge oporuke obavezno, u suprotnom to polje se zanemaruje.

Bit	7	6	5	4	3	2	1	0
	User Name zastavica	*Password* zastavica	*Will Retain*	*Will QoS*		*Will* zastavica	*Clean session*	Rezervirano
	x	x	x				x	X

Tabela 4.9 Zastavica kvalitete usluge oporuke (eng. Will QoS) u varijabilnom zaglavlju MQTT paketa [10]

4.2.4. Zastavica zadržavanja oporuke

Zastavica zadržavanje oporuke (eng. *Will Retain*) ukazuje na to dali MQTT broker treba zadržati poruku oporuke, koju MQTT broker objavljuje u ime klijenta u slučaju da klijent neočekivano prekine vezu. Zastavica za zadržavanje oporuke je obavezna, ako je zastavica oporuke (eng. *Will Flag*) postavljena, inače je zanemarena. Format ove zastavice prikazan je u tabeli 4.10.

Bit	7	6	5	4	3	2	1	0
	User Name zastavica	*Password* zastavica	*Will Retain*	*Will QoS*		*Will* zastavica	*Clean session*	Rezervirano
	x	x		x	x		x	X

Tabela 4.10 Zastavica zadržavanja oporuke u varijabilnom zaglavlju MQTT paketa [10]

4.2.5. Zastavice za korisničko ime i lozinku

Klijent prilikom povezivanja s MQTT brokerom može odrediti korisničko ime i lozinku, ako su bitovi zastavica za korisničko ime/lozinku *(*eng. *User Name and Password Flags*) postavljeni to znači da su korisničko ime i lozinka uključeni u teret (eng. *Payload*) **CONNECT** paketa. **CONNECT** paket je prvi paket poslan prema brokeru od strane klijenta prilikom uspostave veze. Tabela 4.11 prikazuje polja zastavice za korisničko ime i lozinku.

Bit	7	6	5	4	3	2	1	0
	User Name zastavica	*Password* zastavica	*Will Retain*	*Will QoS*		*Will* zastavica	*Clean session*	Rezervirano
			x	x	x	x	x	X

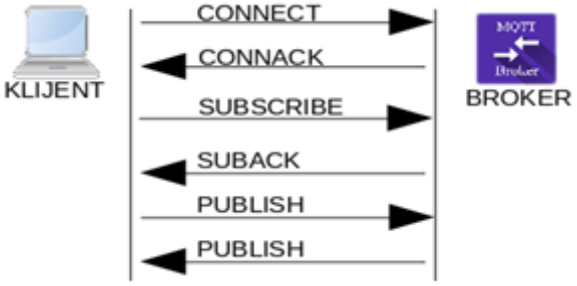
Tabela 4.11 Zastavice za korisničko ime i lozinku u varijabilnom zaglavlju MQTT paketa [10]

4.3. Teret

Teret (eng. *Payload*) odnosno sadržaj MQTT poruke čini važan dio MQTT paketa. Prilikom komunikacije između klijenta i brokera, teret varira tj. samo neki od MQTT paketa sadrže teret. Prilikom povezivanja klijenta s MQTT brokerom veza se uspostavlja na način da klijent MQTT brokeru šalje **`CONNECT`** MQTT paket. Taj paket sadrži i teret, odnosno teret sadrži jedan ili više UTF-8 kodiranih nizova. Oni sadrže jedinstven identifikator za klijenta, temu oporuke, poruku oporuke, korisničko ime i lozinku, ako se koristi (ovisi o postavljenim zastavicama u varijabilnom zaglavlju MQTT paketa). Nadalje kad je veza između klijenta i MQTT brokera uspostavljena, klijent će se pokušati pretplatiti na temu na način da će MQTT brokeru poslati **`SUBSCRIBE`** paket. Taj paket će isto sadržavati teret u kojem će se nalaziti popis naziva tema na koje se klijent želi pretplatiti. Isto tako sadržavat će razine kvalitete usluge (eng. *QoS level*). Nakon što MQTT broker od klijenta primi **`SUBSCRIBE`** paket, na njega odgovara s potvrdom pretplate odnosno klijentu šalje **`SUBACK`** paket. **`SUBACK`** također sadrži teret u kojem se nalazi popis garantiranih razina usluge. Garantirane razine usluge su popisane istim redoslijedom kao i nazivi tema. Kada klijent želi objaviti neku poruku MQTT brokeru šalje **`PUBLISH`** paket sa teretom, koji sadrži specifične podatke vezane za aplikaciju.

5. Standardni paketi u MQTT protokolu

Za razliku od većine klasičnih aplikacijskih protokola uobičajenih u TCP/IP koji prenose upravljačke informacije u tekstualnom obliku, kontrolni zapisi u MQTT su zapisani „kodirano“ u binarnom obliku. Komunikacija između klijenta i brokera odvija se slanjem MQTT paketa.



Slika 5.1 Komunikacija između klijenta i brokera korištenjem MQTT paketa

Iz slike 5.1 može se vidjeti korištenje paketa u MQTT protokolu. Veza je započela tako da je klijent MQTT brokeru poslao **CONNECT** paket na koji je broker odgovorio sa **CONNACK** paketom, odnosno odgovorom na **CONNECT** paket. Iz slike 5.1 također se vidi da isti princip vrijedi za pretplatu na temu (**SUBSCRIBE** paket) i objavu poruke(**PUBLISH** paket). Ovo su samo neki od MQTT paketa. U sljedećim poglavljima MQTT paketi su minimalno razrađeni, detaljnije su objašnjeni u priloženoj dokumentaciji.

5.1. CONNECT i CONNACK paketi

Kada je TCP/IP veza uspostavljena mora se stvoriti sesija na razini MQTT protokola. Sesija se stvara na način da klijent MQTT brokeru šalje **CONNECT** paket, odnosno zahtjev za uspostavu veze. Na tu poruku MQTT broker odgovara s **CONNACK** paketom. **CONNACK** paket će između ostalih stvari u svom varijabilnom zaglavlju imati takozvani povratni kod. Povratni kod prezentiran je sa drugim bajtom u varijabilnom zaglavlju **CONNACK** paketa.

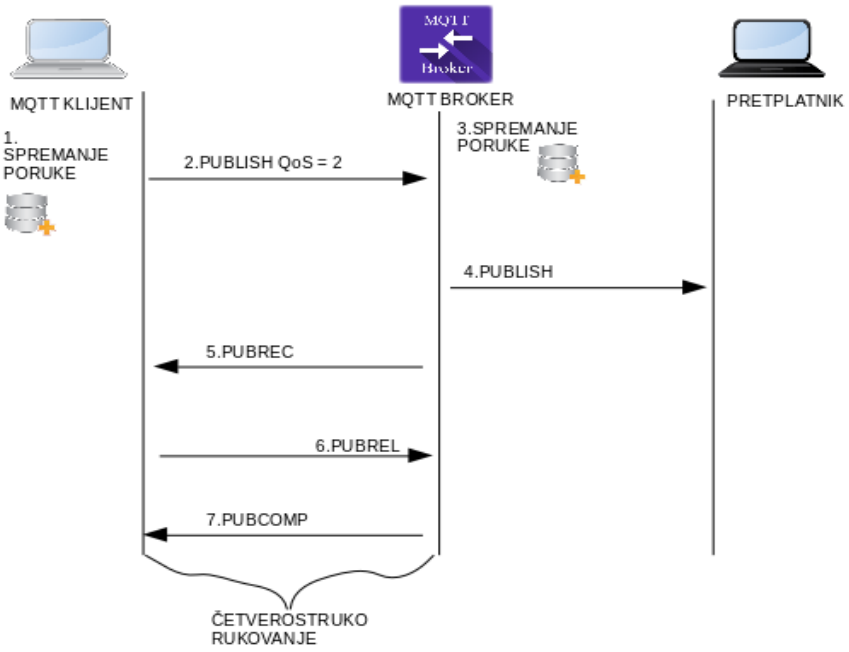
5.2. PUBLISH i PUBACK paketi

PUBLISH paket šalje klijent MQTT brokeru kada želi objaviti neku poruku. Svaka objavljena poruka je povezana nazivom teme (adresirana je na neku temu). Ako klijent traži višu razinu kvalitete usluge (QoS 1), na tu poruku MQTT broker odgovara s **PUBACK** paketom.

5.2.1. PUBREC, PUBREL, PUBCOMP paketi

Ovi paketi namijenjeni su za najvišu razinu kvalitete usluge (QoS 2) uz **PUBLISH** paket iz gornjeg poglavlja 5.2. **PUBREC** paket je odgovor na **PUBLISH** paket, ali s kvalitetom usluge QoS 2. Ovaj paket poslan je od strane MQTT brokera kao odgovor na **PUBLISH** paket koji je objavio

objavitelj (eng. *Publisher*) odnosno klijent. Ali isto tako može biti poslan od strane pretplatnika (eng. *Subscriber*) kao odgovor na **PUBLISH** paket kojeg šalje MQTT broker. **PUBREL** paket je odgovor poslan ili od strane objavitelja na **PUBREC** paket koji je poslan sa MQTT brokera, ili od MQTT brokera na **PUBREC** paket od pretplatnika. Nadalje **PUBCOMP** paket je ili odgovor MQTT brokera na **PUBREL** paket koji je poslan od strane klijenta koji ga je objavio ili je odgovor pretplaćenog klijenta na **PUBREL** paket od strane MQTT brokera. To je četvrti i zadnji paket najviše razine kvalitete usluge (QoS 2) i s njom završava četverostruko rukovanje. Primjer komunikacije u MQTT protokolu gdje se koristi QoS 2 prikazan je na slici 5.2.



Slika 5.2 Primjer komunikacije koristeći najvišu kvalitetu usluge (QoS 2)

5.3. SUBSCRIBE i SUBACK paketi

SUBSCRIBE paket omogućuje klijentu da registrira svoj interes na jednu ili više tema. Teret **SUBSCRIBE** paketa sadržava popis tema na koje se klijent želi pretplatiti. Ova poruka također specificira razinu kvalitete usluge na kojoj klijent želi biti pretplaćen na neku temu. **SUBACK** paket šalje MQTT broker klijentu da bi potvrdio primitak **SUBSCRIBE** poruke. **SUBACK** poruka sadržava popis garantiranih razina kvalitete usluge.

5.4. UNSUBSCRIBE i UNSUBBACK paketi

Ovi paketi koriste se kada klijent više ne želi biti pretplaćen na neku temu. **UNSUBSCRIBE** paket upućen je od strane klijenta MQTT brokeru kada klijent više ne želi biti pretplaćen na neku temu. **UNSUBACK** paket šalje MQTT broker klijentu kao potvrdu da je primio njegov zahtjev za prekid pretplate.

5.5. PINGREQ i PINGRESP paketi

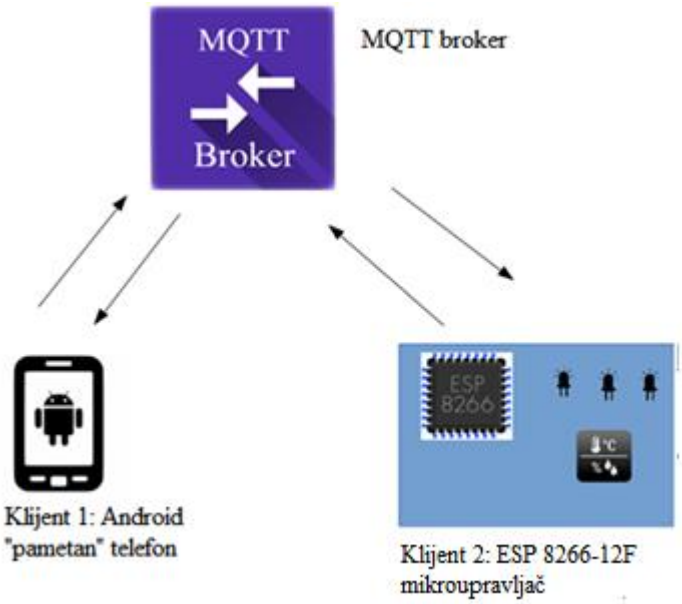
Da bi veza bila otvorena klijent će periodično slati **PINGREQ** (zahtjev za odazivom) MQTT brokeru na što će mu MQTT broker, ukoliko veza nije prekinuta, odgovoriti s **PINGRESP** paketom. Ukoliko klijent ne primi **PINGRESP** od MQTT brokera, klijent će prekinuti vezu.

5.6. DISCONNECT paket

Ovaj paket obavještava MQTT brokera da klijent prekida vezu. Paket šalje klijent MQTT brokeru kao obavijest da će prekinuti vezu. Ovo je vrlo važno zbog toga jer bez ovog paketa MQTT broker ne bi bio svjestan dali je klijent htio prekinuti vezu ili se dogodio neki kvar pa je veza prekinuta. Ta „svijest“ dolazi do posebne važnost kod oporuka.

6. Demonstracija MQTT protokola

U sljedećim poglavljima daje se demonstracija MQTT protokola odnosno implementacija MQTT protokola koristeći mikroupravljač i Android mobilnu aplikaciju kao klijente koji će međusobno komunicirati koristeći MQTT protokol. Cilj demonstracije je kroz Android aplikaciju kontrolirati (uključivati/isključivati) tri LED diode koje su spojene na ulazno/izlazne pinove mikroupravljača te dohvaćati podatke o temperaturi i vlazi, koje će očitavati senzor spojen na mikroupravljač.



Slika 6.1 Koncept izvedenog sustava za demonstraciju MQTT protokola

Sustav funkcionira na način da dva klijenta (Android „pametan“ telefon i ESP 8266-12F mikroupravljač) koristeći MQTT broker instaliran na računalu razmjenjuju poruke. Mobilna aplikacija pomoću koje telefon komunicira s MQTT brokerom u kontekstu MQTT protokola je programirana tako da objavljuje poruke na teme „LED“ i „Odaziv“. Aplikacija također vrši pretplatu na teme „Temperatura“ i „Vlaga“. Program mikroupravljača pretplaćuje se na temu „LED“, gdje će dobivati poruke koje se objavljuju putem Android aplikacije, te na temelju sadržaja tih poruka uključivati/isključivati LED diode. Mikroupravljač će također objavljivati poruke u čijem sadržaju su informacije sa senzora, odnosno temperatura i vlažnost zraka. Te poruke mikroupravljač objavljuje na teme „Temperatura“ i „Vlaga“ na koje je pretplaćena Android aplikacija. U Android aplikaciju također je implementirana vlastita metoda mjerenja odaziva MQTT brokera.

6.1. Instalacija MQTT brokera

Da bi klijenti međusobno mogli komunicirati koristeći MQTT protokol potreban je posrednik (MQTT broker). U ovom odlomku prikazan je postupak instalacije „Mosquitto“ MQTT brokera

na Linux operativnom sustavu. On može biti instaliran i na Windows operativnom sustavu. Operativni sustav Linux odabran je zbog više razloga. Jedan od tih razloga je i cijena (Linux je operativni sustav otvorenog koda). Mosquitto MQTT broker je program otvorenog koda u kojeg je implementirana verzija 3.1.1 MQTT protokola. Nakon same instalacije Mosquitto MQTT brokera važno je postaviti autentifikaciju korisnika. Za analizu MQTT prometa na mrežnom sučelju korišten je program Wireshark, s kojim je napravljena analiza MQTT paketa.

6.2. Postavljanje u lokalnoj mreži

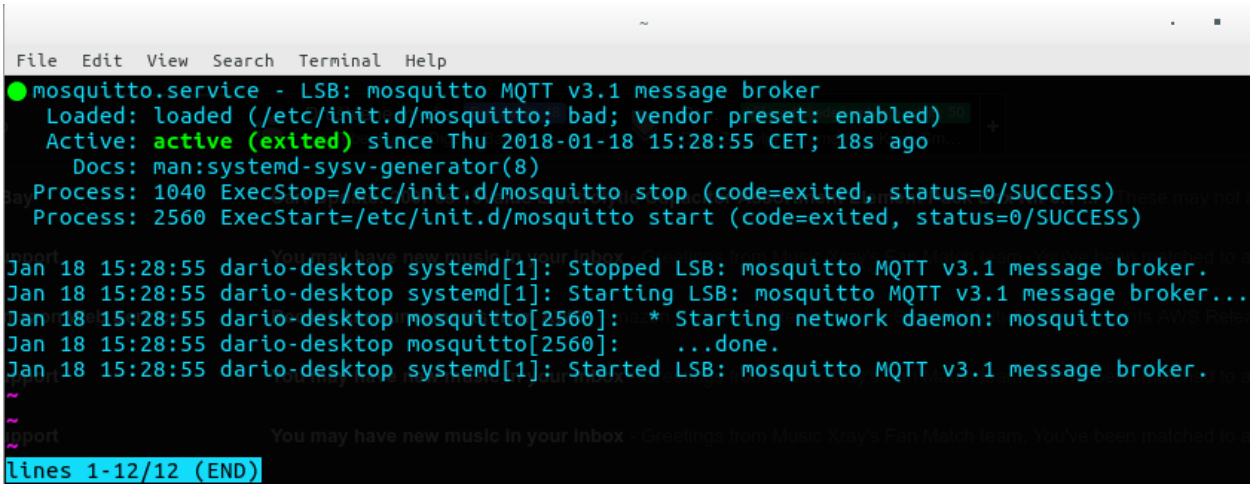
U lokalnoj mreži korištena je Mint distribucija Linux-a. Instalaciju vršimo pomoću Linux komandnog sučelja terminal (eng. *Shell*) tako da se u komandno sučelje upiše naredba:

```
sudo apt-get install mosquitto mosquitto-clients
```

Ova naredba instalirat će Mosquitto MQTT broker sa klijentskim alatima koji će biti korišteni za provjeru ispravnost Mosquitto MQTT brokera. Nakon instalacije Mosqutto servis trebao bi se automatski pokrenuti. Stanje Mosquitto servisa provjerava se upisom sljedeće naredbe u terminal.

```
sudo service mosquitto status
```

Ako je servis pokrenut, naredba bi trebala imati sljedeći odaziv:



Slika 6.2 Odaziv naredbe "sudo service mosquitto status"

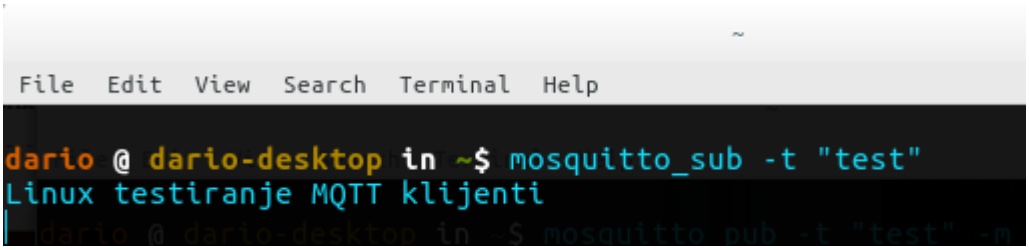
U testiranju Mosquitto MQTT brokera mogu se koristiti klijentski alati koji su instalirani sa Mosquitto MQTT brokerom. S obzirom da u MQTT protokolu može biti istovremeno više spojenih klijenata (aktivnih veza), te svi mogu istovremeno objavljivat poruke ili biti pretplaćeni na neke teme. Testiranje se vrši tako da su otvorena dva ili više prozora Linux terminala. U prvom prozoru pretplaćuje se na temu „test“, a u drugom ili ostalima se objavljuje poruka „Linux testiranje MQTT klijenti“. Naredba za pretplatu na temu:

```
sudo mosquitto_sub –t “test”
```

Nakon uspješne pretplate potrebno je otvoriti drugi Linux terminal, te objaviti poruku sadržaja "Linux testiranje MQTT klijenti" na temu "test", upisom sljedeće naredbe u terminal:

```
sudo mosquitto_pub –t "test" –m "Linux testiranje MQTT klijenti"
```

Ako su sve naredbe bez greške upisane, u prozoru terminala pretplaćenom na temu „test" prikazuje se objavljena poruka.

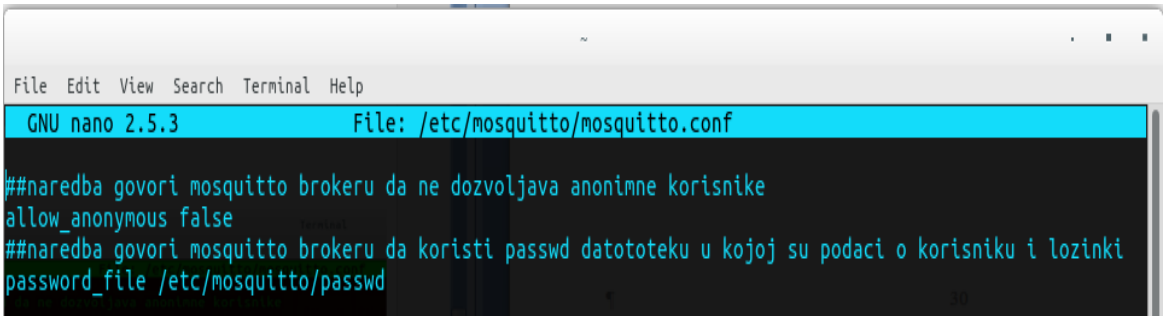


Slika 6.3 Objavljena poruka „Linux testiranje MQTT klijenti"

MQTT broker koristi konfiguracijsku datoteku (*mosquitto.conf*) smještenu u direktoriju */etc/mosquitto*. U tom direktoriju također se nalazi mapa (*mosquitto conf.d*) gdje korisnik može staviti svoju konfiguraciju, koju će Mosquitto koristiti kao zadanu (eng. *default*). Za korištenje autentifikacije potrebno je kreirati korisnika i lozinku. Također je potrebno generirati takozvanu „*passwd*" datoteku, odnosno datoteku koja sadržava korisničko ime i lozinku. Datoteku sa korisničkim imenom i lozinkom moguće je generirati upisom sljedeće naredbe u terminal:

```
sudo mosquitto_passwd –c /etc/mosquitto/passwd testuser
```

Ovom naredbom stvara se *passwd* datoteka u direktoriju */etc/mosquitto* koja sadrži korisnika „testuser" nakon što se potvrdi naredba sa tipkom „enter" korisnik će biti zamoljen da unese lozinku za korisnika (u ovom slučaju „testuser"). Da bi MQTT broker mogao koristiti autentifikaciju potrebno je izmijeniti *mosquitto.conf* datoteku. Izmjene uključuju postavljanje zabrane anonimnih korisnika te navođenje direktorija s datotekom korisničkog imena i lozinke (*passwd* datoteka), što je prikazano na sljedećoj slici.

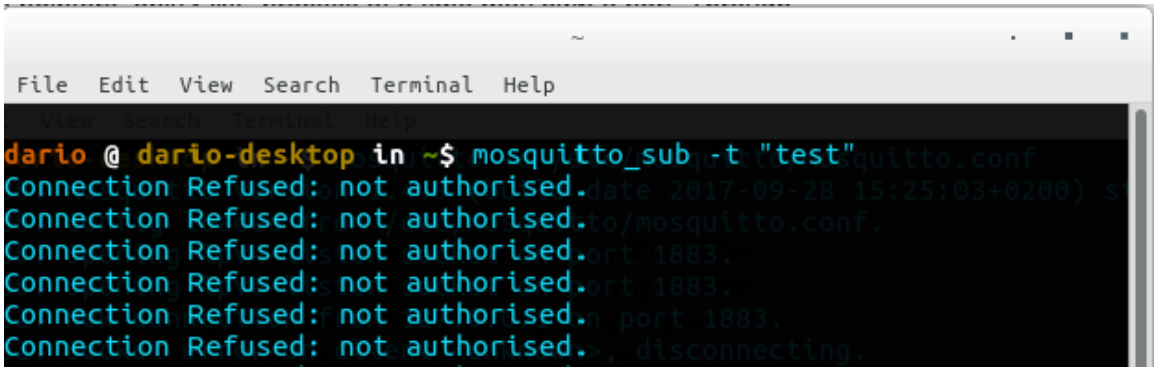


Slika 6.4 Izgled mosquitto.conf datoteke nakon potrebnih izmjena

Nakon spremanja izmjena potrebno je ponovno pokrenuti Mosquitto MQTT broker uz parametar konfiguracijske datoteke koristeći sljedeću naredbu:

```
sudo mosquitto -c /etc/mosquitto/mosquitto.conf
```

Ponovna pretplata na temu“test“ sada bi rezultirala sljedećim odazivom terminala.



Slika 6.5 Slika prikazuje neovlašteni pokušaj pristupa Mosquitto MQTT brokeru

Iz slike 6.5 vidi se da postavljena autentifikacija radi, odnosno sada nije moguća pretplata ili objava ako se ne koristi korisničko ime i lozinka. Da bi sa postavljenom autentifikacijom bilo moguće objaviti neku poruku, kao i pretplatiti se na neku temu, potrebno je u naredbi za objavu poruke, kao i u naredbi za pretplatu na temu koristiti parametre korisničkog imena i lozinke.

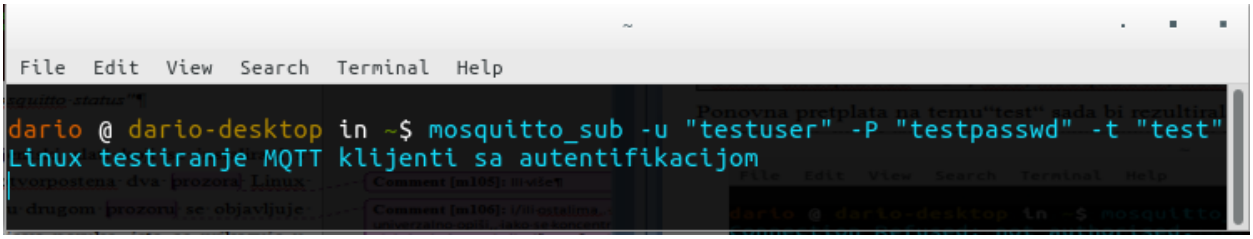
Naredba za pretplatu na temu „test“, uz korištenje autentifikacije sada izgleda ovako:

```
sudo mosquitto_sub –u “testuser” –P “testpasswd” –t “test”
```

Naredba za objavu poruke sadržaja „Linux testiranje MQTT klijenti sa autentifikacijom“ na temu „test“:

```
sudo mosquitto_pub –u “testuser” –P “testpasswd” –t “test” –m “Linux testiranje MQTT klijenti sa autentifikacijom”
```

Upis ovih naredbi ima sljedeći odaziv.



Slika 6.6 Prikaz objavljene poruke „Linux testiranje MQTT klijenti sa autentifikacijom“

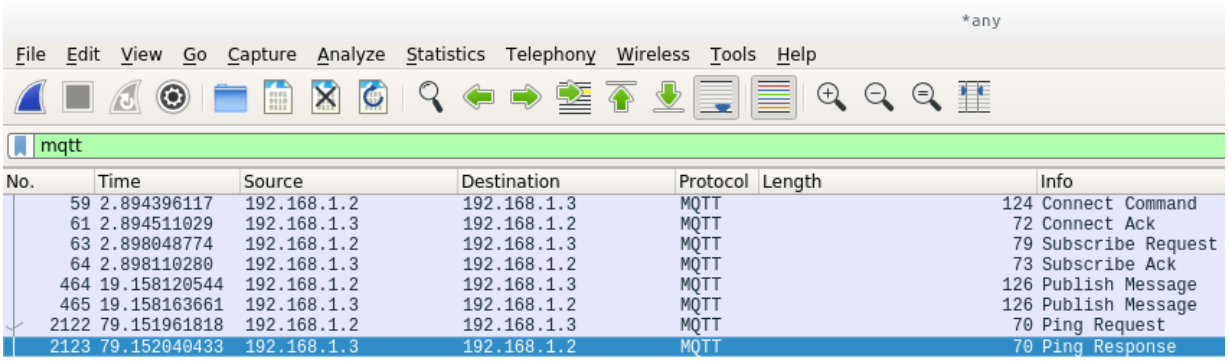
Ovakva konfiguraciju MQTT brokera može se koristiti na lokalnoj mreži (korisnici imaju pristup MQTT brokeru unutar lokalne mreže). Da bi se omogućio pristup MQTT brokeru izvan lokalne mreže trebalo bi koristiti servis za „sinkronizaciju“ dinamičkih DNS/IP adresa, te korištenje privatnih prema van. Jedan od takvih servisa je NoIP. Takvim pristupom bilo bi moguće postaviti MQTT broker na mikro računalu kao što je Raspberry Pi koji bi služio kao MQTT broker dostupan preko Interneta, s čime bi se korisniku znatno povećao pristup MQTT brokeru s obzirom na geografsku lokaciju.

6.2.1. Analiza MQTT paketa na lokalnoj mreži pomoću programa Wireshark

Da bi se izvršila analiza MQTT paketa potreban je programski alat za analizu mrežnog prometa. Jedan od takvih alata je Wireshark. Instalacija ovog programa na Linux operativnom sustavu vrši se tako da se u terminal upiše naredba:

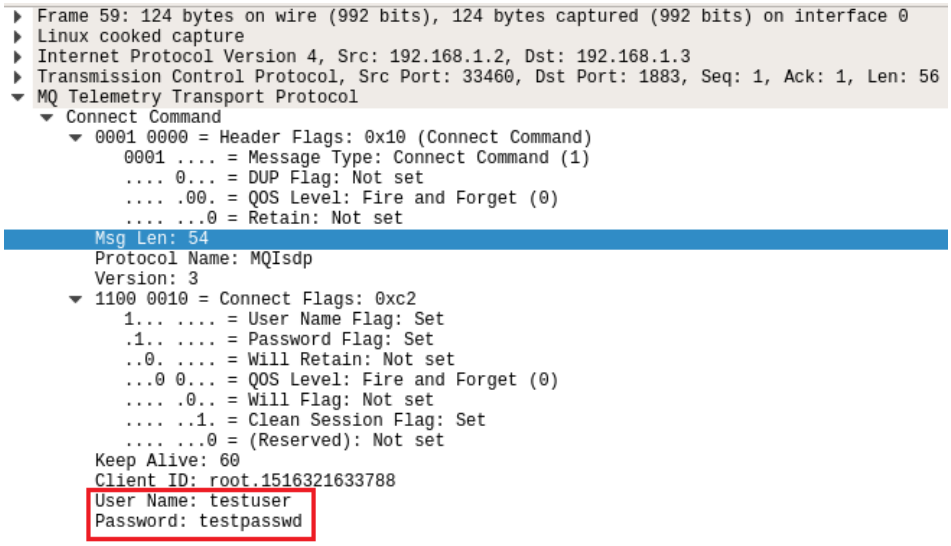
```
sudo apt-get install wireshark
```

Nakon instalacije potrebno je započeti analizu prometa na mrežnom sučelju računala te generirati MQTT promet. Budući da Wireshark analizira sav mrežni promet (a među mnoštvom protokola koje zna analizirati prepoznaje i MQTT), radi lakšeg snalaženja moguće je filtrirati MQTT protokol. Uhvaćeni odnosno snimljeni MQTT paketi koristeći Wireshark prikazani su na slici 6.7



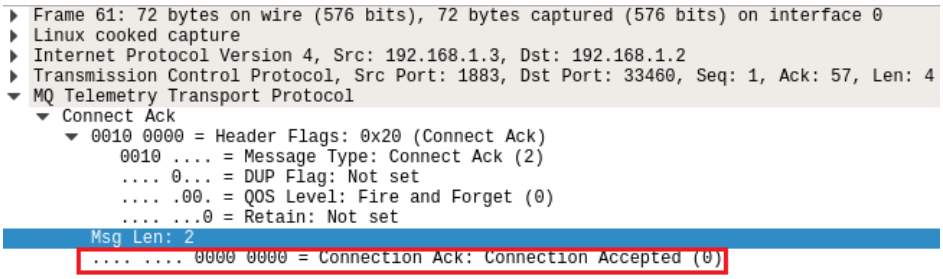
Slika 6.7 Prikaz snimljenih MQTT paketa koristeći program Wireshark

Slika 6.7 prikazuje cijeli tijek komunikacije između MQTT brokera i klijenta. MQTT broker instaliran je na računalu s IP adresom 192.168.1.3. Druga IP adresa 192.168.1.2 predstavlja klijenta. Prvi paket šalje klijent prema MQTT brokeru, odnosno šalje mu zahtjev za uspostavu veze (**`CONNECT`** paket). Analizom samog **`CONNECT`** paketa vide se postavljene zastavice za korisničko ime i lozinku u varijabilnom zaglavlju, ali interesantnije je korisničko ime i lozinka u teretu **`CONNECT`** paketa, što je prikazano kao „čisti“ tekst (eng. *Plain Text*). Ovo je posebno zanimljivo s obzirom na sigurnost jer svatko tko analizira promet može doći do korisničkog imena i lozinke. Prikaz zaglavlja **`CONNECT`** paketa vidljiv je na slici 6.8



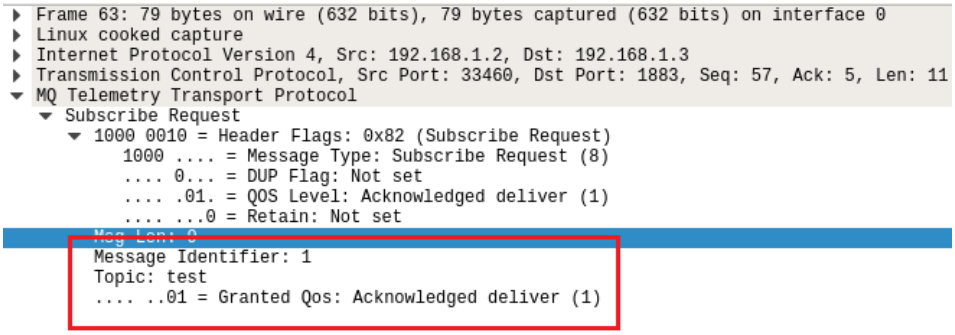
Slika 6.8 Prikaz fiksnog i varijabilnog zaglavlja MQTT paketa, iz čijega se tereta može pročitati korisničko ime i lozinka

Sljedeći paket prema slici 6.9 je potvrda zahtjeva za vezu (**CONNACK**). Ovaj paket u svojem varijabilnom zaglavlju sadrži takozvani povratni kod, prema čijoj je vrijednosti veza prihvaćena, odbijena itd... Vrijednost povratnog koda u ovom slučaju je nula, što znači da je veza prihvaćena.



Slika 6.9 CONNACK paket, vrijednost povratnog koda je 0 što znači da je zahtjev za uspostavu veze prihvaćen

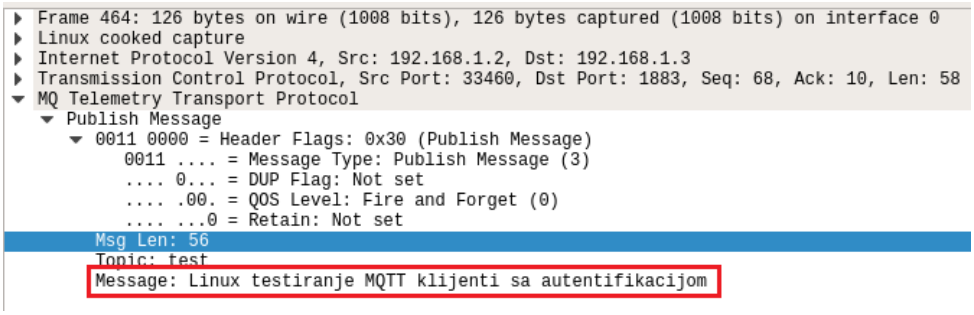
Slijedi zahtjev za pretplatu takozvani **SUBSCRIBE** paket kojeg šalje klijent MQTT brokeru.



Slika 6.10 Prikaz SUBSCRIBE kontrolnog MQTT paketa

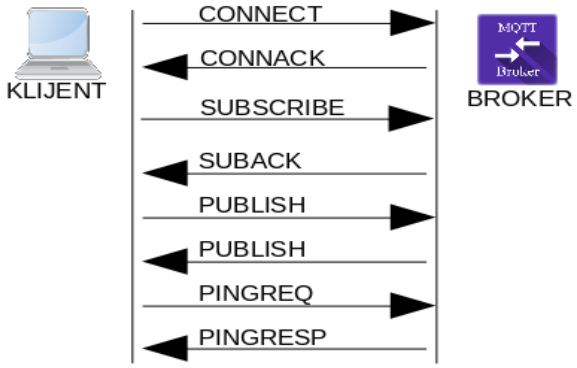
Iz slike 6.10 vidi se da teret ovog paketa sadrži popis teme (u ovom slučaju je samo jedna, no može ih biti i više). Paket u svom teretu isto tako sadrži garantiranu kvalitetu usluge (QoS). U nizu paketa nalaze se potvrda zahtjeva za pretplatom **SUBACK** kojeg šalje MQTT broker klijentu, kao potvrdu da je primio popis tema na koje se klijent želi pretplatiti. Pri samom kraju nalazi se

PUBLISH paket u kojem je objavljena poruka „Linux testiranje MQTT klijenti sa autentifikacijom“. Poruku šalje klijent prema MQTT brokeru te ukoliko se provede analiza ovog paketa vidi se objavljena poruka, što je prikazano na slici 6.11.



Slika 6.11 Prikaz PUBLISH MQTT paketa

Dijagram na slici 6.12 prikazuje sve MQTT pakete snimljene putem Wireshark programa.



Slika 6.12 Prikaz paketa snimljenih pomoću programa Wireshark

Dijagram na slici 6.12 prikazuje razmjenu paketa između MQTT broker i klijenta. Veza se uspostavlja tako da klijent MQTT brokeru šalje **CONNECT** paket, na što broker odgovara sa **CONNACK**. S obzirom da se klijent želi pretplatiti na temu, brokeru šalje **SUBSCRIBE** paket (zahtjev za pretplatu). Broker potvrđuje pretplatu sa **SUBACK** paketom. Prilikom objave poruke na neku temu klijent šalje **PUBLISH** paket. Na slici 6.12 zanimljivo je da **PUBLISH** paket ide dva puta. Prvi put od klijenta prema brokeru, te drugi put od brokera prema klijentu. Ovdje se radi o tome da se klijent prije same objave poruke pretplatio na temu „test“, i zbog toga je broker proslijedio poruku natrag klijentu.

6.2.2. Korištenje sigurnosnog transportnog sloja SSL/TLS u lokalnoj mreži

Da se MQTT poruke ne bi prikazivale kao čitljivi tekst u MQTT protokol implementirana je mogućnost korištenja sigurnosnog transportnog sloja SSL/TLS. Da bi veza između klijenta i brokera bila kriptirana, između ostalih rješenja postoji mogućnost korištenja alata OpenSSL. OpenSSL je besplatan kriptografski alat koji implementira SSL/TLS mrežne protokole, kao i

ostale standarde i funkcije koje se odnose na sigurnosne protokole. Instalacija ovog alata na Linux operativnim sustavima moguća je korištenjem apt-get, što je jako jednostavno i brzo.

```
sudo apt-get install openssl
```

Nakon same instalacije alata potrebno je generirati vlastiti CA autoritet kojim je potpisan certifikat poslužitelja odnosno brokera. Ovaj CA autoritet bit će potrebno postaviti na MQTT brokeru, kao i na klijentima budući da certifikat nije javno dostupan. Na brokeru je još potrebno postaviti CA certificiran certifikat poslužitelja kao i privatni ključ za dekodiranje podataka. Potrebne certifikate moguće je generirati koristeći OpenSSL u par koraka:

- Generiranje privatnog ključa za CA autoritet korištenjem slijedeće naredbe u terminalu.

  ```
  sudo openssl genrsa -des3 -out ca.key 2048
  ```

- Generiranje CA certifikata, koji je potpisan korištenjem privatnog ključa iz prethodnog koraka.

  ```
  sudo openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
  ```

- Generiranje privatnog ključa za poslužitelj, odnosno MQTT broker.

  ```
  sudo openssl genrsa -out server.key 2048
  ```

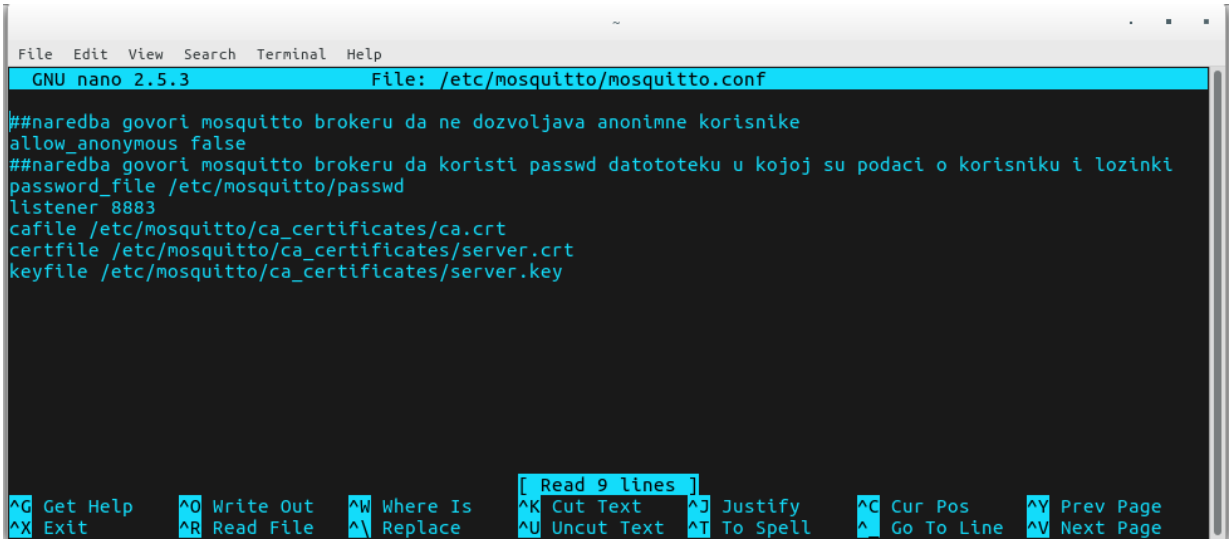
- Generiranje zahtjeva za potpis certifikata CSR. Prilikom popunjavanja zahtjeva, polje „*common name*“ mora sadržavati domenu MQTT brokera. Budući da se radi o lokalnoj mreži može se koristiti IP adresa uređaja na kojemu je postavljen MQTT broker. Ako se koriste zadane postavke DHCP poslužitelj će uređaju dodijeliti IP adresu koja neće uvijek biti ista. Zbog toga bi valjalo promijeniti postavke DHCP poslužitelja na način da uređaju na kojem je MQTT broker dodjeli uvijek istu IP adresu. Naredba za generiranje zahtjeva potpisa certifikata:

  ```
  sudo openssl req -new -out server.csr -key server.key
  ```

- Korištenje privatnog ključa iz prvog koraka (ca.key) s kojim se potpisuje certifikat poslužitelja korištenjem slijedeće naredbe.

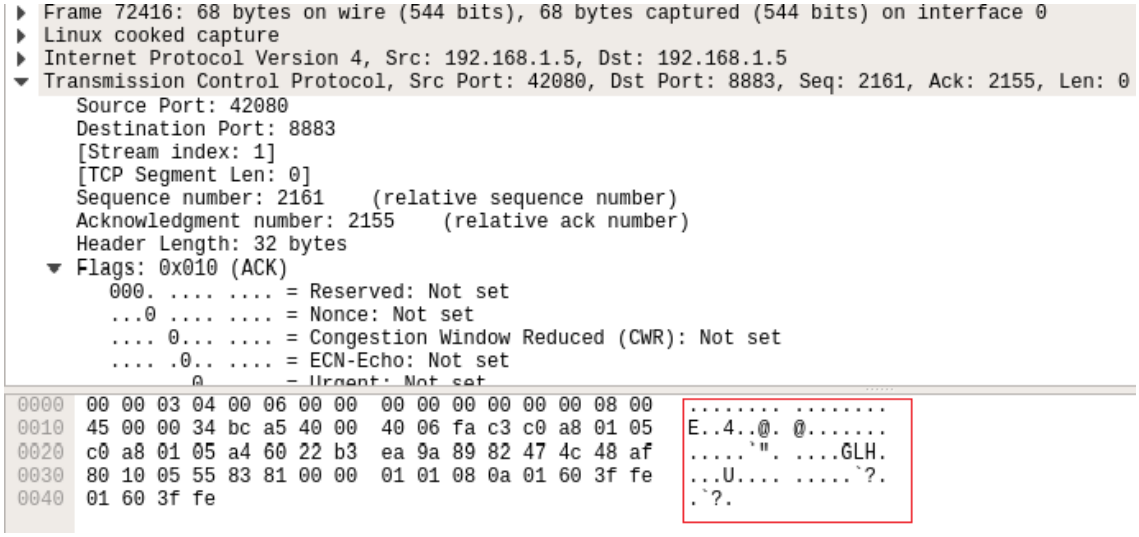
  ```
  sudo openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key
  -CAcreateserial -out server.crt -days 360
  ```

Sa ovim naredbama stvorene su slijedeće datoteke: *ca.crt*, *ca.key*, *ca.srl*, *server.crt*, *server.csr*, *server.key*. Da bi Mosquitto broker koristio TLS/SSL u njegovu konfiguracijsku datoteku potrebno je uključiti slijedeće datoteke: *ca.crt*, *server.crt*, *server.key*. Ispravno postavljena konfiguracijska datoteka Mosquitto MQTT brokera prikazana je na slijedećoj slici.



Slika 6.13 Izgled Mosquitto konfiguracijske datoteke uz korištenje TLS/SSL protokola

Ako se klijent poveže na MQTT broker korištenjem TLS/SSL protokola, u programu Wireshark vidi se da je veza kriptirana.



Slika 6.14 Prikaz zarobljenih paketa MQTT protokola uz korištenje TLS/SSL protokola

Ovim primjerom prikazana je implementacija TLS/SSL sigurnosnog protokola na Mosquitto MQTT brokeru, koji je postavljen na lokalnoj mreži. Postoji mogućnost korištenja dinamičkih DNS servisa i tako pristupiti MQTT brokeru kada je klijent izvan lokalne mreže. U tom slučaju korisnik može konfigurirati TLS/SSL pomoću servisa kao što je *Let's Encrypt*. U tom slučaju klijentima ne bi bilo potrebno ručno unositi CA certifikate jer bi oni bili javno dostupni.

6.3. Instalacija MQTT brokera na udaljeni poslužitelj

Korištenje MQTT protokola u IoT projektima, od MQTT brokera iziskuje dostupnost putem Interneta. U tome može pomoći virtualni privatni poslužitelj (VPS), ili iznajmljeni poslužitelj (eng. *dedicated server*). Naravno gotova rješenja u oblaku (eng. *cloud*) se isto mogu iskoristiti.

Kada se MQTT brokeru pristupi putem Interneta, od posebne je važnosti sigurnost podataka. MQTT protokol ima mogućnost korištenja sigurnosnog transportnog sloja (TLS/SSL) kojeg valja koristiti. Kada je MQTT broker dostupan putem Internet mreže povećava se njegova dostupnost s obzirom na geografsku lokaciju, odnosno ako klijent ima mogućnost korištenja Interneta može komunicirati s drugim klijentima koji isto tako komuniciraju sa MQTT brokerom koristeći Internet. Virtualni privatni poslužitelj (eng. *Virtual Private Server*, *VPS*) je način dijeljenja fizičkog poslužitelja na više virtualnih poslužitelja tako da se svaki virtualni poslužitelj može koristiti kao vlastito namjensko računalo. Svako od tih računala može pokretati svoj operativni sustav, koji ima pristup Internet mreži.

6.3.1. Instalacija Mosquitto MQTT brokera na udaljeni poslužitelj

U ovom poglavlju opisan je postupak instalacije Mosquitto MQTT brokera na virtualnom privatnom poslužitelju koji koristi Ubuntu server operativni sustav sa „non-root“ korisnikom naziva „mqtt“ uključenim u „sudo grupu“ te može izvršavati sudo naredbe. U ovom primjeru korištena je domena „mqtt-projekt.tk“ koja upućuje na IP adresu virtualnog privatnog poslužitelja. Da bi se pristupilo VPS-u korišten je SSH protokol, pomoću kojeg se otvara terminal na udaljenom računalu (VPS-u). SSH (eng. *secure shell*) je mrežni protokol koji korisnicima omogućuje uspostavu sigurnog komunikacijskog kanala između dva računala. Upisom sljedeće naredbe u terminal stvara se SSH sesija sa VPS-om na kojeg upućuje domena „mqtt-projekt.tk“.

```
sudo ssh mqtt@mqtt-projekt.tk
```

Nakon uspješne prijave u operativni sustav slijedi instalacija Mosquitto MQTT brokera koja je obavljena na isti način kao i u poglavlju 6.2. Nadalje, nakon uspješne instalacije Mosquitto MQTT brokera potrebno je obratiti pozornost na sigurnost komunikacije između klijenata i MQTT brokera. Zbog zaštite od potencijalnih napadača na mreži važno je šifrirati kanal veze koristeći TLS/SSL enkripcijski protokol. Za rad s TLS/SSL protokolom potrebno je generirati certifikate, za to je korišten besplatni servis *Let's Encrypt*. Korištenje ovog servisa zahtjeva instalaciju Certbot klijenta, za čiju je instalaciju u otvorenu SSH sesiju potrebno upisati slijedeće naredbe:

```
sudo add-apt-repository ppa:certbot/certbot
```

```
sudo apt-get update
```

```
sudo apt-get install certbot
```

Prije samog pokretanja, Certbot klijent mora odgovoriti na niz kriptografskih izazova koje je generirao *Let's Encrypt* automatski API. Za to se koristi HTTP port 80 ili HTTPS port 443.

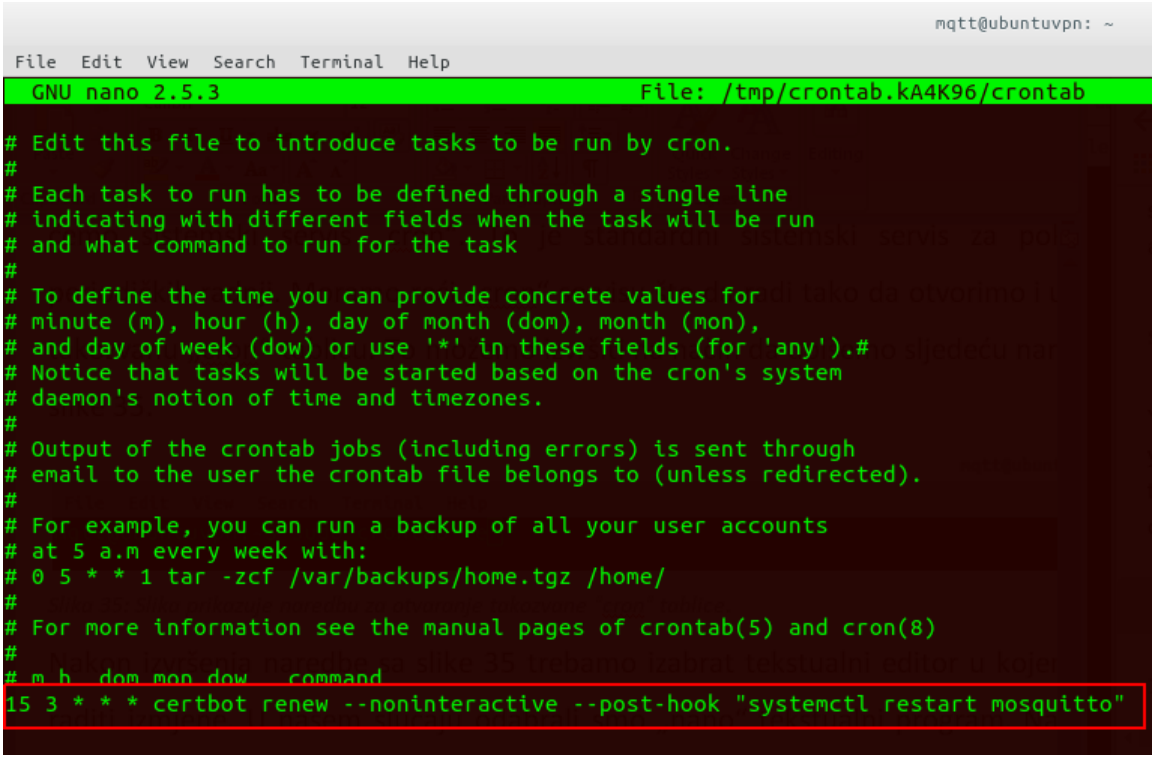
Važno je promijeniti postavke vatrozida (eng. *Fire Wall*) kako bi se omogućilo korištenje ovih portova. Naredba za dodavanje pravila u sustav vatrozida zbog korištenja HTTP porta 80:

`sudo ufw allow 80`

Pokretanjem Certbot klijenta uz parametre: `-standalone` (Certboot automatski svladava HTTP izazov), `standalone-supported-chellenges http-01` (ograničavanje komunikacije na port 80), `-d` (navođenje domene koju želimo certificirati), `certonly` (Certbot klijent će povući certifikat bez dodatnih konfiguracijskih koraka). Naredba sa ovakvim parametrima:

`sudo certbot certonly --standalone --standalone-supported-challenges http-01 –d mqtt-projekt.tk`

Budući da je *Let's Encrypt* certifikat valjan samo 90 dana, potrebno je regularno izvršavati naredbu za provjeru roka certifikata kao i njegovu obnovu. Da bi se izbjegla ručna provjera certifikata moguće je „automatizirati“ proces periodičke provjere koristeći sistemski servis `Cron`. `Cron` je standardni sistemski servis za pokretanje periodičkih radnji na Linux operativnom sustavu. Da bi se proces obnove certifikata „automatizirao“ potrebno je urediti takozvane `Cron` tablice.

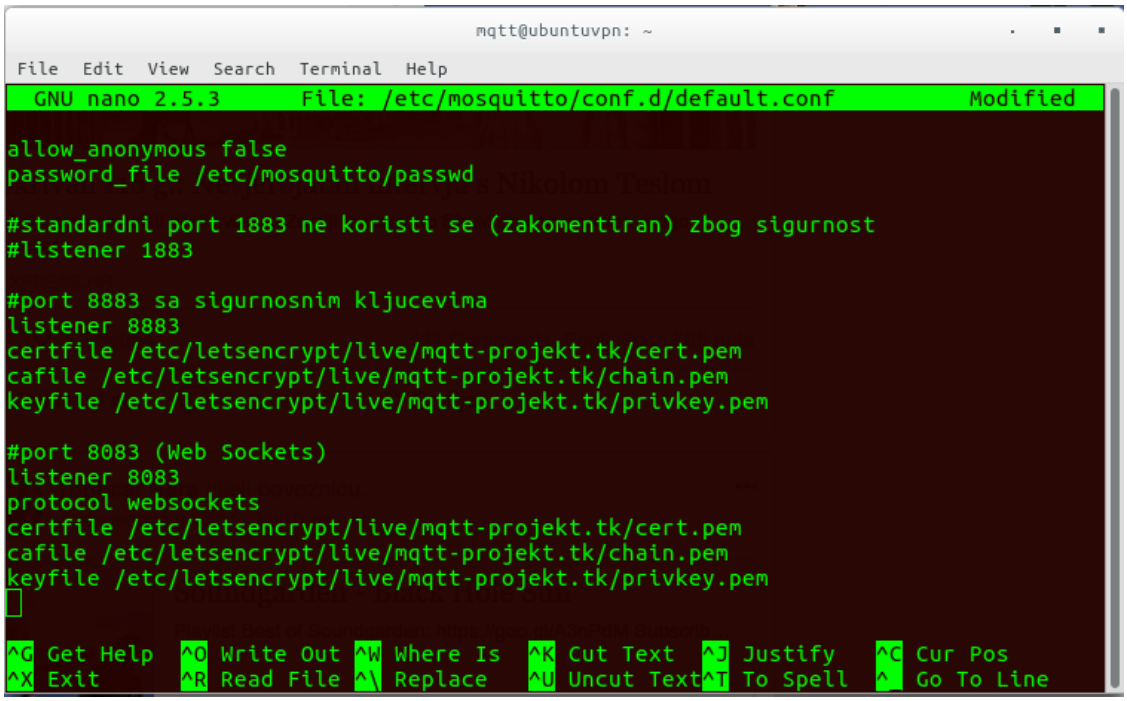


Slika 6.15 Prikaz naredbe za periodičku provjeru i obnovu certifikata u Cron tablici

Naredba sa slike 6.15 će automatski obnoviti certifikat i ponovno pokrenuti Mosquitto servis.

Da bi Mosquitto MQTT broker mogao koristiti *Let's Encrypt* certifikate, potrebno je učiniti promjene u njegovoj konfiguracijskoj datoteci, odnosno da bi Mosquitto svaki put nakon ponovnog pokretanja očitao novu konfiguraciju bitno je promijeniti njegovu zadanu

konfiguraciju. Zadana konfiguracija smještena je u */etc/mosquitto/conf.d* direktoriju. U konfiguraciji je bitno zabraniti pristup anonimnim korisnicima, navesti direktorij u kojem je smještena datoteka sa korisnicima i lozinkama (*pwd* datoteka), postaviti port 8883 te navesti direktorije s potrebnim certifikatima i ključevima zbog korištenja TLS/SSL protokola. Opcionalno isto je potrebno učiniti i za port 8083 koji ima takozvanu web-socket podršku. Ovako postavljena konfiguracije prikazana je na slici 6.16.

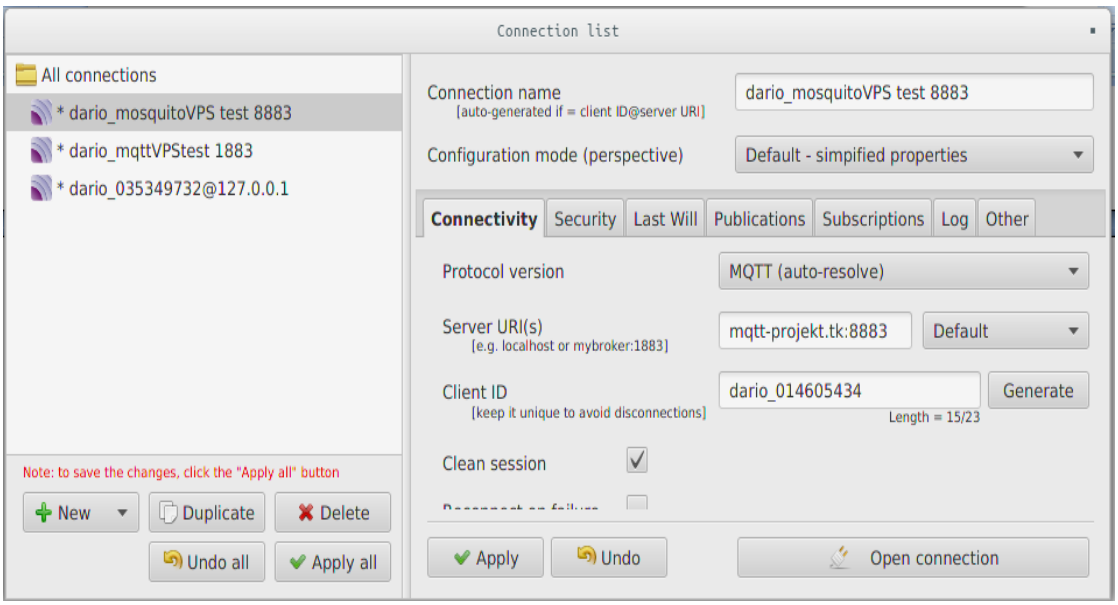


Slika 6.16 Zadana konfiguracije Mosquitto brokera

Za korištenje Mosquitto MQTT brokera potrebno je konfigurirati vatrozid na način da zabranimo komunikaciju na portu 1883, a dozvolimo na portu 8883 te 8083.

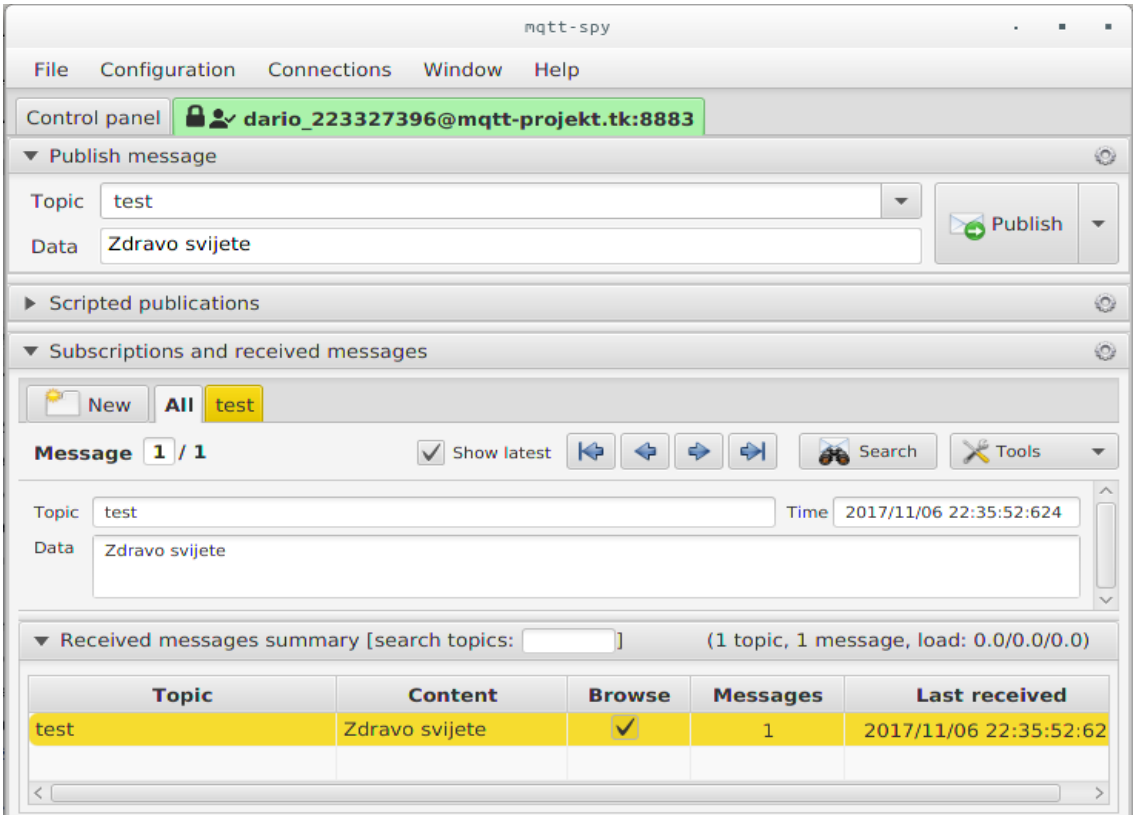
6.3.2. Testiranje povezivanja na Mosquitto MQTT broker instaliran na udaljenom poslužitelju

Program korišten za testiranje veze naziva se Mqtt-spy. Nakon pokretanja programa potrebno je kreirati novu vezu. Zatim je potrebno upisati podatke o povezivanju: IP adresu ili domenu MQTT brokera, port, korisničko ime i lozinku. Također je potrebno postaviti potrebne opcije za korištenje TLS/SSL protokola.



Slika 6.17 Izgled sučelja programa Mqtt-spy (tabela sa opcijama za uspostavu veze)

Nakon uspješnog povezivanja s MQTT brokerom otvara se sučelje s mogućnostima pretplate na temu te objave poruka. Izgled sučelja prikazan je na slici 6.18.



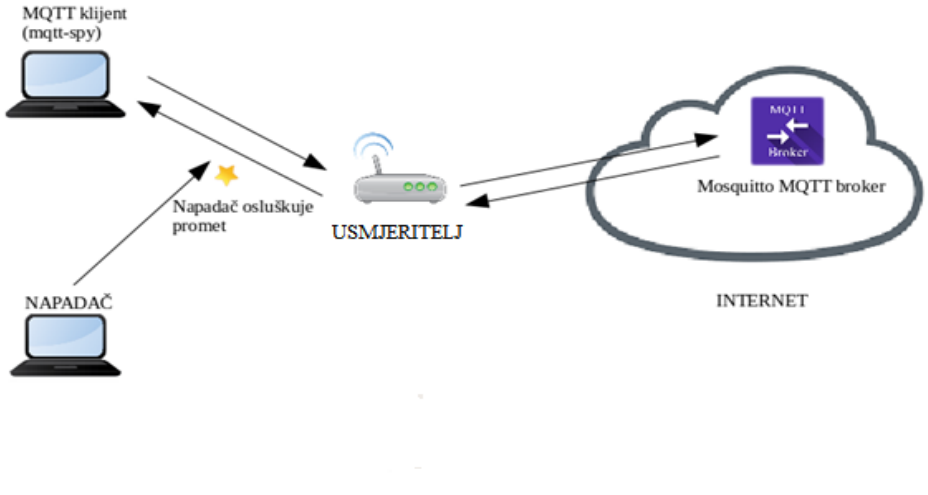
Slika 6.18 Izgled sučelja programa Mqtt-spy (prikaz objavljene poruke „Zdravo svijete“ na temu „test“)

6.3.3. Funkcioniranje TLS/SSL protokola za uspostavu sigurnosne veze

TLS (eng. *Transport Layer Security*) kao i njegov prethodnik SSL (eng *Secure Socket Layer*) su kriptografski protokoli koji omogućavaju sigurnu komunikaciju putem Interneta. Postoje mnoge razlike između ova dva protokola, ali u osnovi su isti. Klijent i poslužitelj dogovaraju vezu korištenjem procedure rukovanja (eng. *Handshaking procedure*). Tijekom te procedure oni dogovaraju parametre koji se koriste za uspostavljanje sigurnosne veze. Rukovanje počinje prilikom spajanja klijenta s poslužiteljem, a klijent zatim zahtijeva sigurnosnu vezu te poslužitelju predstavlja popis podržanih šifri odnosno tzv. hash funkcija. Poslužitelj će odabrati najjaču šifru s popisa te o tome obavijestiti klijenta te nakon toga poslužitelj šalje informacije o svojem identitetu u obliku digitalnog potpisa, odnosno šalje mu certifikat koji sadrži ime poslužitelja, povjerljivi CA (eng. *Certificate authority*) entitet te poslužiteljev javni ključ kriptiranja. Prije same obrade klijent može kontaktirati poslužitelja koji izdaje certifikat te provjeriti da li je taj certifikat autentičan. Kako bi se generirao ključ sjednice sigurnosne veze, klijent kriptira proizvoljno odabran broj pri čemu koristi poslužiteljev javni ključ, a rezultat šalje poslužitelju. Samo poslužitelj može dekriptirati poruku sa svojim privatnim ključem koji je poznat samo poslužitelju (klijent zna samo javni ključ). Iz proizvoljno odabranog broja klijent kao i poslužitelj generiraju podatke koji će se koristiti za kriptiranje i dekriptiranje. Time završava faza rukovanja i inicira se sigurnosna veza između klijenta i poslužitelja koja je kriptirana do njenog prekida.

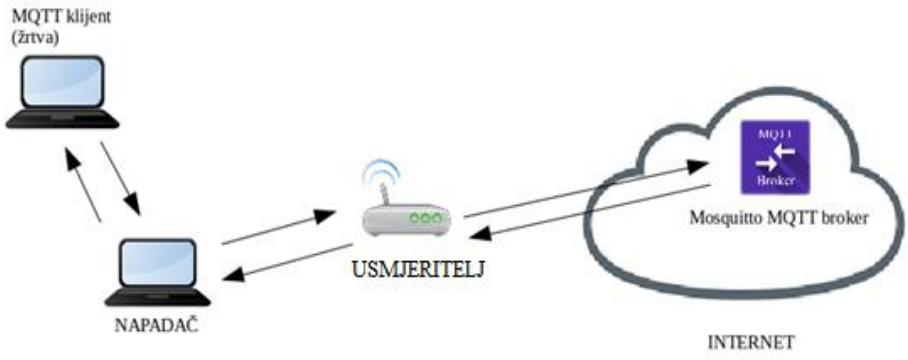
6.3.4. Demonstracija napada na Mosquito MQTT broker

Da bi se istaknula važnost korištenja sigurnosnog TLS/SSL protokola kod komunikacije između klijenta i MQTT brokera, u ovom primjeru je demonstriran mrežni napad na MQTT broker instaliran na virtualnom privatnom poslužitelju. MQTT protokol kao zadani TCP port za komunikaciju koristi port 1883. Korištenjem ovog TCP porta veza između klijenta i MQTT brokera nije kriptirana. S obzirom na to da veza nije kriptirana svi podaci koje klijent razmjenjuje s MQTT brokerom preko mreže prenose se kao čitljiv (eng. *Plain*) tekst, uključujući korisničko ime i lozinku. Ako na mreži postoji treća strana koja osluškuje promet između klijenta i MQTT brokera („napadač“), taj napadač ima uvid u korisničko ime/lozinku klijenta, kao i u sadržaj poruka koje se razmjenjuju na mreži.



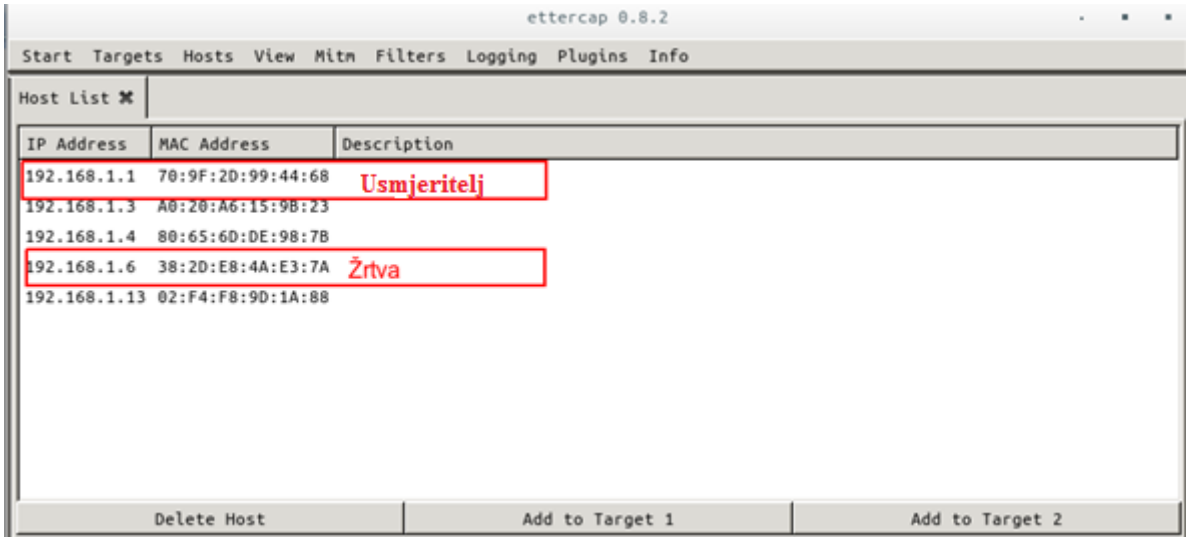
Slika 6.19 Mrežni napad na komunikaciju između klijenta i MQTT brokera

Mrežni napad prikazan slikom 6.19 poznat je još pod nazivom „čovjek u sredini“ (eng. *Man in the middle,* MiM). Da bi se napadač mogao ubaciti u komunikaciju između klijenta i MQTT brokera može se „predstaviti“ kao usmjeritelj preko kojeg klijent komunicira s MQTT brokerom i na taj način preko svog mrežnog sučelja skrenuti promet između klijenta i MQTT brokera. Ovakvo skretanje mrežnog prometa prikazano je na slici 6.20.



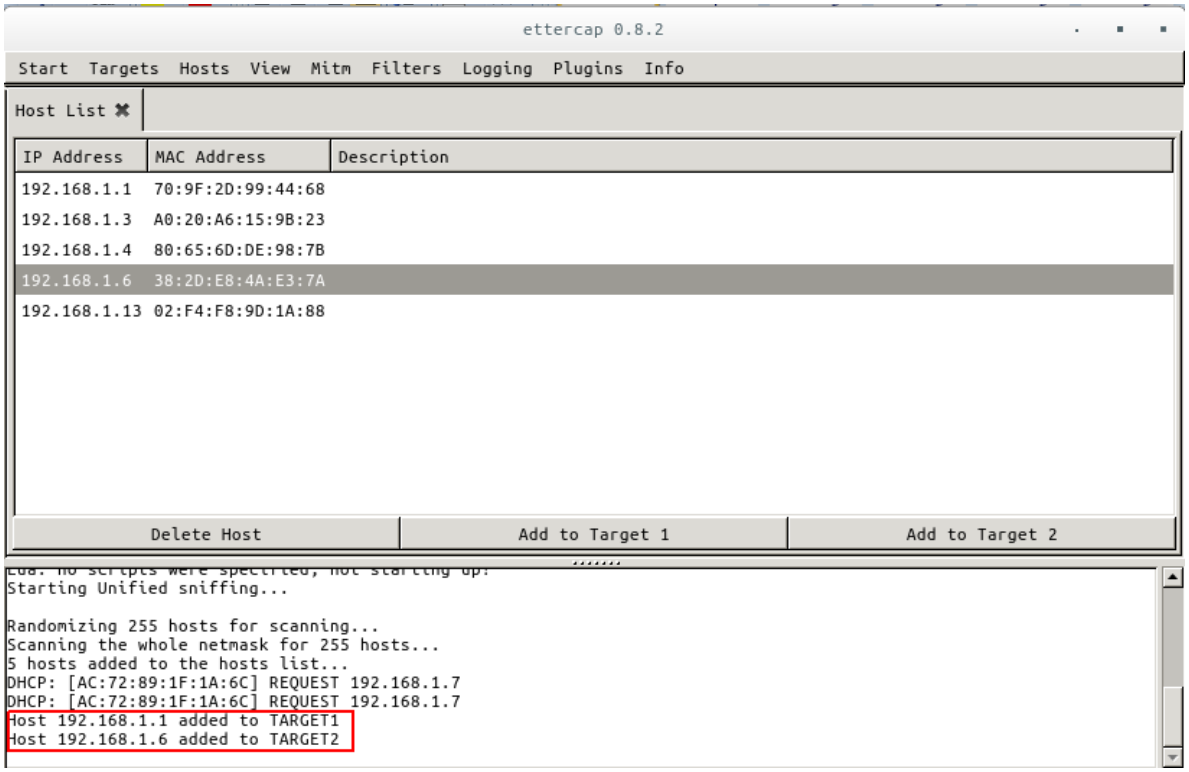
Slika 6.20 Skretanje prometa kroz napadačevo mrežno sučelje

Za ovakvu vrstu mrežnog napada „napadač“ treba saznati IP adrese klijenta i usmjeritelja. Nadalje „napadač“ će se „predstaviti“ klijentu kao usmjeritelj koristeći njegovu MAC adresu. IP adresa se koristi prilikom kontaktiranja domaćina (eng. *Host*) na mreži, a prilikom same dostave mrežnih paketa koristi se MAC adresa. S obzirom na to da MAC adresa nije verificirani protokol, napadač se može „predstaviti“ s MAC adresom klijenta. Kada MQTT broker šalje nešto klijentu, usmjeritelj će to proslijediti napadaču. Isto tako, napadač se može koristiti MAC adresom usmjeritelja, što znači da takvim manipulacijama napadač može osluškivati mrežni promet u oba pravca. Programski alat korišten za ovakvu vrstu napada u ovom primjeru je Ettercap. Skeniranjem mreže koristeći Ettercap otkrivaju se potrebne IP i MAC adrese.



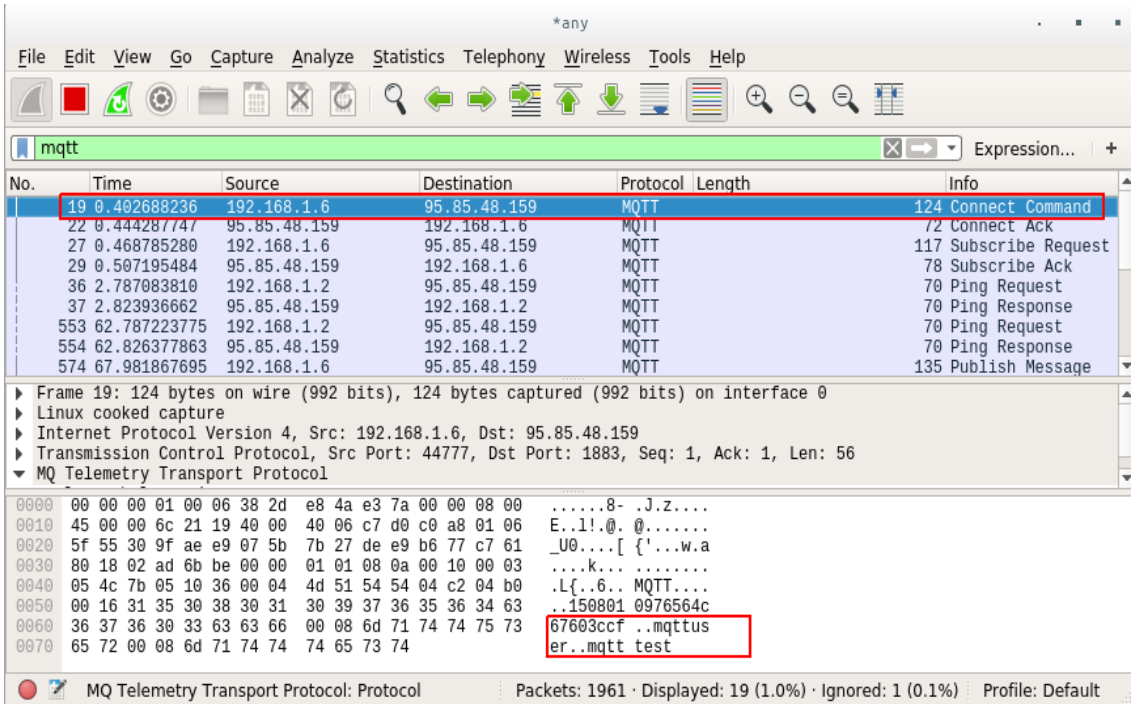
Slika 6.21 Prikaz IP i MAC adresa žrtve (MQTT klijenta), kao i usmjeritelja

Nakon otkrivanja IP adresa potrebno je dodati usmjeritelj kao „cilj 1“ (eng. *Target 1*) te žrtvu odnosno klijenta kao „cilj 2“(eng. *Target 2*), prikazano na slici 6.22



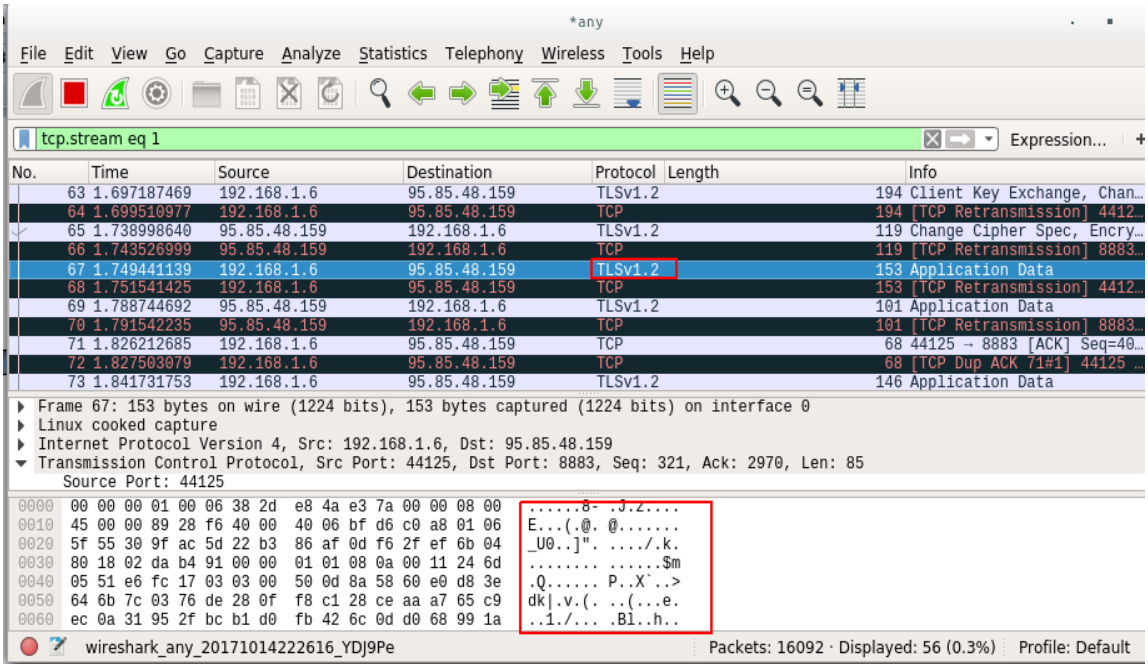
Slika 6.22 Slika prikazuje usmjeritelj kao „ cilj 1“ i žrtvu kao „ cilj 2“

Odabirom opcije „Mitm“ započinje takozvano ARP trovanje (eng. ARP *poisoning*) i napadač je uspješno presreo komunikaciju između klijenta i MQTT brokera. Korištenjem programskog alata za analizu prometa kao što je Wireshark napadač započinje osluškivati mrežni promet između klijenta i MQTT brokera, prikazano na slici 6.23.

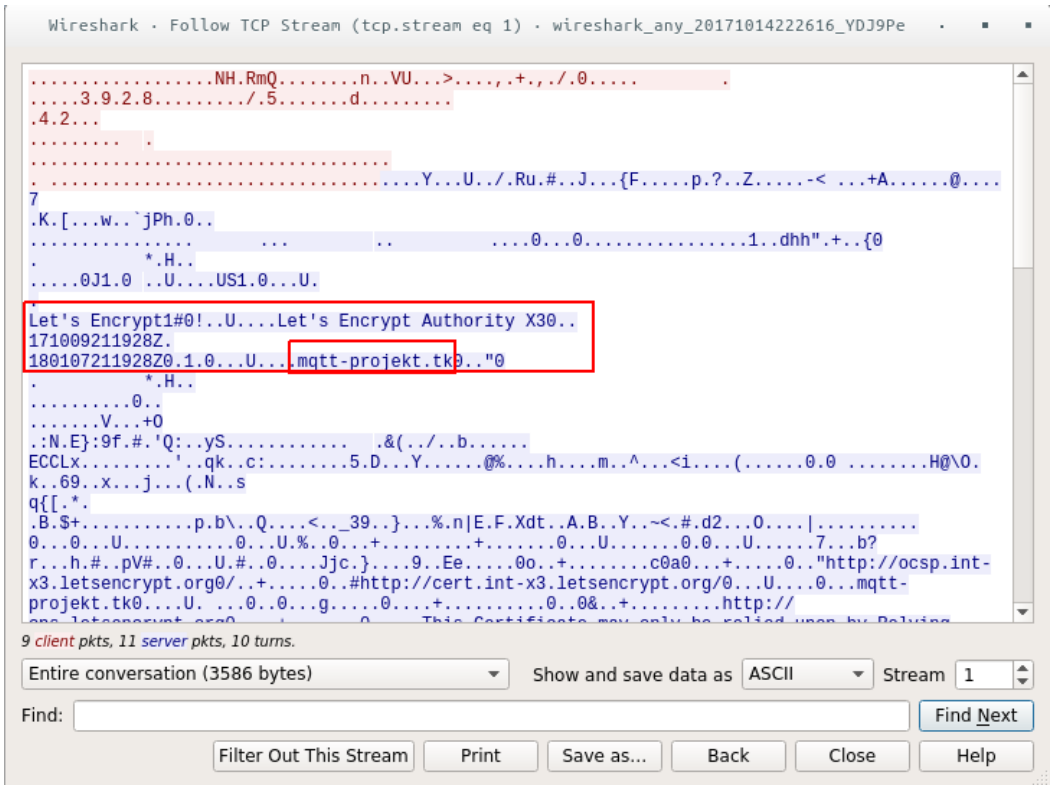


Slika 6.23 Prikaz sadržaja zarobljenog CONNECT MQTT paketa is kojega se može pročitati korisničko ime i lozinka

Ako se ne koristi kriptirana veza između klijenta i MQTT brokera, treća strana koja osluškuje mrežni promet (napadač) lako dolazi do korisničkog imena i lozinke. Zbog toga je važno umjesto TCP porta 1883 koristiti TCP port 8883. Razlika između ova dva TCP porta je u tome što se kod TCP porta 8883 koristi sigurnosni TLS/SSL protokol. Ako je veza kriptirana, prilikom korištenja TLS/SSL protokola, napadač nije u mogućnost čitati sadržaj mrežnih paketa što je prikazano na slikama 6.24 i 6.25.



Slika 6.24 *Prikaz paketa koje razmjenjuju klijent i MQTT broker korištenjem TLS/SSL sigurnosnog protokola*



Slika 6.25 Prikaz paketa koje razmjenjuju klijent i MQTT broker koristeći TLS/SSL protokol

7. ESP 8266-12F mikroupravljač kao MQTT klijent

Zbog demonstracije rada MQTT protokola na resursno ograničenim uređajima korišten je ESP 8266-12F mikroupravljač. Funkcija mikroupravljača je uspostavljanje sigurnosne (TLS/SSL) veze s MQTT brokerom te korištenje funkcija programske MQTT knjižnice, s kojima mikroupravljač ima funkcionalnosti MQTT klijenta. Pomoću funkcija MQTT klijenta, mikroupravljač može preko MQTT brokera komunicirati s Android mobilnom aplikacijom. Program mikroupravljača pretplaćen je na temu „LED“ preko koje dobiva poruke od strane korisnika mobilne aplikacije. Ovisno o sadržaju tih poruka mikroupravljač uključuje/isključuje svoje izlazne pinove na kojima su spojene LED diode, što je prikazano u tabeli 7.1.

Sadržaj MQTT poruke	Funkcija
10	Isključi crvenu LED diodu
11	Uključi crvenu LED diodu
20	Isključi zelenu LED diodu
21	Uključi zelenu LED diodu
30	Isključi plavu LED diodu
31	Uključi plavu LED diodu

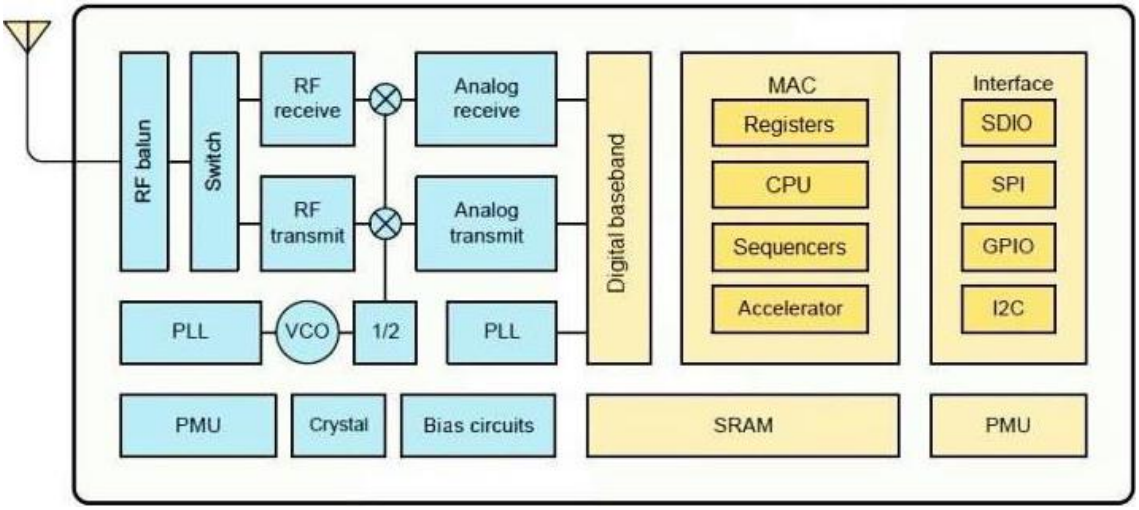
Tabela 7.1 Prikaz sadržaja MQTT poruka pomoću kojih se vrši kontrola mikroupravljača

Program mikroupravljača također očitava vrijednosti iz DHT11 senzora temperature i vlage, a zatim te vrijednosti svake 3 sekunde objavljuje na teme „Temp“ i „Vlaga“.

7.1. ESP 8266-12F mikroupravljač

Ovaj mikroupravljač kao jezgru sustava koristi ESP8266EX integrirani sklop, razvijen od kineske kompanije Espressif Systems. ESP8266EX integrirani sklop utjelovljuje Tensilica L106 32 bitnu mikroprocesorsku jedinicu s radnim taktom do 160 MHz. Navedeni integrirani sklop također ima ugrađenu radnu memoriju, kao i EEPROM s punom podrškom korištenja bežične komunikacije prema IEEE 802.11b/g/n standardu te podršku TCP/IP protokola. Postoje različite mogućnosti programiranja mikroupravljača. Prilikom pojave na tržištu ovaj mikroupravljač programirao se koristeći LUA programski jezik. Najpoznatija implementacija LUA programskog jezika za ESP8266 je program „NodeMCU Lua firmware“. Zahvaljujući velikoj zajednici korisnika kasnije je razvijena i Arduino IDE jezgra s kojom se ESP 8266 implementira u Arduino razvojno okruženje, poznato kao Arduino IDE. Kompanija Espressif Systems također je izdala SDK kojeg je lako implementirati u razvojno okruženje poput Eclipse-a te koristiti RTOS (eng. *Real-time operating system*).

Slika 7.1 ESP 8266-12F mikroupravljač [10]



Slika 7.2 Prikaz blok dijagrama ESP8266EX integriranog sklopa [11]

Popis pinova ESP 8266-12F mikroupravljača s generalnom ulaz/izlaz namjenom kao i specijalnom namjenom prikazan je u tabeli 7.2.

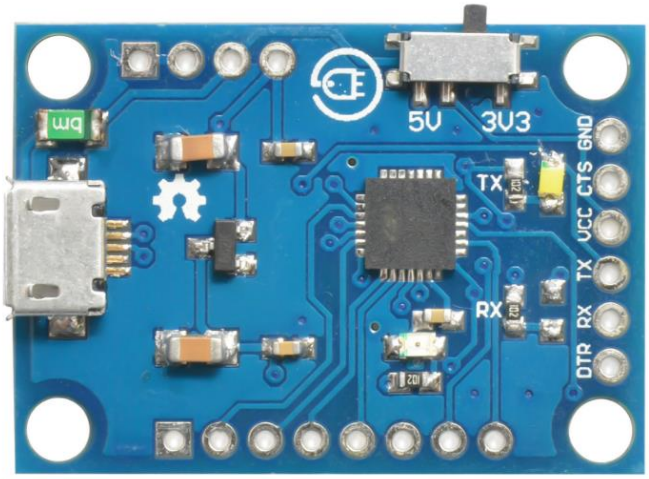
Broj pina	Naziv pina	Funkcija
1	RST	Resetiranje modula
2	ADC	Analogni ulaz A/D pretvornika. Ulazni napon u rasponu od 0 V do 1 V. Rezultat A/D pretvorbe je vrijednosti u registru prikazana brojem u rasponu od 0 do 1024.
3	EN	Omogućava rad uređaja. Da bi uređaj radio treba biti spojen na 3.3V
4	IO16	GPIO 16 (pin generalne namjene) može biti korišten za buđenje modula iz „dubokog sna“
5	IO14	GPIO14;HSPI_CLK
6	IO12	GPIO12;HSP_MISO
7	IO13	GPIO13;HSPI_MOSI;UART0_CTS
8	VCC	Napajanje 3.3V
9	CS02	Chip selection
10	MISO	Slave output, Main input
11	IO9	GPIO9
12	IO10	GBIO10
13	MOSI	Main output slave input
14	SCLK	Clock

15	GND	GND
16	IO15	GPIO15;MTDO;HSPIC;UART0_RTS
17	IO2	GPIO2;UART1_TXD
18	IO0	GPIO0
19	IO4	GPIO4
20	IO5	GPIO5
21	RXD	UART0_RXD;GPIO3
22	TXD	UART0_TXD;GPIO1

Tabela 7.2 Prikaz pinova ESP 8266-12E mikroupravljača [11]

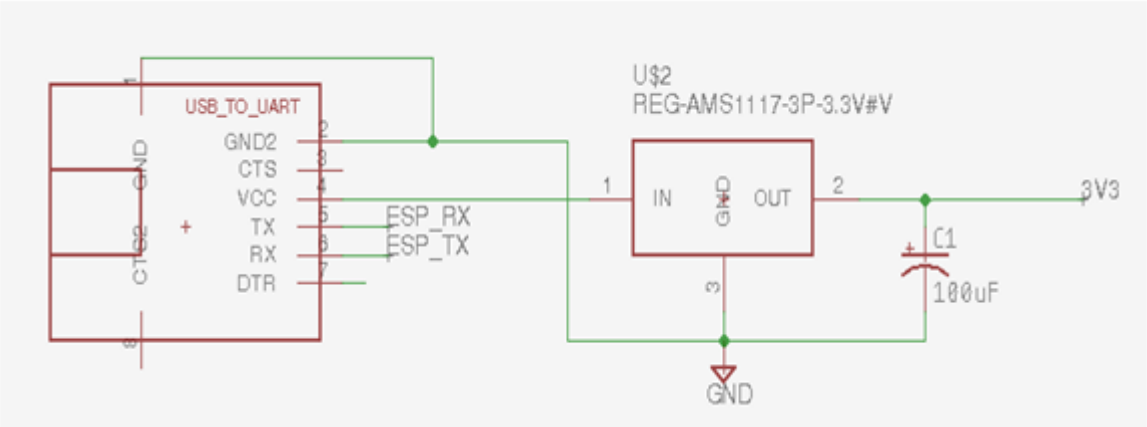
7.2. Elektronički sklop za rad s ESP 8266-12F mikroupravljačem

Za rad sa ESP 8266-12F mikroupravljačem korišten je elektronički sklop koji se sastoji od USB-UART adaptera baziranog na CP2102 integriranom sklopu.



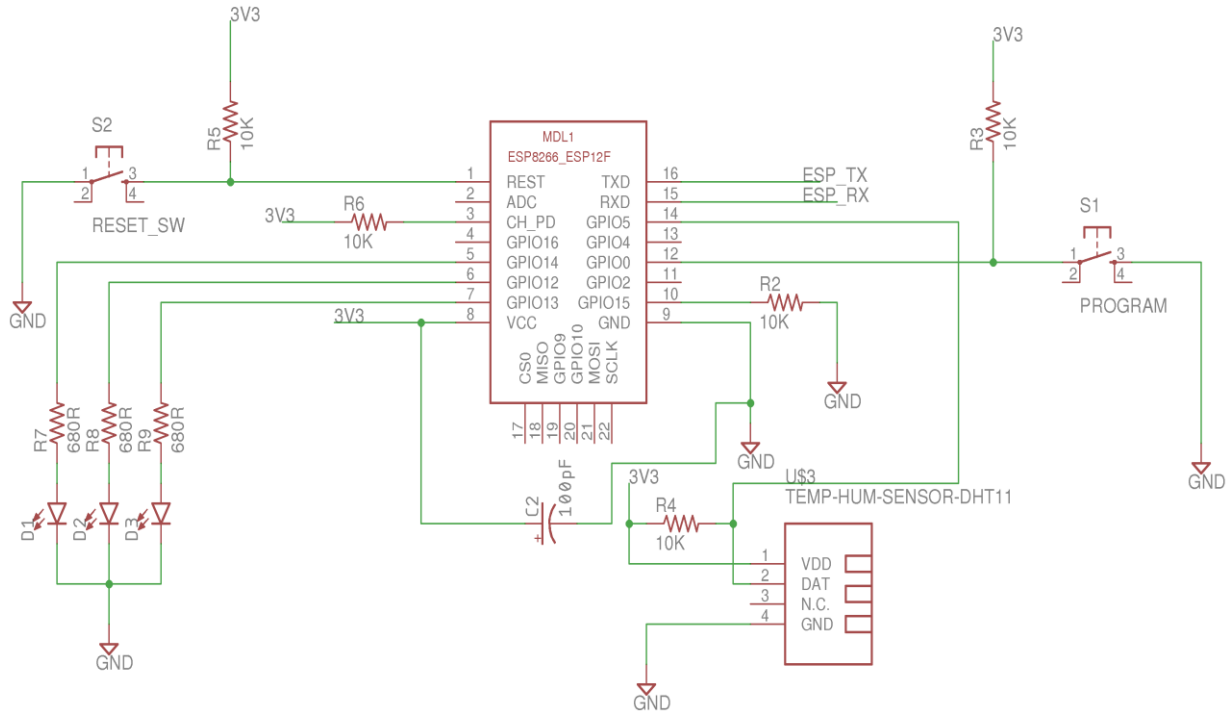
Slika 7.3 Slika prikazuje USB-UART adapter baziran na CP2102 integriranom sklopu.[12]

Korištenjem USB-UART adaptera sa mikroupravljačem moguće je komunicirati preko računala korištenjem RS232 protokola (UART komunikacija) i na taj način zapisivanje programskog koda u mikroupravljač. Osim same komunikacije sa računalom ovaj adapter služi i kao izvor napajana od 5V. Budući da ESP8266-12F mikroupravljač za napajanje koristi 3.3V, u sklopu je korišten i AMS11173.3 regulator napona. Zbog veće stabilnost napona uz regulator u paralelu je spojen i kondenzator od 100µF.



Slika 7.4 Slika prikazuje shemu spoja USB-UART adaptera i regulatora napona.

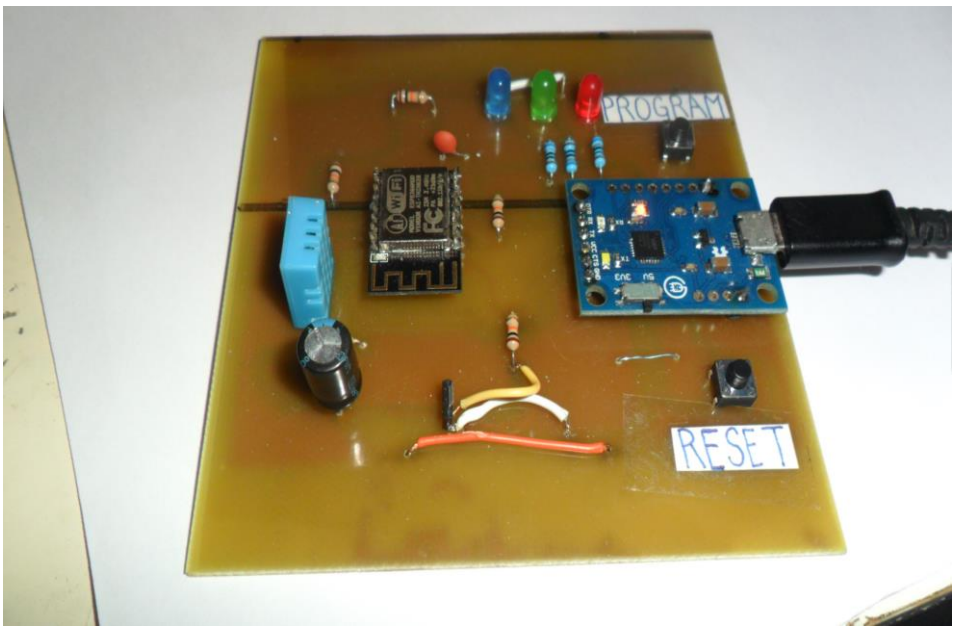
Nadalje u sklopu osim USB-UART adaptera i regulatora napajanja korišteni su elementi poput LED dioda, DHT11 senzora vlage i temperature, dva tipkala, pull-up otpornici od 10KΩ, otpornici od 680Ω za ograničavanje struje kroz LED diode, kao i kondenzator od 100pF koji je spojen između VCC i GND pina mikroupravljača. Pomoću kondenzatora između VCC i GND pina mikroupravljača odvojen je krug mikroupravljača od ostalog djela kruga u kojem se mogu pojaviti smetnje uzrokovane brzom promjenom logičkih stanja u mikroupravljaču.



Slika 7.5 Shema spoja ESP8266 mikroupravljača sa LED diodama, DHT 11 senzorom vlage i temperature, te dva tipkala sa funkcijom ponovnog pokretanja i programiranja.

ESP 8266-12F mikroupravljač ima dva režima rada. U prvom režimu mikroupravljač pokreče program, takozvani (eng. *boot mode*). To je normalan rad mikroupravljača, u tom režimu mikroupravljač prilikom pokretanja odmah izvršava program koji je zapisan u njegovu

memoriju. ESP 8266-12F ovom režimu rada pristupa onda kada je GPIO 0 pin spojen na VCC odnosno visoko logičko stanje. Drugi režim rada je stanje u kojem je mikroupravljač spreman da se u njega zapiše programski kod (eng. *program mode*). U stanje programiranja mikroupravljač ulazi kada je GPIO 0 pin spojen na GND odnosno nisko logičko stanje. Prema shemi iz slike 7.5 odabir između ovih ražima vrši se pomoću tipkala S1 (program). Kada tipka nije pritisnuta, GPIO 0 pin je preko pull-up otpornika R3 spojen na VCC odnosno visoko logično stanje, i mikroupravljač će nakon pokretanja izvršavati programski kod. U slučaju da je tipka pritisnuta zbog otpornika R3 GPIO 0 pin spojen je na GND odnosno nisko logično stanje. Važno je napomenuti će se promjene ovih režima izvršiti tak nakon ponovnog pokretanja mikroupravljača. Za ponovno pokretanje služi tipkalo S2 (reset). Kada tipkalo nije pritisnuto RESET pin je preko pull-up otpornika R5 spojeno na VCC odnosno visoko logičko stanje, te se mikroupravljač neće ponovno pokrenuti. U slučaju kada je tipka pritisnuta RESET pin je spojen na GND odnosno nisko logičko stanje, te se mikroupravljač ponovno pokreče. S obzirom na sve navedeno prije samog zapisivanja programskog koda u mikroupravljač potrebno ga je dovesti u režim rada u kojem je spreman da se u njegovu memoriju zapiše programski kod. To se radi kombinacijom tipki „program“ i „reset“. Odnosno prije samog unosa programskog koda u mikroupravljač potrebno je pritisnuti program tipku, a zatim držeći program tipku pritisnutom, pritisnuti i reset tipku. Nakon toga mikroupravljač je spreman da se u njegovu memoriju zapiše programski kod. Prema shemi iz slike 7.4, kao i 7.5 izrađena je i tiskana pločica.



Slika 7.6 Tiskana pločica za rad sa ESP 8266-12F mikroupravljačem

7.3. Programiranje ESP 8266-12F mikroupravljača

Program korišten kod programiranja mikroupravljača je Arduino IDE. Arduino IDE je poznato programsko razvojno okruženje za programiranje raznih mikroupravljača baziranih na Arduino platformi. S obzirom da je napravljena takozvana Arduino jezgra za ESP 8266 mikroupravljač, postoji mogućnost korištenja Arduino IDE-a za programiranje mikroupravljača baziranih na ESP8266EX integriranom sklopu. Da bi se omogućilo programiranje ESP 8266 mikroupravljača koristeći program Arduino IDE, potrebno je dodati ESP 8266 „pločicu“ u Arduino IDE te postaviti opcije kao što su: brzina procesora, brzina prijenosa programa, veličina memorije ESP 8266 mikroupravljača itd… Sam proces programiranja isti je kao i kod programiranja Arduino mikroupravljača. Arduino program poznatiji kao skica (eng. *sketch*) sastoji se od dva djela. Prvi dio je `void setup`. Ovaj dio izvršit će se prvi i samo jedanput, prilikom pokretanja programa na mikroupravljaču. Drugi dio je `void loop`, odnosno petlja koja se izvršava nakon `void setup`-a. Za povezivanje s Wi-Fi pristupnom točkom, MQTT brokerom te za korištenje DHT11 senzora vlage i temperature, u programu mikroupravljača korištene su programske knjižnice. Programska knjižnica za korištenje funkcija Wi-Fi veze je „ESP8266WiFi“ [13], za rad sa DHT11 senzorom vlage i temperature korištena je knjižnica „DHT senzor library“ [14]. Da bi mikroupravljač imao funkcije MQTT klijenta korištena je programska knjižnica „MQTT“ [15].

7.4. Programski kod ESP 8266-12F mikroupravljača

Budući da je za pristup MQTT brokeru potreban pristup Internet mreži, mikroupravljač se najprije povezuje s Wi-Fi pristupnom točkom usmjeritelja, koji mu omogućava pristup Internetu. Pozivanjem funkcije `connect`, mikroupravljač se najprije povezuje s Wi-Fi pristupnom točkom, a nakon toga s MQTT brokerom. U istoj funkciji, odmah nakon uspostave veze s MQTT brokerom, mikroupravljač se pretplaćuje na temu „LED“.

```
1.  void connect() {
2.    Serial.print("\nPovezivanje na Wi-Fi AP:");
3.    while (WiFi.status() != WL_CONNECTED) {
4.      Serial.print(".");
5.      delay(1000);
6.    }
7.    Serial.print("\nPovezano sa pristupnom točkom: ");
8.    Serial.print(ssid);
9.    Serial.print("\nIP Adresa ESP8266 mikroupravljaca: ");
10.   Serial.print(WiFi.localIP());
11.   Serial.print("\nPovezivanje sa MQTT brokerom...");
12.   while (!client.connect("ESP8266-12F", broker_korisnickoime, broker_lozinka)) {
13.     Serial.print(".");
14.     delay(1000);
15.   }
16.   Serial.print("\nPovezano sa MQTT brokerom!:");
17.   Serial.print(broker_host);
18.   client.publish(tema_oporuka,"",1,0);
```

```
19.   delay (10);
20.   client.subscribe("SINK");
21.   client.subscribe("LED");
22.   client.publish("SINK","101");
23.   delay (10);
24.   }
```

Programski blok 7.1 Connnect funkcija mikroupravljača

Funkcija *connect* prikazana programskim blokom 7.1 poziva se unutar *void setup-a*. Kasnije se u petlji samo provjerava povezanost klijenta te ako klijent nije povezan, ponovno se poziva funkcija za povezivanje. U programu je definirana i takozvana metoda povratnog poziva (eng. *callback*) koja je programski pozvana svaki put kada mikroupravljač primi MQTT poruku.

```
1.  void messageReceived(String &topic, String &payload) {
2.    topic.toCharArray(topic_c, topic.length() + 1);
3.    if (strcmp (topic_c, "LED") == 0) {
4.      int i = payload.toInt();
5.      switch (i) {
6.        case 10:
7.          digitalWrite(Led_crv, LOW);
8.          Serial.print("\nLED crvena OFF");
9.          break;
10.       case 11:
11.         digitalWrite(Led_crv, HIGH);
12.         Serial.print("\nLED crvena ON");
13.         break;
14.       case 20:
15.         digitalWrite(Led_zel, LOW);
16.         Serial.print("\nLED zelena OFF");
17.         break;
18.       case 21:
19.         digitalWrite(Led_zel, HIGH);
20.         Serial.print("\nLED zelena ON");
21.         break;
22.       case 30:
23.         digitalWrite(Led_plv, LOW);
24.         Serial.print("\nLED plava OFF");
25.         break;
26.       case 31:
27.         digitalWrite(Led_plv, HIGH);
28.         Serial.print("\nLED plava ON");
29.         break;
30.     }
31.   }
32.   if (strcmp (topic_c,"SINK" )== 0){
33.     int i = payload.toInt();
34.     switch (i){
35.       case 101:
36.       sinkronizacija=true;
37.       sinkronizacija_pos=false;
38.       }
39.    }
40. }
```

Programski blok 7.2 Funkcija povratnog poziva mikroupravljača (messageReceived)

Funkcija povratnog poziva važna je zbog toga jer „obavještava“ klijenta svaki put kada je objavljena poruka s temom na koju je klijent pretplaćen. Unutar ove funkcije uspoređuju se teme. Ako je objavljena neka poruka na temu „LED“, čita se njen sadržaj te se ovisno o njemu uključuju/isključuju LED diode. U petlji programa (*void loop*), osim provjere stanja veze s MQTT brokerom te ponovnog povezivanja, očitane vrijednost s DHT 11 senzora vlage i

temperature objavljuju se na teme „Temp“, i „Vlaga“. Da bi klijent mogao redovito procesuirati dolazeće poruke i na temelju njih pozivati funkciju povratnog poziva u petlji, također se poziva funkcija *client.loop*.

```
1.  void loop() {
2.    client.loop();
3.    delay(10);
4.    if (!client.connected()) {
5.      connect();
6.    }
7.    stanje_crv = digitalRead (Led_crv);
8.    stanje_zel = digitalRead (Led_zel);
9.    stanje_plv = digitalRead (Led_plv);
10.   povratna_veza();
11.   if (millis() - lastMillis > 3000) {
12.     lastMillis = millis();
13.     float fvlaga = dht.readHumidity();
14.     float ftemp = dht.readTemperature();
15.     temp_s = String(ftemp);
16.     temp_s.toCharArray(temp_c, temp_s.length() + 1);
17.     vlaga_s = String(fvlaga);
18.     vlaga_s.toCharArray(vlaga_c, vlaga_s.length() + 1);
19.     client.loop();
20.     client.publish("temp", temp_c );
21.     Serial.print("\nobjavljujem temperaturu:" + temp_s + "°C");
22.     client.publish("vlaga", vlaga_c);
23.     Serial.print("\nobjavljujem vlažnost zraka:" + vlaga_s + "%");
24.   }
25. }
```

Programski blok 7.3 Glavna petlja mikroupravljača

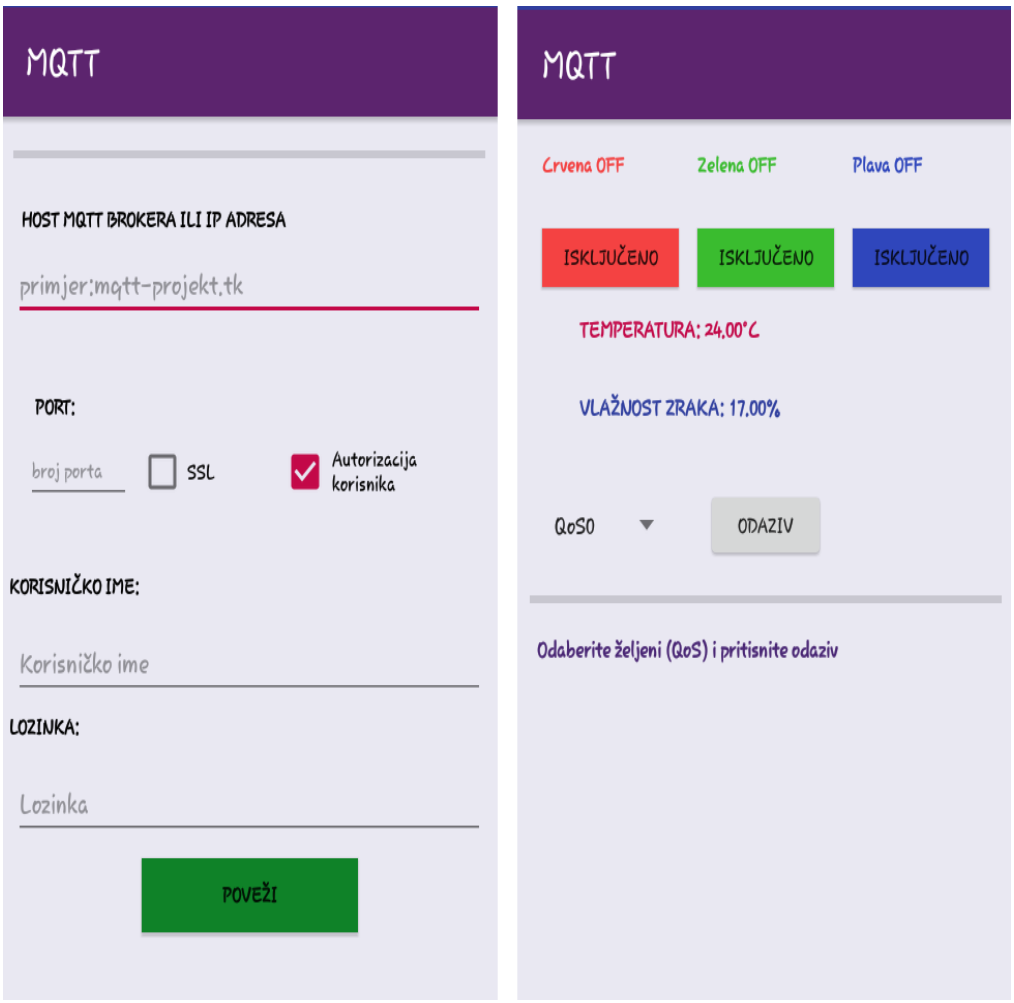
U glavnoj petlji programa očitavaju se stanja GPIO pinova na koje su spojene LED diode, te se poziva funkcija *povratna_veza*. Ove radnje služe za sinkronizaciju stanja LED dioda između mikroupravljača i Android aplikacije.

8. Android aplikacija kao MQTT klijent

Pomoću Android mobilne aplikacije vrši se kontrola ESP 8266-12F mikroupravljača. Android aplikacija koristi Eclipse Paho programsku knjižnicu koja joj daje mogućnost komuniciranja s MQTT brokerom, odnosno daje joj funkcionalnosti MQTT klijenta. Korištenjem funkcija MQTT klijenta, Android aplikacija ima mogućnosti kao što su: povezivanje s MQTT brokerom, objava/pretplata poruka na teme, prekidanje veze s MQTT brokerom, odnosno aplikacija može biti programirana da koristi sve funkcije MQTT protokola. Korisnik aplikacije ima mogućnost uključivanja/isključivanja ulazno/izlaznih pinova mikroupravljača, kao i mogućnost vizualizacije podataka (temperatura i vlažnost zraka) sa senzora mikroupravljača. Android aplikacija podešena je na način da je pretplaćena na teme „Temp“ i „Vlaga“ na koje mikroupravljač objavljuje podatke s DHT 11 senzora temperature i vlage. U aplikaciju su također implementirane metode koje korisnik poziva pritiskom na određene tipke. Pozivanjem tih metoda program objavljuje određenu poruku na temu „LED“. Na istu temu pretplaćen je program mikroupravljača, koji s obzirom na sadržaj poruke, uključuje/isključuje određeni ulazno/izlazni pin.

8.1. Dizajn Android aplikacije

Aplikacija se sastoji od dva *activity-a*. U Android aplikacijama *activity* predstavlja korisničko sučelje (ekran) aplikacije. Prvi *activity* predstavlja sučelje u koje korisnik unosi podatke te se povezuje sa MQTT brokerom. Ovaj *activity* sastoji se od polja za unos podataka, te tipke koja omogućuje povezivanje sa MQTT brokerom. Nakon što korisnik unese podatke, te ako su oni ispravni otvara se drugi *activity*. U njemu korisnik može upravljati sa mikroupravljačem (LED diodama) koristeći tipke, te očitavati vrijednosti sa senzora koji je spojen na njega. Budući da je Android aplikacija sinkronizirana sa mikroupravljačem omogućeno je i praćenje trenutnog stanja LED dioda. Korisnik može testirati odaziv MQTT brokera na svim razinama kvalitete usluge koju podržava MQTT protokol. Aplikacija je postavljena na način da je interakcija sa mikroupravljačem moguća samo ako se koristi MQTT broker postavljen na VPS-u, kojemu se pristupa sa domene *`mqtt-projekt.tk`*. Za sve ostale MQTT brokere korisnik aplikacije ima samo opciju testiranja odaziva.



Slika 8.1 Izgled sučelja osnovnih ekrana Android aplikacije

8.2. Programiranje Android aplikacije

Pri programiranju Android aplikacije korišteno je programsko razvojno okruženje Android Studio. Da bi aplikacija bila u mogućnosti komunicirati s MQTT brokerom, u razvojno okruženje uvezena je vanjska Eclipse Paho programska knjižnica. Eclipse Paho projekt pruža mogućnosti implementacije MQTT protokola na uređaje bazirane na Java ili C programskom jeziku. Eclipse Paho je projekt otvorenog koda. Da bi se uspostavila veza s MQTT brokerom kreiran je MQTT klijent koristeći funkcije Eclipse Paho programske knjižnice.

```
1.  String clientId = MqttClient.generateClientId();
2.          client = new MqttAndroidClient(this.getApplicationContext(), URL_prefix + Broker_URL
    + ":" + port, clientId);
3.          options = new MqttConnectOptions();
4.          if (zahtjev_autorizacije) {
5.              options.setUserName(kor_ime);
6.              options.setPassword(lozinka.toCharArray());
7.          }
8.          options.setCleanSession(false);
9.          options.setAutomaticReconnect(true);
```

Programski blok 8.1 Inicijalizacija MQTT klijenta u Android aplikaciji

Nakon inicijalizacije MQTT klijenta kreće povezivanje sa MQTT brokerom.

```
try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(MainActivity.this, "spojeno!!", Toast.LENGTH_LONG).show();

                    pretplata();
                    sinkronizacija();
                    odaziv_gumb.setVisibility(View.VISIBLE);
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                    Toast.makeText(MainActivity.this, "greška prilikom povezivanja!!!!", Toast
    .LENGTH_LONG   ).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();

        }
```

Programski blok 8.2 Metoda povezivanja Android aplikacije sa MQTT brokerom

Aplikacija također ima definiranu metodu za prekid veze s MQTT brokerom.

```
public void test_odspoji_funkcija() {
        try {
                IMqttToken token = test_client.disconnect();
                token.setActionCallback(new IMqttActionListener() {
                    @Override
                    public final void onSuccess(IMqttToken asyncActionToken) {
                    }
                    @Override
                    public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

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

Programski blok 8.3 Metoda za prekid veze sa MQTT brokerom

S obzirom da Aplikacija ima mogućnost uključivanja/isključivanja LED dioda na mikroupravljaču, definirane su metode pomoću kojih se objavljuju poruke na temu „LED“. Prikazan je primjer realizacije metode za uključivanje/isključivanje crvene LED diode (programski blok 8.4). Za ostale vrijedi isti postupak, jedino što se mijenja je sadržaj poruka.

```
public void pali_gasi_crvenu(View view) {
        Boolean stanje_on = ((ToggleButton) view).isChecked();
        if (stanje_on) {
            String topic = topicLED;
            String message = "11";
            try {
                client.publish(topic, message.getBytes(), 0, false);
                vibrator.vibrate(150);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        } else {
            String topic = topicLED;
            String message = "10";
            try {
                client.publish(topic, message.getBytes(), 0, false);
                vibrator.vibrate(150);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        }
    }
```

Programski blok 8.4 Metoda za uključivanje/isključivanje crvene LED diode

Zbog mogućnosti vizualizacije podataka sa senzora koji je spojen na mikroupravljač u Android aplikaciji, definirana je metoda pretplate na teme od interesa.

```
public void pretplata() {
        try {
            client.subscribe(topicStanje, 0);
            client.subscribe(topicTEMP, 0);
            client.subscribe(topicVLAGA, 0);
            client.subscribe(topicSINK_POV1, 0);
            client.subscribe(topicSINK_POV2, 0);
            client.subscribe(topicSINK_POV3, 0);
            client.subscribe(topicOPORUKA, 0);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
```

Programski blok 8.5 Metoda pretplate na teme

Manipulacija s dolaznim porukama riješena je metodom povratnog poziva (eng. *callback*). Ova metoda ima funkciju konstantnog „nadzora“ poruka koje su objavljene na pretplaćene teme. Primjer: ukoliko dođe poruka na temu „Temp“ pomoću *if* uvjetnog grananja izdvaja se tema, te se sadržaj poruke upisuje na mjesto predviđeno za prikaz temperature. Isti postupak vrijedi i za vlažnost zraka.

```
public void messageArrived(String topic, MqttMessage message) throws Exception {
            if (topic.equals(topicTEMP)) {
                prikaztemp.setText("TEMPERATURA: " + new String(message.getPayload()) + "°C
");
              }
              if (topic.equals(topicVLAGA)) {
                  prikazvlaga.setText("VLAŽNOST ZRAKA: " + new String(message.getPayload()) +
 "%");
              }
        }
```

Programski blok 8.6 Metoda povratnog poziva u Android aplikaciji

8.3. Metoda mjerenja odaziva MQTT protokola korištena u Android aplikaciji

Android aplikacija osim kontrole i vizualizacije mikroupravljača ima mogućnost mjerenja odaziva MQTT brokera. Odaziv MQTT brokera je vrijeme između slanja i primanja MQTT poruke. Ova metoda programski je realizirana na način da se računa razlika vremena između primanja i slanja poruke. Pozivanjem metode `odaziv_funkcija` prvo se izvršava programski kod pomoću kojega se klijent pretplaćuje na temu odaziva. Nakon što je klijent pretplaćen izvršava se kod u pozadini Android aplikacije, takozvani AsyncTask. AsyncTask java razred omogućuje izvođenje određene funkcionalnosti u radnoj dretvi (eng. *worker thread*).

```
1.  public void odaziv_funkcija(View view) {
2.          try {
3.              client.subscribe(topicODAZIV, QoS_odaziva);
4.          } catch (MqttException e) {
5.              e.printStackTrace();
6.          }
7.          new Radi_u_pozadini().execute();
8.      }
```

Programski blok 8.7 Metoda za inicijalizaciju mjerenja odaziva MQTT brokera

Korištenje AsyncTask-a zahtjeva implementaciju metoda kao što su: *`onPreExecute()`*, *`onProgressUpdate`*, *`doInBackground`* i *`onPostExecute`*. Prilikom izvođenja AsyncTask-a najprije će biti pozvana metoda *`onPreExecute()`*. U ovoj metodi obavljena je inicijalizacija varijabli te postavljanje maksimalne vrijednosti trake napretka, kao i pražnjenje lista za upis vrijednosti odaziva. Liste je potrebno isprazniti da bi se spriječio konflikt trenutnih sa budućim podacima u listi. Isto tako zabranjuje se interakcija sa tipkom odaziv (tipka pomoću koje se poziva AsyncTask).

```
1. @Override
2. protected void onPreExecute() {
3.     super.onPreExecute();
4.     lista_rezultata.clear();
5.     sadržaj_odaziva = 100;
6.     traka_napratka.setMax(9);
7. }
```

Programski blok 8.8 AsyncTask metoda onPreExecute

Sljedeća metoda u AsyncTask-u je *`doInBackground`*. U ovoj metodi izvršavat će se zadatak mjerenja odaziva MQTT brokera. Odaziv je mjeren na način da se pomoću *`for`* petlje objavi deset poruka na temu „Odaziv“, a kao sadržaj zapisuje se vrijednost brojača *`for`* petlje. Zatim se mjeri vrijeme potrebno da objavljene poruke budu primljene. Mjerenje vremena programski je realizirano pozivanjem metode *`System.nanoTime()`*, odnosno metode koja vrača trenutno vrijeme najpreciznijeg sistemskog brojača mjereno u nano sekundama. Korištenjem ove metode trenutno stanje brojača, zapisuje se u varijablu *`t_trenutno_poslano`* nakon samoga

objavljivanja poruke. S obzirom da se u funkciji povratnog poziva registrira dolazak objavljenih poruka, kao i onih koje su objavljene na temu „Odaziv“. Sadržaj dolaznih poruka sa teme „Odaziv“ sprema se u varijablu *sadržaj_odaziva*. Prilikom slanja poruke , unutar *for* petlje korištena je prazna *while* petlja. Praznom *while* petljom odrađeno je čekanje promjene varijable *sadržaj_odaziva*, odnosno čeka se da poruka bude primljena, što znači i izlazak iz *while* petlje. Nakon *while* petlje računa se razlika između trenutnog vremena i vremena slanja poruke, odnosno *t_trenutno_poslano*. Ta razlika predstavlja odaziv MQTT brokera i sprema se u listu rezultata, gdje se podatak (razlika vremena) unutar liste indeksira koristeći vrijednost brojača *for* petlje. Prije samog završetka ciklusa for petlje ažurirana je vrijednost trake napretka, kojom se prati napredak izvršavanja AsyncTask-a. Također na samom početku *for* petlje poziva se metoda *Thred.sleep* kojom je osigurano da se poruke šalju svakih pola sekunde (500 ms).

```
1.  @Override
2.          protected Boolean doInBackground(Void... voids) {
3.              for (int i = 0; i < 10; i++) {
4.                  try {
5.                      Thread.sleep(500);
6.                  } catch (InterruptedException e) {
7.                      e.printStackTrace();
8.                  }
9.                  String topic = topicODAZIV;
10.                 String message = Integer.toString(i);
11.
12.                 try {
13.                     client.publish(topic, message.getBytes(), QoS_odaziva, false);
14.                 } catch (MqttException e) {
15.                     e.printStackTrace();
16.                 }
17.                 t_trenutno_poslano = System.nanoTime() / djelitelj;
18.                 while (i != sadržaj_odaziva) {
19.                 }
20.                 double odaziv = (System.nanoTime() / djelitelj) - t_trenutno_poslano;
21.                 lista_rezultata.add(i, odaziv);
22.                 publishProgress(i);
23.             }
24.             return null;
25.         }
```

Programski blok 8.9 AsyncTask metoda doInBackground

Korištenjem metode *onProgressUpdate* osigurano je ažuriranje trake napretka u glavnoj dretvi java programa (eng. *UI Thread*).

```
1.  @Override
2.          protected void onProgressUpdate(Integer... values) {
3.              super.onProgressUpdate(values);
4.              traka_napratka.setProgress(values[0]);
5.              odaziv_gumb.setClickable(false);
6.          }
```

Programski blok 8.10 AsyncTask metoda onProgressUpdate

Nakon metode *doInBackground*, poziva se *onPostExecute* metoda u kojoj se iz rezultata odaziva računa prosjek odaziva kao i standardna devijacija.

```
protected void onPostExecute(Boolean aBoolean) {
            super.onPostExecute(aBoolean);
            List<Double> lista_rez_zaokr =new ArrayList<>();
            for (int j=0;j<lista_rezultata.size();j++){
                double zaokr_vrj =Precision.round(lista_rezultata.get(j),2,BigDecimal.ROUND_HAL
F_UP);
                lista_rez_zaokr.add(j,zaokr_vrj);
            }
            double sum = 0;
            for (int k = 0; k < lista_rez_zaokr.size(); k++) {
                sum += lista_rez_zaokr.get(k);
            }
            double prosjek = sum / lista_rez_zaokr.size();
            List<Double> lista_razlike = new ArrayList<>();
            double SD_razlika;
            for (int x = 0; x < lista_rez_zaokr.size(); x++) {
                SD_razlika = lista_rez_zaokr.get(x) - prosjek;
                lista_razlike.add(x, SD_razlika);
            }
            List<Double> lista_kvadrata = new ArrayList<>();
            double kvadrati;
            for (int z = 0; z < lista_razlike.size(); z++) {
                kvadrati = lista_razlike.get(z) * lista_razlike.get(z);
                lista_kvadrata.add(kvadrati);
            }
            double zbroj_kvadrata = 0;
            for (int y = 0; y < lista_kvadrata.size(); y++) {
                zbroj_kvadrata += lista_kvadrata.get(y);
            }
            double rez = zbroj_kvadrata / (lista_rez_zaokr.size() - 1);
            double st_devijacija = Math.sqrt(rez);
            double pros_zaokr = Precision.round(prosjek,2,BigDecimal.ROUND_HALF_UP);
            double st_dev_zakr = Precision.round(st_devijacija,2,BigDecimal.ROUND_HALF_UP);
            prikazodaziv.setText("REZULTATI ODAZIVA [ms]: "+"\n"+lista_rez_zaokr);
            prikaz_pros.setText("PROSJEK ODAZIVA: "+pros_zaokr+" ms");
            prikaz_st_dev.setText("STANDARDNA DEVIJACIJA: "+st_dev_zakr);
            odaziv_gumb.setClickable(true);
        }
```

Programski blok 8.11 AsyncTask metoda onPostExecute

8.4. Postavljanje oporuke

Da bi se korisniku Android aplikacije signalizirala graška na mikroupravljaču, na njemu je postavljena oporuka. Budući da je oporuka normalna MQTT poruka koju ima svoju temu, kvalitetu usluge itd… Nju šalje MQTT broker automatski čim primijeti da je klijent prekinuo vezu na neočekivan način. Oporuka je postavljena prije samog povezivanja mikroupravljača sa MQTT brokerom.

```
client.setWill(tema_oporuka,poruka_oporuka,1,2);
```

Programski blok 8.12 Postavljanje oporuke na mikroupravljaču

Oporuka je postavljena kao zadržana što znači da će zadnja poruka na temu oporuke biti spremljena. Ovime je omogućeno da aplikacija dobije obavijest čim se pretplati na temu oporuke.Nakon svakog spajanja sa MQTT brokerom mikroupravljač je postavljen tako da prebriše zadnju zadržanu oporuku na način da na temu oporuke objavi praznu poruku sa postavljenom zastavicom zadržavanja.

```
client.publish(tema_oporuka,"",1,0);
```

Programski blok 8.13 Brisanje zadržane oporuke

Tema oporuke je „ESP_oporuka“, a sadržaj poruka oporuke je „RIP“. Budući da Android aplikacija u svojoj metodi povratnog poziva prati poruke na temu „ESP_oporuka“, kada na tu temu dođe poruka sadržaja „RIP“ ispisuje se takozvana `Toast` poruka „ESP GREŠKA“. S obzirom da postoji mogućnost da se veza prekinula zbog brzog pritiskanja tipki za uključivanje/isključivanje LED dioda, nakon primitka poruke „RIP“ aplikacija je postavljena tako da koristi `AsyncTask` za čekanje da se klijent ponovno poveže. Ukoliko je vrijeme čekanja prošlo a oporuka je još uvijek aktivirana (nije se prebrisala prilikom ponovnog povezivanja mikroupravljača sa MQTT brokerom) aplikacija će javiti grešku tj. ispisat će se poruka „ESP GREŠKA“.

```
if (topic.equals(topicOPORUKA)) {
                        sadržaj_oporuke = new String(message.getPayload(), "UTF8");
                        if (sadržaj_oporuke.equals("RIP")) {
                               new čekaj().execute();
                            }
                    }
```

Programski blok 8.14 Manipulacija sa dolaznim porukama ne temu „Oporuka“

```
class čekaj extends AsyncTask<Void,Integer,Integer>{
        @Override
        protected Integer doInBackground(Void... voids) {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            return null;
        }
        @Override
        protected void onPostExecute(Integer integer) {
            super.onPostExecute(integer);
            if (sadržaj_oporuke.equals("RIP")) {
                Toast.makeText(MainActivity.this, "ESP GREŠKA !!", Toast.LENGTH_LONG).show();

            }
        }
    }
```

Programski blok 8.15 AsyncTask čekanje da oporuka bude prebrisana

9. Analiza i mjerenja u implementiranom sustavu

U ovom poglavlju opisana je metoda sinkronizacije između Android uređaja i mikroupravljača. S obzirom da je u Android aplikaciju implementirana mogućnost mjerenja odaziva MQTT brokera, u poglavlju je također izvršena analiza prikupljenih rezultata odaziva, kao i analiza skalabilnosti MQTT brokera.

9.1. Sinkronizacija između Android aplikacije i mikroupravljača

Da bi korisnik koristeći Android aplikaciju imao uvid u trenutna stanja LED dioda, Android aplikacija i mikroupravljač su sinkronizirani. Nakon što se Android uređaj uspješno poveže sa MQTT brokerom poziva se metoda *pretplata()*, a nakon nje *sinkronizacija()*.

```
1.  public void onSuccess(IMqttToken asyncActionToken) {
2.                      Toast.makeText(MainActivity.this, "spojeno!!", Toast.LENGTH_LONG).show();
3.                      pretplata();
4.                      sinkronizacija();
5.                      odaziv_gumb.setVisibility(View.VISIBLE);
6.                  }
```

Programski blok 9.1 Metoda pretplata i sinkronizacija nakon uspješnog povezivana sa MQTT brokerom (onSuccess metoda)

Pozivanjem metode *pretplata()*, između ostalih, Android uređaj pretplaćuje se na temu „SINK“.

```
1.  public void pretplata() {
2.          try {
3.              client.subscribe(topicStanje, 0);
4.              client.subscribe(topicTEMP, 0);
5.              client.subscribe(topicVLAGA, 0);
6.              client.subscribe(topicSINK_POV1, 0);
7.              client.subscribe(topicSINK_POV2, 0);
8.              client.subscribe(topicSINK_POV3, 0);
9.              client.subscribe(topicOPORUKA, 0);
10.         } catch (MqttException e) {
11.             e.printStackTrace();
12.         }
13.     }
```

Programski blok 9.2 Metoda pretplate u Android aplikaciji

Nakon pretplate izvršava se metoda *sinkronizacija()*, u kojoj se objavljuje MQTT poruka sadržaja „101“ na temu „SINK“.

```
1.  public void sinkronizacija() {
2.          String topic = topicSINK;
3.          String message = ("101");
4.          try {
5.              client.publish(topic, message.getBytes(), 0, false);
6.          } catch (MqttException b) {
7.              b.printStackTrace();
8.          }
9.      }
```

Programski blok 9.3 Prikaz metode „sinkronizacija“ u Android aplikaciji

Budući da je mikroupravljač pretplaćen da temu „SINK“, čim zaprimi poruku sadržaja „101“ programski se izaziva promjena varijable *sinkronizacija*, kao i varijable *sinkronizacija_pos*.

```
1.  if (strcmp (topic_c,"SINK" )== 0){
2.      int i = payload.toInt();
3.      switch (i){
4.        case 101:
5.        sinkronizacija=true;
6.        sinkronizacija_pos=false;
7.        }
8.     }
```

Programski blok 9.4 Manipulacija sa dolaznom porukom sa teme „SINK“ u programu mikroupravljača

U programu mikroupravljača definirana je metoda *povratna_veza()*. U toj metodi mikroupravljač „očitava“ promjene na ulazno/izlaznim pinovima (pinovima na koje su spojene LED diode), kao i promjenu varijable *sinkronizacija*. Čim se dogodi neka promjena mikroupravljač će objaviti MQTT poruke sa trenutnim stanjima LED dioda na sljedeće teme: „SINK_POVRATNO1“, „SINK_ POVRATNO2“, „SINK_ POVRATNO3“. S obzirom da je Android aplikacija također pretplaćena na te teme, u funkciji povratnog poziva moguće je procesuirati poruke sa tih tema i prema njihovom sadržaju ažurirati status Led dioda, kao i stanje tipki za njihovo uključivanje/isključivanje.

```
void povratna_veza() {
  if (stanje_crv != stanje_crv_pos) {
    if (stanje_crv == HIGH) {
      client.publish("Stanje", "41");
    } else {
      client.publish("Stanje", "40");
    }
    delay (50);
    stanje_crv_pos = stanje_crv;
  }
  if (stanje_zel != stanje_zel_pos) {
    if (stanje_zel == HIGH) {
      client.publish("Stanje", "51");
    } else {
      client.publish("Stanje", "50");
    }
    delay (50);
    stanje_zel_pos = stanje_zel;
  }
  if (stanje_plv != stanje_plv_pos) {
    if (stanje_plv == HIGH) {
      client.publish("Stanje", "61");
    } else {
      client.publish("Stanje", "60");
    }
    delay (50);
    stanje_plv_pos = stanje_plv;
  }
  if (sinkronizacija!=sinkronizacija_pos){
   stanje_crv_s= String (stanje_crv);
   stanje_zel_s= String (stanje_zel);
   stanje_plv_s= String (stanje_plv);
   stanje_crv_s.toCharArray(stanje_crv_ch,stanje_crv_s.length()+1);
   stanje_zel_s.toCharArray(stanje_zel_ch,stanje_zel_s.length()+1);
   stanje_plv_s.toCharArray(stanje_plv_ch,stanje_plv_s.length()+1);
   client.publish("SINK_POVRATNO1",stanje_crv_ch);
   delay (50);
   client.publish("SINK_POVRATNO2",stanje_zel_ch);
   delay (50);
   client.publish("SINK_POVRATNO3",stanje_plv_ch);
   delay (50);
   sinkronizacija_pos=sinkronizacija;
   }
}
```

Programski blok 9.5 Funkcija „povratna veza“ u mikroupravljaču

Metoda „*povratna_veza()*“ poziva se u glavnoj petlji mikroupravljača, tako da će se stanja LED dioda ažurirati čim se dogodi promjena njihovog stanja. Ako se dogodi promjena LED diode mikroupravljač programski objavljuje MQTT poruke na temu „Stanje“ koje opet može očitati Android aplikacija jer je pretplaćena na tu temu. Sinkronizacija se odvija samo kad se Android klijent povezuje sa MQTT brokerom, dok se poruke na temu „Stanje“ programski objavljuju za svaku promjenu stanja LED dioda. Procesiranje poruka sa tema vezanih za sinkronizaciju, kao i poruka sa teme „Stanje“ prikazano je programskim blokovima 9.6 i 9.7.

```
1.  if (topic.equals(topicSINK_POV1)) {
2.                      String poruka_sink_pov1 = new String(message.getPayload(), "UTF8");
3.                      ToggleButton tipka_crvena = (ToggleButton) findViewById(R.id.toggleButton3
    );
4.                      SINK_LED_STANJE_crv = Integer.parseInt(poruka_sink_pov1);
5.                      switch (SINK_LED_STANJE_crv) {
6.                          case 1:
7.                              led_crv_stanjetxt.setText("Crvena ON");
8.                              tipka_crvena.setChecked(true);
9.                              break;
10.                         case 0:
11.                             led_crv_stanjetxt.setText("Crvena OFF");
12.                             tipka_crvena.setChecked(false);
13.                             break;
14.                     }
15.                 } else if (topic.equals(topicSINK_POV2)) {
16.                     String poruka_sink_pov2 = new String(message.getPayload(), "UTF8");
17.                     ToggleButton tipka_zelena = (ToggleButton) findViewById(R.id.toggleButton)
    ;
18.                     SINK_LED_STANJE_zel = Integer.parseInt(poruka_sink_pov2);
19.                     switch (SINK_LED_STANJE_zel) {
20.                         case 1:
21.                             led_zel_stanjetxt.setText("Zelena ON");
22.                             tipka_zelena.setChecked(true);
23.
24.                             break;
25.                         case 0:
26.                             led_zel_stanjetxt.setText("Zelena OFF");
27.                             tipka_zelena.setChecked(false);
28.                             break;
29.                     }
30.                 } else if (topic.equals(topicSINK_POV3)) {
31.                     String poruka_sink_pov3 = new String(message.getPayload(), "UTF8");
32.                     ToggleButton tipka_plava = (ToggleButton) findViewById(R.id.toggleButton2)
    ;
33.                     SINK_LED_STANJE_plv = Integer.parseInt(poruka_sink_pov3);
34.                     switch (SINK_LED_STANJE_plv) {
35.                         case 1:
36.                             led_pla_stanjetxt.setText("Plava ON");
37.                             tipka_plava.setChecked(true);
38.                             break;
39.                         case 0:
40.                             led_pla_stanjetxt.setText("Plava OFF");
41.                             tipka_plava.setChecked(false);
42.                     }
43.                 }
```

Programski blok 9.6 Ažuriranje statusa LED dioda i tipki za njihovo upravljanje, koristeći dolazne poruke sa tema: SINK_POVRATNO1, SINK_POVRATNO2, SINK_POVRATNO3

```
1.  if (topic.equals(topicStanje)) {
2.                      String poruka_stanja = new String((message.getPayload()), "UTF8");
3.                      ToggleButton tipka_crvena = (ToggleButton) findViewById(R.id.toggleButton3
    );
4.                      ToggleButton tipka_zelena = (ToggleButton) findViewById(R.id.toggleButton)
    ;
5.                      ToggleButton tipka_plava = (ToggleButton) findViewById(R.id.toggleButton2)
    ;
6.                      trenutno_stanje_LED = Integer.parseInt(poruka_stanja);
7.                      switch (trenutno_stanje_LED) {
8.                          case 41:
9.                              led_crv_stanjetxt.setText("Crvena ON");
10.                             tipka_crvena.setChecked(true);
11.                             break;
12.                         case 40:
13.                             tipka_crvena.setChecked(false);
14.                             led_crv_stanjetxt.setText("Crvena OFF");
15.                             break;
16.                         case 51:
17.                             led_zel_stanjetxt.setText("Zelena ON");
18.                             tipka_zelena.setChecked(true);
19.                             break;
20.                         case 50:
21.                             led_zel_stanjetxt.setText("Zelena OFF");
22.                             tipka_zelena.setChecked(false);
23.                             break;
24.                         case 61:
25.                             led_pla_stanjetxt.setText("Plava ON");
26.                             tipka_plava.setChecked(true);
27.                             break;
28.
29.                         case 60:
30.                             led_pla_stanjetxt.setText("Plava OFF");
31.                             tipka_plava.setChecked(false);
32.                             break;
33.                     }
34.                 }
```

Programski blok 9.7 Ažuriranje statusa LED dioda i tipki za njihovo upravljanje, koristeći dolazne poruke sa teme „Stanje“

Kod ovakvog pristupa sinkronizaciji Android uređaj nakon što se poveže sa MQTT brokerom ažurira stanja LED dioda i njihovih tipki u upravljačkoj aplikaciji. U slučaju da se mikroupravljač resetira, korisnik Android aplikacije trebao bi se ponovno povezati sa MQTT brokerom da bi se opet sinkronizirao sa mikroupravljačem. Ovaj problem riješen je na način da mikroupravljač u `connect()` funkciji (programski blok 7.1) nakon povezivanja sa MQTT brokerom objavljuje poruku sadržaja „101“ na temu „SINK“, nakon čega kraće sinkronizacija kao da je dobio istu poruku od Android aplikacije.

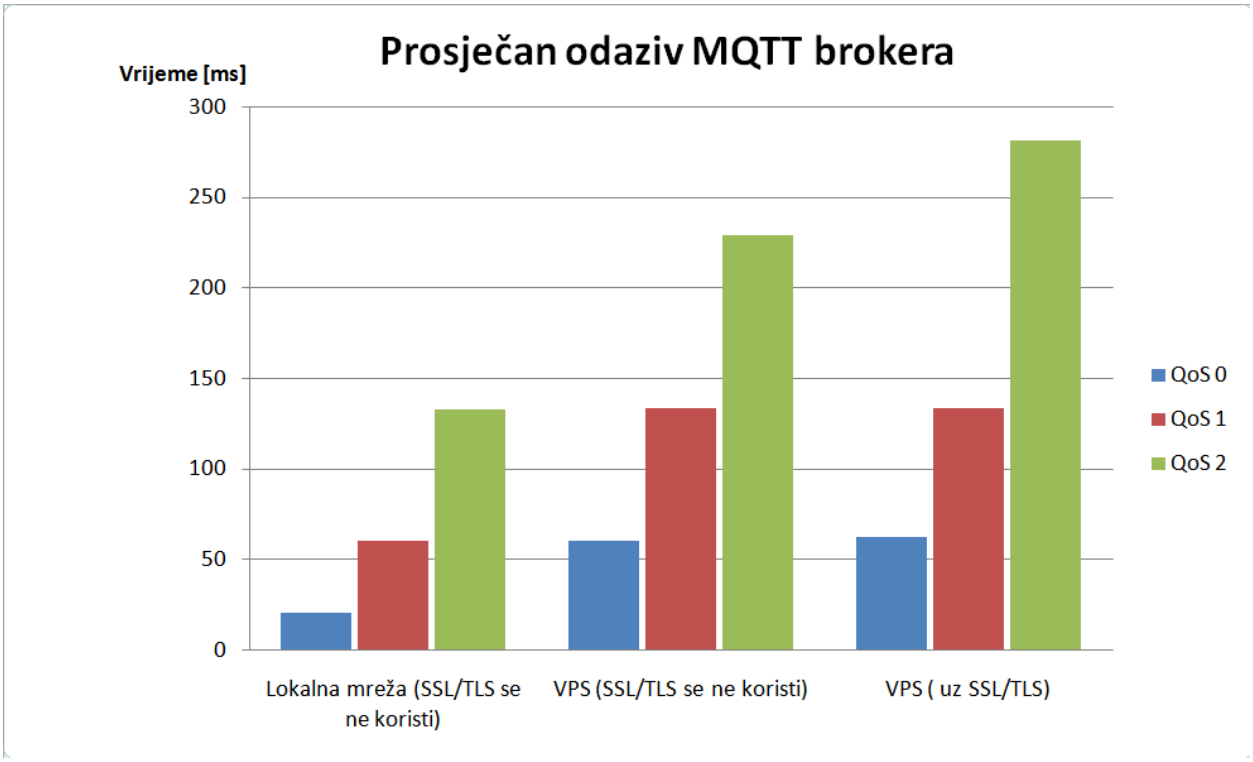
9.2. Rezultati odaziva MQTT brokera

Rezultati su prikupljani koristeći metodu mjerenja odaziva pomoću Android aplikacije, te je objavljeno deset poruka na temu „Odaziv“. Poruke su objavljivane u razmacima od pola sekunde, te je mjereno proteklo vrijeme od objave poruke do njenoga primitka. MQTT broker korišten kod mjerenja odaziva je Mosquitto, koji je postavljen na lokalnoj mreži kao i VPS-u.

Mreža	**Lokalna**			**Virtualni privatni poslužitelj**					
SSL/TLS	Ne			Ne			Da		
Kvaliteta usluge (QoS)	QoS 0	QoS 1	QoS 2	QoS 0	QoS 1	QoS 2	QoS 0	QoS 1	QoS 2
Vrijeme 1 [ms]	21.42	22.13	106.44	50.29	95.86	243.67	54.18	95.34	234.30
Vrijeme 2 [ms]	17.43	55.19	112.36	94.99	151.89	226.34	54.93	228.09	238.29
Vrijeme 3 [ms]	17.97	112.63	102.54	57.31	141.26	236.16	55.06	97.13	225.79
Vrijeme 4 [ms]	16.80	36.31	87.46	53.38	150.92	220.86	53.42	138.27	240.48
Vrijeme 5 [ms]	18.27	58.98	291.95	58.96	109.55	221.80	48.44	132.45	721.82
Vrijeme 6 [ms]	39.56	71.44	293.69	69.06	132.70	225.71	57.05	93.24	235.67
Vrijeme 7 [ms]	21.20	70.48	89.65	54.65	130.47	225.34	54.96	140.67	226.94
Vrijeme 8 [ms]	18.81	59.98	79.65	52.36	142.62	232.31	60.15	131.31	237.70
Vrijeme 9 [ms]	17.49	57.49	79.17	57.31	139.65	227.93	59.86	150.52	240.62
Vrijeme 10 [ms]	20.43	60.62	81.93	55.95	139.35	232.40	125.83	131.01	219.90
Prosjek [ms]	**20.94**	**60.53**	**132.73**	**60.43**	**133.73**	**229.25**	**62.39**	**133.80**	**281.45**
Standardna devijacija	**6.74**	**23.63**	**85.11**	**13.17**	**17.79**	**6.98**	**22.54**	**39.05**	**154.88**

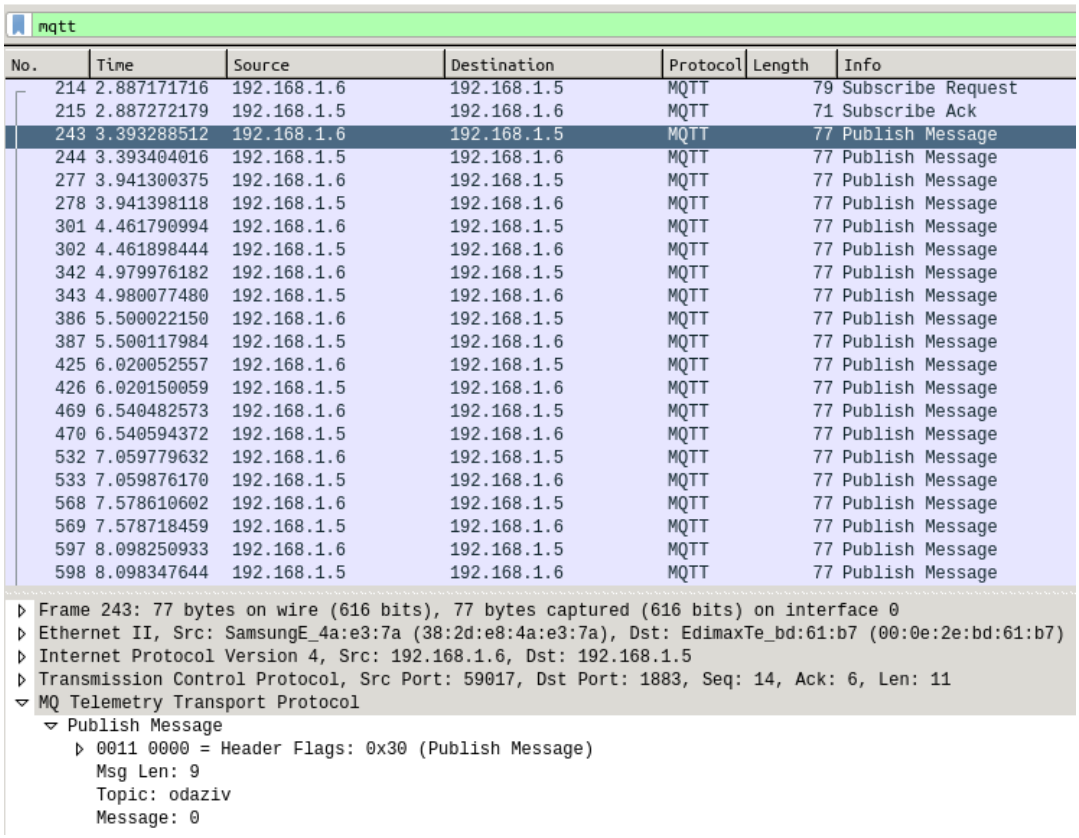
Tabela 9.1 Rezultati mjerenja odaziva MQTT brokera.

S obzirom da je teret (sadržaj) poruka objavljenih na temu odaziv vrijednost brojača for petlje koja je u rasponu između nula i devet, osigurano je da sve poruke imaju istu dužinu sadržaja (1 bajt).



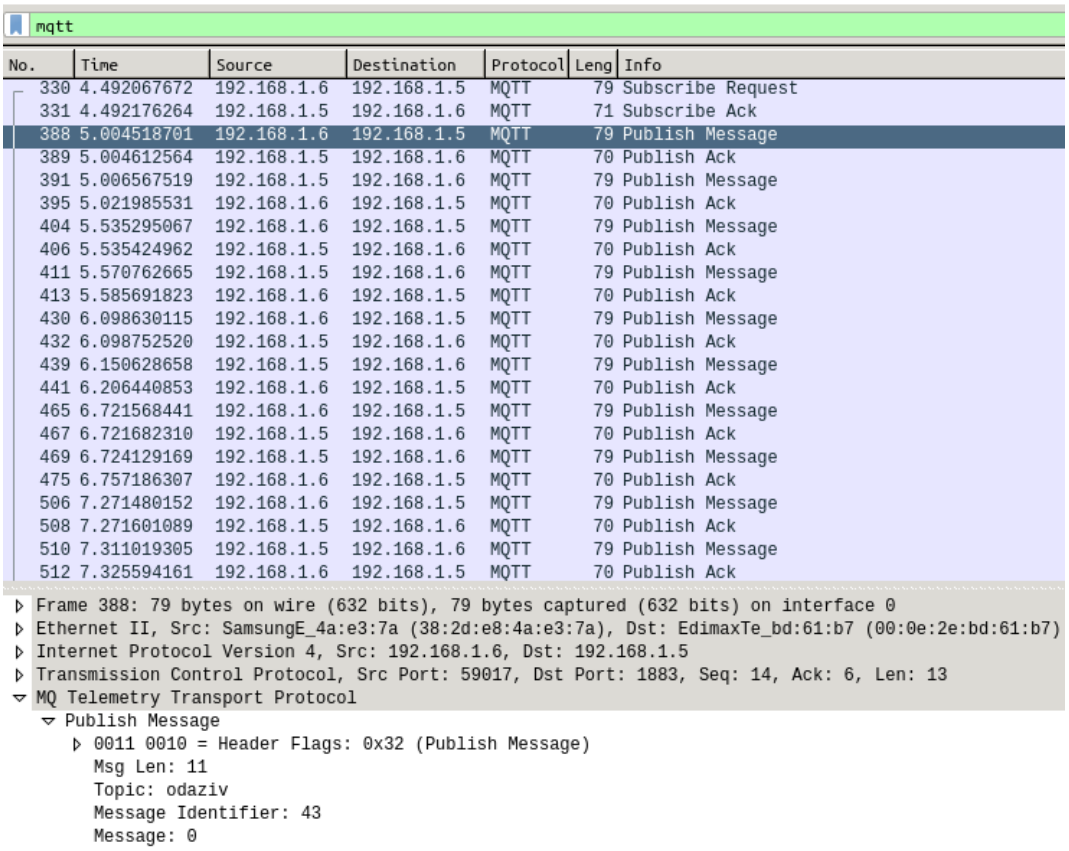
Slika 9.1 Dijagram prosječnog odaziva MQTT brokera

Iz dijagrama se vidi da odaziv MQTT brokera najviše ovisi o odabranoj razini kvalitete usluge, kao i propusnosti mreže. S obzirom na korištenje TLS/SSL sigurnosnog protokola razlike su zanemarive, što je još jedan razlog više za korištenje ove sigurnosne značajke. U ovom slučaju mjerenja odaziva MQTT broker ima samo jednu aktivnu vezu (povezan je samo sa Android uređajem). U slučaju više aktivnih veza bilo bi i veće opterećenje MQTT brokera, a samim time u obzir bi valjalo uzeti i konfiguraciju računala na kojem je instaliran MQTT broker. Prilikom mjerenja odaziva na lokalnoj mreži snimljen je mrežni promet pomoću programa Wireshark.



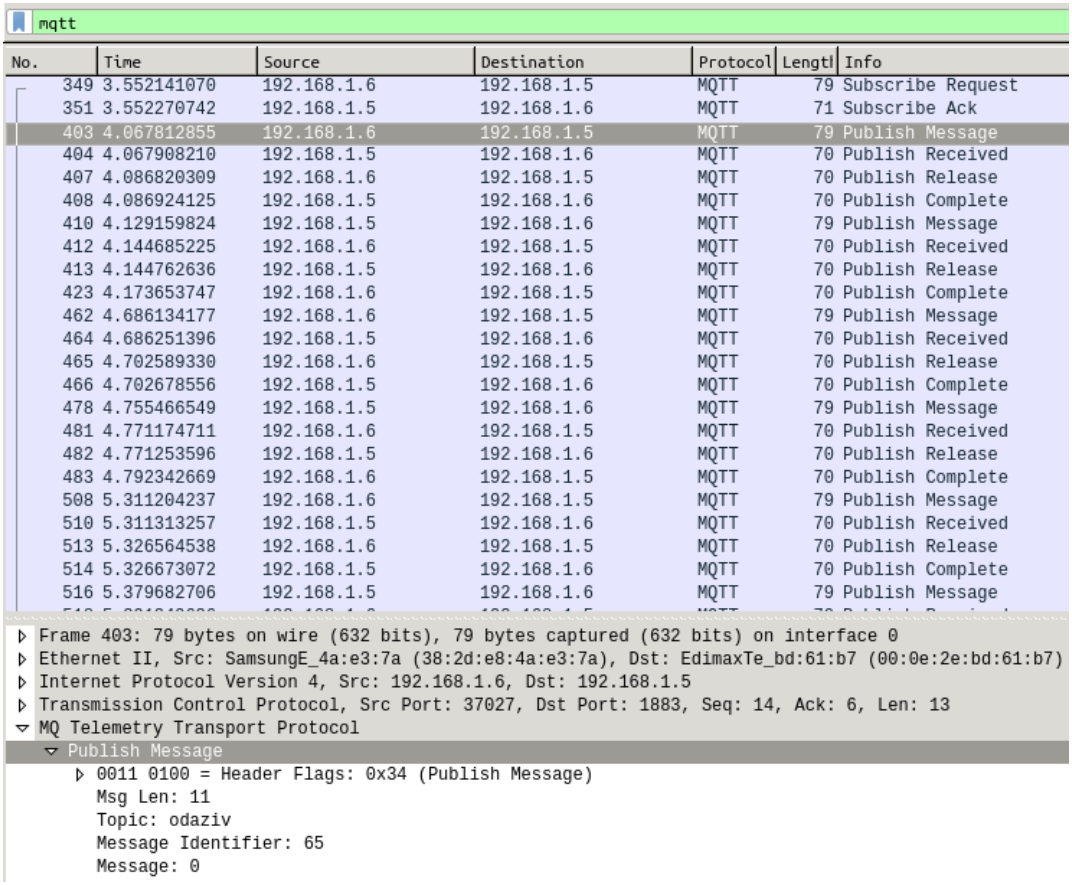
Slika 9.2 Snimljeni MQTT paketi prilikom mjerenja odaziva MQTT brokera na lokalnoj mreži sa kvalitetom usluge QoS 0

Prema snimljenim paketima MQTT protokola prikazanih na slici 9.2 primjećuje se da je prvi paket koji šalje klijent (IP 192.168.1.6) zahtjev za pretplatom na temu „Odaziv“. Nakon priznavanja pretplate od strane MQTT brokera (IP 192.168.1.5) klijent šalje prvu poruku (**PUBLISH** paket). S obzirom da je klijent ujedno i pretplaćen na temu odaziv MQTT broker prosljeđuje poruku natrag klijentu. Što znači da klijent šalje jednu poruku a MQTT broker tu poruku dalje razdjeljuje pretplaćenim klijentima, što je bitno jer se time smanjuje opterećenje klijenta (mrežnu komunikaciju preuzima MQTT broker). Budući da se kod mjerenja odaziva kao sadržaj poruke upisuje vrijednost brojača for petlje koji može poprimiti vrijednost od nule do devet (jedan znak), time MQTT **PUBLISH** paketi imaju istu duljinu. Zbog čega cijeli Ethernet okvir u koji je enkapsuliran i MQTT **PUBLISH** paket ima duljinu 77 bajta. Mrežni promet također je snimljen prilikom mjerenja odaziva MQTT brokera na višim nivoima kvalitete usluge.



Slika 9.3 Snimljeni MQTT paketi prilikom mjerenja odaziva MQTT brokera na lokalnoj mreži sa kvalitetom usluge QoS 1

Iz Analize snimljenog mrežnog prometa korištenjem QoS 1 prilikom mjerenja odaziva vidi se da sada uz **PUBLISH** MQTT paket MQTT broker šalje i potvrdu **PUBACK**. Prema slici 9.3 MQTT klijent nakon potvrde za pretplatom na temu odaziv šalje prvi **PUBLISH** MQTT paket. S obzirom da se koristi QoS 1 MQTT broker potvrđuje primitak poruke sa **PUBACK** paketom, te ju prosljeđuje natrag klijentu, budući da je klijent pretplaćen na temu „Odaziv“, te se između klijenta i MQTT brokera sada razmjenjuje veći broj paketa i sam odaziv MQTT brokera je sporiji. Za razliku od prijašnje slike 9.2 sada se može primijetiti razlika u dužini Ethernet okvira u koji je enkapsuliran MQTT **PUBLISH** paket, iako je dužina poruke ista. To je uzrokovano time što se koristi viši nivo kvalitete usluge i zbog toga je u fiksnom zaglavlju **PUBLISH** paketa postavljena zastavica kvalitete usluge za što su iskorištena dva bita. Budući da MQTT koristi UTF8 zapis gdje se znak prikazuje kao bajt, sada Ethernet okvir u kojem je enkapsuliran **PUBLISH** kontrolni paket iznosi 79 bajta. Isto vrijedi i za QoS 2, ali zbog četverostrukog rukovanja između klijenta i MQTT brokera razmjenjuje se još veći broj paketa što rezultira sporijim odazivom MQTT brokera.



Slika 9.4 Snimljeni MQTT paketi prilikom mjerenja odaziva MQTT brokera na lokalnoj mreži sa kvalitetom usluge QoS 2

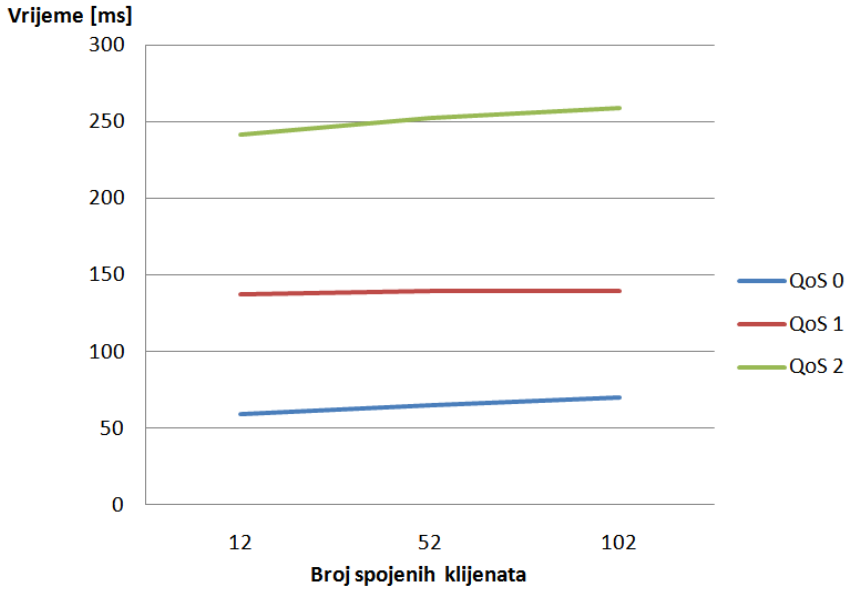
9.3. Skalabilnost MQTT brokera

U ovom odlomku napravljen je test skalabilnost MQTT brokera. MQTT broker opterećen je sa 12, 52, 102 klijenta, uz mjerenje odaziva praćeno je i opterećenje procesora kao i radne memorije računala na kojem je postavljen MQTT broker. Broker korišten prilikom ovog testa je Mosquitto postavljen na VPS-u. Svi klijenti koji se povezuju sa MQTT brokerom koriste sigurnosni SSL/TLS protokol. Dva osnovna klijenta su ESP 8266-12F mikroupravljač kao i Android mobilna aplikacije. Uz ova dva klijenta generirani su i ostali klijenti pomoću klijentske aplikacije Mqtt-spy. Unutar aplikacije Mqtt-spy generirano je dodatnih sto klijenta, koji su spojeni sa MQTT brokerom. Mikroupravljač sa svojeg senzora čita podatke o temperaturi i vlažnosti zraka i te podatke objavljuje na teme „Temp“ i „Vlaga“. Svi ostali klijenti (Android aplikacija i Mqtt-spy) pretplaćeni su na te teme te primaju podatke sa senzora. Pomoću Android aplikacije izvršeno je mjerenje odaziva.

Broj spojenih klijenta	Mjerenje		QoS 0	QoS 1	QoS2
12	1.	**Prosjek [ms]**	**58.45**	**141.45**	**243.28**
		St. devijacija	0.81	20.03	15.84
	2.	**Prosjek [ms]**	**57.21**	**136.61**	**240.18**
		St. devijacija	2.01	17.7	4.08
	3.	**Prosjek [ms]**	**71.09**	**135.01**	**241.12**
		St. devijacija	24.49	13.14	4.21
52	1.	**Prosjek [ms]**	**58.61**	**134.85**	**244.64**
		St. devijacija	2.40	18.14	4.72
	2.	**Prosjek [ms]**	**61.03**	**139.72**	**266.90**
		St. devijacija	9.47	15.42	65.92
	3.	**Prosjek [ms]**	**75.26**	**143.52**	**246.05**
		St. devijacija	44.88	39.14	12.23
102	1.	**Prosjek [ms]**	**64.97**	**136.89**	**244.29**
		St. devijacija	21.28	15.54	7.99
	2.	**Prosjek [ms]**	**79.78**	**147.91**	**273.91**
		St. devijacija	36.14	32.06	41.10
	3.	**Prosjek [ms]**	**65.07**	**133.36**	**259.91**
		St. devijacija	15.46	18.25	34.18

Tabela 9.2 Rezultati odaziva MQTT brokera s obzirom na broj spojenih klijenata

Prema podacima iz tabele 9.2 izračunat je prosjek svih mjerenja te je na temelju njega dobiven graf kojim je prikazana ovisnost odaziva MQTT brokera o broju spojenih klijenata.



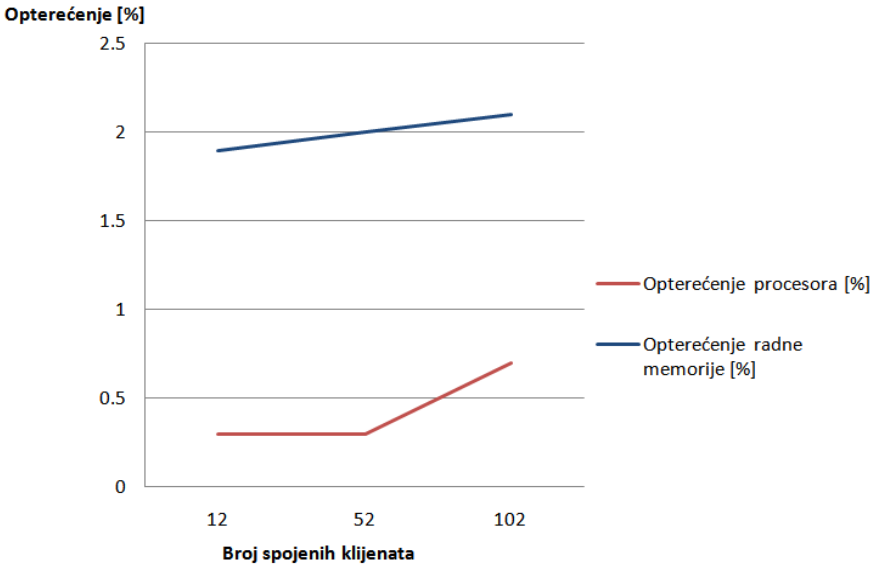
Slika 9.5 Graf ovisnosti odaziva MQTT brokera prema broju spojenih klijenata

Graf prikazan slikom 9.5 prikazuje blagi porast odaziva prema broju spojenih klijenta (aktivnih veza) na svim razinama kvalitete usluge. Konfiguracija virtualnog računala na udaljenom poslužitelju (VPS-u) na kojem su izvršena mjerenja prikazana je u sljedećoj tabeli.

Procesor	Intel Xenon E5-2630L
Radni takt procesora	2.00 GHz
Veličina podatkovne sabirnice procesora	64 bita
Radna memorija	512 MB

Tabela 9.3 Konfiguracija VPS-a na kojem je mjerena skalabilnost MQTT brokera

Uz mjerenje odaziva MQTT brokera također je praćeno opterećenje procesora, kao i radne memorije računala (VPS-a) na kojem je postavljen MQTT broker.



Slika 9.6 Graf potrošnje računalnih resursa MQTT brokera u ovisnosti sa brojem spojenih klijenata

S obzirom da je kod testiranja skalabilnosti korišten Mosquitto MQTT broker, rezultati opterećenja računalnih resursa MQTT brokera prikupljeni su na način praćenja samog Mosquitto računalnog procesa. Prema grafu prikazanog slikom 9.6 vidljivo je povećanje opterećenja procesora kao i radnje memorije s obzirom na broj spojenih klijenata (aktivnih veza). Budući da je mjerenje odaziva izvršeno korištenjem Android mobilne aplikacije sve poruke prema kojima je mjeren odaziv imaju sadržaj jednake duljine (1 bajt).

10. Zaključak

U završnom radu i prilozima detaljno je opisan MQTT protokol. MQTT je protokol aplikacijskog sloja, koji na nižim slojevima koristi TCP/IP protokol. Namijenjen je za komunikaciju između uređaja ograničenih resursa kao što su mikroupravljači. Da bi klijenti međusobno mogli komunicirati potreban je posrednik, odnosno MQTT broker. MQTT protokol koristi objavi/pretplati obrazac, gdje je svaka objavljena poruka vezana uz određenu temu. Klijenti se mogu pretplatiti na teme od njihovog interesa i tako dobivati objavljene poruke. U praktičnom dijelu rada demonstriran je princip rada MQTT protokola na primjeru komunikacije između Android uređaja te ESP 8266-12F mikroupravljača, gdje se pomoću jednostavne Android aplikacije vrši manipulacija ulazno/izlaznih pinova mikroupravljača. Također je demonstriran primjer instalacije MQTT brokera na virtualnom privatnom poslužitelju (VPS-u). Pri tome je stavljen poseban fokus na sigurnost prijenosa podataka, odnosno implementaciju SSL/TLS protokola između MQTT (aplikacijskog) i transportnog sloja. Korištenje kriptirane i sigurne veze od posebnog je značaja u IoT projektima, gdje u osnovi klijent MQTT brokeru pristupa koristeći Internet. Kroz primjere izrađene u praktičnom dijelu rada, pokazalo se da je, za osnovnu primjenu, implementacija MQTT protokola relativno jednostavna na raznim platformama to omogućuju programske knjižnice dostupne za većinu programskih jezika. Testiranjem skalabilnosti MQTT brokera primjećuje se blagi porast vremena odaziva MQTT brokera s obzirom na broj spojenih klijenta. Sa većim brojem spojenih klijenta osjeti se i opterećenje procesora kao i memorije računala na kojem je MQTT broker postavljen. Te činjenice bi valjalo uzeti u obzir prilikom projektiranja sustava baziranog na MQTT protokolu. Analizom MQTT paketa utvrđeno je da klijent prema MQTT brokeru šalje poruku, koju MQTT broker razdjeljuje ostalim pretplaćenim klijentima. To je bitno za resursno ograničene klijente kao što su mikroupravljači jer mrežnu komunikaciju preuzima MQTT broker. Iz svega navedenog može se zaključiti da MQTT predstavlja dobar protokol za komunikaciju između uređaja ograničenih resursa, što je osnova svakog IoT projekta.

U Varaždinu _____ _____

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, DARIO KOŠUTAR *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZVEDBA I TESTIRANJE KOMUNIKACIJSKOG SUSTAVA BAZIRANOG NA MQTT PROTOKOLU *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Dario Košutar

(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, DARIO KOŠUTAR *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZVEDBA I TESTIRANJE KOMUNIKACIJSKOG SUSTAVA BAZIRANOG NA MQTT PROTOKOLU *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Dario Košutar

(vlastoručni potpis)

11. Literatura

[1] Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Negash Subrahmanyam, Rong Xiang: Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, International Technical Support Organization, 2012.

[2] https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt, dostupno 17.02.2018.

[3] https://www.ibm.com/developerworks/library/iot-mqtt-why-good-for-iot/index.html, dostupno 17.02.2018.

[4] https://en.wikipedia.org/wiki/MQTT, dostupno 17.02.2018.

[5] https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment, dostupno 17.02.2018

[6] https://github.com/emqtt/emqttd dostupno 17.02.2018.

[7] https://github.com/mqtt/mqtt.github.io/wiki/servers 17.02.2018.

[8] https://github.com/mqtt/mqtt.github.io/wiki/public_brokers, dostupno 17.02.2018.

[9] https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels, dostupno 17. 02.2018.

[10] https://www.banggood.com/ESP8266-ESP-12F-Remote-Serial-Port-WIFI-Transceiver-Wireless-Module-p-1007260.html?cur_warehouse=CN, dostupno 17.02.2018.

[11] www.elecrow.com/download/ESP-12F.pdf, dostupno 17.02.2018.

[12] https://e-radionica.com/hr/usb-uart-adapter-s-cp2102-made-by-e-radionica.html, dostupno 17.02.2018.

[13] https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi, dostupno 17.02.2018.

[14] https://github.com/adafruit/DHT-sensor-library, dostupno 17.02.2018.

[15] https://github.com/256dpi/arduino-mqtt, dostupno 17.02.2018.

Popis slika

Slika 2.1 Prikaz MQTT protokola u OSI i TCP/IP mrežnom modelu 3
Slika 2.2 Princip rada MQTT protokola 3
Slika 2.3 Uspostava veze između MQTT klijenta i brokera 6
Slika 3.1 Razmjena poruke između MQTT klijenta i brokera sa kvalitetom usluge 0 10
Slika 3.2 Razmjena poruke između MQTT klijenta i brokera sa kvalitetom usluge 1 10
Slika 3.3 Razmjena poruke između MQTT klijenta i brokera sa kvalitetom usluge 2 10
Slika 5.1 Komunikacija između klijenta i brokera korištenjem MQTT paketa 20
Slika 5.2 Primjer komunikacije koristeći najvišu kvalitetu usluge (QoS 2) 21
Slika 6.1 Koncept izvedenog sustava za demonstraciju MQTT protokola 23
Slika 6.2 Odaziv naredbe "sudo service mosquitto status" 24
Slika 6.3 Objavljena poruka „Linux testiranje MQTT klijenti“ 25
Slika 6.4 Izgled mosquitto.conf datoteke nakon potrebnih izmjena 25
Slika 6.5 Slika prikazuje neovlašteni pokušaj pristupa Mosquitto MQTT brokeru 26
Slika 6.6 Prikaz objavljene poruke „Linux testiranje MQTT klijenti sa autentifikacijom“ 26
Slika 6.7 Prikaz snimljenih MQTT paketa koristeći program Wireshark 27
Slika 6.8 Prikaz fiksnog i varijabilnog zaglavlja MQTT paketa, iz čijega se tereta može pročitati korisničko ime i lozinka 28
Slika 6.9 CONNACK paket, vrijednost povratnog koda je 0 što znači da je zahtjev za uspostavu veze prihvaćen 28
Slika 6.10 Prikaz SUBSCRIBE kontrolnog MQTT paketa 28
Slika 6.11 Prikaz PUBLISH MQTT paketa 29
Slika 6.12 Prikaz paketa snimljenih pomoću programa Wireshark 29
Slika 6.13 Izgled Mosquitto konfiguracijske datoteke uz korištenje TLS/SSL protokola 31
Slika 6.14 Prikaz zarobljenih paketa MQTT protokola uz korištenje TLS/SSL protokola 31
Slika 6.15 Prikaz naredbe za periodičku provjeru i obnovu certifikata u Cron tablici 33
Slika 6.16 Zadana konfiguracije Mosquitto brokera 34
Slika 6.17 Izgled sučelja programa Mqtt-spy (tabela sa opcijama za uspostavu veze) 35
Slika 6.18 Izgled sučelja programa Mqtt-spy (prikaz objavljene poruke „Zdravo svijete“ na temu „test“) 35
Slika 6.19 Mrežni napad na komunikaciju između klijenta i MQTT brokera 37
Slika 6.20 Skretanje prometa kroz napadačevo mrežno sučelje 37
Slika 6.21 Prikaz IP i MAC adresa žrtve (MQTT klijenta), kao i usmjeritelja 38
Slika 6.22 Slika prikazuje usmjeritelj kao „ cilj 1“ i žrtvu kao „ cilj 2“ 38

Slika 6.23 Prikaz sadržaja zarobljenog CONNECT MQTT paketa is kojega se može pročitati korisničko ime i lozinka 39
Slika 6.24 *Prikaz paketa koje razmjenjuju klijent i MQTT broker korištenjem TLS/SSL sigurnosnog protokola* 40
Slika 6.25 Prikaz paketa koje razmjenjuju klijent i MQTT broker koristeći TLS/SSL protokol ..40
Slika 7.1 ESP 8266-12F mikroupravljač [10] 42
Slika 7.2 Prikaz blok dijagrama ESP8266EX integriranog sklopa [11] 42
Slika 7.3 Slika prikazuje USB-UART adapter baziran na CP2102 integriranom sklopu.[12] 43
Slika 7.4 Slika prikazuje shemu spoja USB-UART adaptera i regulatora napona. 44
Slika 7.5 Shema spoja ESP8266 mikroupravljača sa LED diodama, DHT 11 senzorom vlage i temperature, te dva tipkala sa funkcijom ponovnog pokretanja i programiranja. 44
Slika 7.6 Tiskana pločica za rad sa ESP 8266-12F mikroupravljačem 45
Slika 8.1 Izgled sučelja osnovnih ekrana Android aplikacije 50
Slika 9.1 Dijagram prosječnog odaziva MQTT brokera 63
Slika 9.2 Snimljeni MQTT paketi prilikom mjerenja odaziva MQTT brokera na lokalnoj mreži sa kvalitetom usluge QoS 0 64
Slika 9.3 Snimljeni MQTT paketi prilikom mjerenja odaziva MQTT brokera na lokalnoj mreži sa kvalitetom usluge QoS 1 65
Slika 9.4 Snimljeni MQTT paketi prilikom mjerenja odaziva MQTT brokera na lokalnoj mreži sa kvalitetom usluge QoS 2 66
Slika 9.5 Graf ovisnosti odaziva MQTT brokera prema broju spojenih klijenata 67
Slika 9.6 Graf potrošnje računalnih resursa MQTT brokera u ovisnosti sa brojem spojenih klijenata 68
Slika 12.1 Elektronički sklop za rad sa ESP 8266-12F mikroupravljačem 84

Popis tabela

Tabela 2.1 Neki od MQTT brokera s kratkim opisom [7] 4
Tabela 2.2 Prikaz „gotovih“ web rješenja MQTT brokera [8] 5
Tabela 2.3 Usporedba MQTT i HTTP protokola [1] 7
Tabela 3.1 Opis kvalitete usluge (QoS) u MQTT protokolu [9] 9
Tabela 4.1 Struktura MQTT paketa [10] 13
Tabela 4.2 Struktura fiksnog zaglavlja MQTT paketa [10] 13
Tabela 4.3 Tipovi MQTT paketa [10] 14
Tabela 4.4 Bitovi zastavica u fiksnom zaglavlju MQTT paketa [10] 14
Tabela 4.5 Kodiranje duljine ostatka paketa [10] 15
Tabela 4.6 Opis vrijednosti povratnog koda [10] 16
Tabela 4.7 Zastavica čiste sesije u varijabilnom zaglavlju MQTT paketa (polje koje nije označeno sa „X“) [10] 17
Tabela 4.8 Zastavica oporuke u varijabilnom zaglavlju MQTT paketa (polje koje nije označeno sa „X“) [10] 17
Tabela 4.9 Zastavica kvalitete usluge oporuke (eng. Will QoS) u varijabilnom zaglavlju MQTT paketa [10] 18
Tabela 4.10 Zastavica zadržavanja oporuke u varijabilnom zaglavlju MQTT paketa [10] 18
Tabela 4.11 Zastavice za korisničko ime i lozinku u varijabilnom zaglavlju MQTT paketa [10] 18
Tabela 7.1 Prikaz sadržaja MQTT poruka pomoću kojih se vrši kontrola mikroupravljača 41
Tabela 7.2 Prikaz pinova ESP 8266-12E mikroupravljača [11] 43
Tabela 9.1 Rezultati mjerenja odaziva MQTT brokera. 62
Tabela 9.2 Rezultati odaziva MQTT brokera s obzirom na broj spojenih klijenata 67
Tabela 9.3 Konfiguracija VPS-a na kojem je mjerena skalabilnost MQTT brokera 68
Tabela 12.1 Fiksno zaglavlje `CONNECT` paketa [10] 76
Tabela 12.2 Varijabilno zaglavlje `CONNECT` paketa [10] 76
Tabela 12.3 Fiksno zaglavlje `CONNACK` paketa [10] 77
Tabela 12.4 Varijabilno zaglavlje `CONNACK` paketa [10] 77
Tabela 12.5 Fiksno zaglavlje `PUBLISH` paketa [10] 77
Tabela 12.6 Primjer podataka u varijabilnom zaglavlju `PUBLISH` paketa [10] 77
Tabela 12.7 Varijabilno zaglavlje PUBLISH paketa prema primjeru iz tabele 12.6 [10] 77
Tabela 12.8 Fiksno zaglavlje `PUBACK` paketa [10] 78
Tabela 12.9 Varijabilno zaglavlje `PUBACK` paketa [10] 78
Tabela 12.10 Fiksno zaglavlje `PUBREC` paketa [10] 78

Tabela 12.11 Varijabilno zaglavlje PUBREC paketa [10] 78
Tabela 12.12 Fiksno zaglavlje PUBREL paketa [10].....79
Tabela 12.13 Varijabilno zaglavlje PUBREL paketa [10] 79
Tabela 12.14 Fiksno zaglavlje PUBCOMP paketa [10].....79
Tabela 12.15 Varijabilno zaglavlje PUBCOMP paketa [10] 79
Tabela 12.16 Fiksno zaglavlje SUBSCRIBE paketa [10] 79
Tabela 12.17 Varijabilno zaglavlje SUBSCRIBE paketa [10].....80
Tabela 12.18 primjer tereta u SUBSCRIBE paketu [10].....80
Tabela 12.19 Varijabilno zaglavlje SUBSCRIBE paketa prema primjeru iz tabele 12.18 [10].....80
Tabela 12.20 Fiksno zaglavlje SUBACK paketa [10].....81
Tabela 12.21 Varijabilno zaglavlje SUBACK paketa [10] 81
Tabela 12.22 Format tereta SUBACK paketa [10].....81
Tabela 12.23 Primjer tereta SUBACK paketa [10] 81
Tabela 12.24 Format tereta SUBACK paketa na primjeru iz tablice 12.23 [10].....81
Tabela 12.25 Fiksno zaglavlje UNSUBSCRIBE paketa [10].....81
Tabela 12.26 Varijabilno zaglavlje UNSUBSCRIBE paketa [10] 82
Tabela 12.27 Primjer tereta UNSUBSCRIBE paketa [10] 82
Tabela 12.28 Teret UNSUBSCRIBE paketa prema tabeli 12.27 [10].....82
Tabela 12.29 Fiksno zaglavlje UNSUBACK paketa [10] 82
Tabela 12.30 Varijabilno zaglavlje UNSUBACK paketa [10] 83
Tabela 12.31Fiksno zaglavlje PINGREQ paketa [10].....83
Tabela 12.32 Fiksno zaglavlje PINGRESP paketa [10] 83
Tabela 12.33 Fiksno zaglavlje DISCONNECT paketa [10].....83

Popis programskih blokova

Programski blok 7.1 Connnect funkcija mikroupravljača 47
Programski blok 7.2 Funkcija povratnog poziva mikroupravljača (messageReceived) 47
Programski blok 7.3 Glavna petlja mikroupravljača 48
Programski blok 8.1 Inicijalizacija MQTT klijenta u Android aplikaciji 50
Programski blok 8.2 Metoda povezivanja Android aplikacije sa MQTT brokerom 51
Programski blok 8.3 Metoda za prekid veze sa MQTT brokerom 51
Programski blok 8.4 Metoda za uključivanje/isključivanje crvene LED diode 52
Programski blok 8.5 Metoda pretplate na teme 52
Programski blok 8.6 Metoda povratnog poziva u Android aplikaciji 52
Programski blok 8.7 Metoda za inicijalizaciju mjerenja odaziva MQTT brokera 53
Programski blok 8.8 AsyncTask metoda onPreExecute 53
Programski blok 8.9 AsyncTask metoda doInBackground 54
Programski blok 8.10 AsyncTask metoda onProgressUpdate 54
Programski blok 8.11 AsyncTask metoda onPostExecute 55
Programski blok 8.12 Postavljanje oporuke na mikroupravljaču 55
Programski blok 8.13 Brisanje zadržane oporuke 56
Programski blok 8.14 Manipulacija sa dolaznim porukama ne temu „Oporuka“ 56
Programski blok 8.15 AsyncTask čekanje da oporuka bude prebrisana 56
Programski blok 9.1 Metoda pretplata i sinkronizacija nakon uspješnog povezivana sa MQTT brokerom (onSuccess metoda) 57
Programski blok 9.2 Metoda pretplate u Android aplikaciji 57
Programski blok 9.3 Prikaz metode „sinkronizacija“ u Android aplikaciji 57
Programski blok 9.4 Manipulacija sa dolaznom porukom sa teme „SINK“ u programu mikroupravljača 58
Programski blok 9.5 Funkcija „povratna veza“ u mikroupravljaču 59
Programski blok 9.6 Ažuriranje statusa LED dioda i tipki za njihovo upravljanje, koristeći dolazne poruke sa tema: SINK_POVRATNO1, SINK_POVRATNO2, SINK_POVRATNO3 60
Programski blok 9.7 Ažuriranje statusa LED dioda i tipki za njihovo upravljanje, koristeći dolazne poruke sa teme „Stanje“ 61

12. Prilozi

12.1. Struktura MQTT paketa (MQTT verzija 3.1.1.)

12.1.1.Struktura fiksnog zaglavlja CONNECT paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1.	Tip poruke (1)				Rezervirano			
	0	0	0	1	0	0	0	0
Bajt 2.	Preostala dužina							

Tabela 12.1 Fiksno zaglavlje CONNECT paketa [10]

12.1.2.Struktura varijabilnog zaglavlja CONNECT paketa

	Opis	7	6	5	4	3	2	1	0
Ime protokola									
Bajt 1	Dužina MSB (0)	0	0	0	0	0	0	0	0
Bajt 2	Dužina LSB (4)	0	0	0	0	0	1	0	0
Bajt 3	‘M’	0	1	0	0	1	1	0	1
Bajt 4	‘Q’	0	1	0	1	0	0	0	1
Bajt 5	‘T’	0	1	0	1	0	1	0	0
Bajt 6	‘T’	0	1	0	1	0	1	0	0
Nivo protokola (verzija)									
	Opis	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bajt 7	Nivo (4)	0	0	0	0	0	1	0	0
Zastavice za povezivanje (CONNECT flags)									
Bajt 8	Zastavica za korisničko ime(1) Zastavica za lozinku (1) Zastavica zadržavanja oporuke (0) Zastavica kvalitete usluge oporuke (01) Zastavica oporuke (1) Zastavica čiste sesije (1) Rezervirano (0)	1	1	0	0	1	1	1	0
„Održi na životu“									
Bajt 9	Održi na životu MSB	0	0	0	0	0	0	0	0
Bajt 10	Održi na životu LSB	0	0	0	0	1	0	1	0

Tabela 12.2 Varijabilno zaglavlje CONNECT paketa [10]

12.1.3.Struktura fiksnog zaglavlja CONNACK paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (2)				Rezervirano			
	0	0	1	0	0	0	0	0
Bajt 2	Preostala dužina (2)							
	0	0	0	0	0	0	1	0

Tabela 12.3 Fiksno zaglavlje CONNACK paketa [10]

12.1.4.Struktura varijabilnog zaglavlja CONNACK paketa

	Opis	7	6	5	4	3	2	1	0
Zastavice potvrde povezivanja (CONNACK zastavica)		Rezervirano							SP1
Bajt 1		0	0	0	0	0	0	0	x
CONNECT povratni kod (*Connect return code*)									
Bajt 2		x	x	x	x	x	x	x	x

Tabela 12.4 Varijabilno zaglavlje CONNACK paketa [10]

12.1.5.Struktura fiksnog zaglavlja PUBLISH paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (3)				DUP zastavica	QoS		RETAIN
	0	0	1	1	x	x	x	x
Bajt 2	Preostala dužina							

Tabela 12.5 Fiksno zaglavlje PUBLISH paketa [10]

12.1.6.Struktura varijabilnog zaglavlja PUBLISH paketa

Polje	Vrijednost
Naziv teme	"a/b"
Identitet poruke	10

Tabela 12.6 Primjer podataka u varijabilnom zaglavlju PUBLISH paketa [10]

	Opis	7	6	5	4	3	2	1	0
Naziv teme									
Bajt 1	Dužina MSB(0)	0	0	0	0	0	0	0	0
Bajt 2	Dužina LSB(3)	0	0	0	0	0	0	1	1
Bajt 3	'a'(0x61)	0	1	1	0	0	0	0	1
Bajt 4	'/'(0x2F)	0	0	1	0	1	1	1	1
Bajt 5	'b'(0x62)	0	1	1	0	0	0	1	0
Identifikator poruke									
Bajt 6	MSB(0)	0	0	0	0	0	0	0	0
Bajt 7	LSB(10)	0	0	0	0	1	0	1	0

Tabela 12.7 Varijabilno zaglavlje PUBLISH paketa prema primjeru iz tabele 12.6 [10]

12.1.7.Struktura fiksnog zaglavlja PUBACK paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (4)				Rezervirano			
	0	1	0	0	0	0	0	0
Bajt 2	Preostala dužina (2)							
	0	0	0	0	0	0	1	0

Tabela 12.8 Fiksno zaglavlje PUBACK paketa [10]

12.1.8.Struktura varijabilnog zaglavlja PUBACK paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Identifikator poruke MSB							
Bajt 2	Identifikator poruke LSB							

Tabela 12.9 Varijabilno zaglavlje PUBACK paketa [10]

12.1.9.Struktura fiksnog zaglavlja PUBREC paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (5)				Rezervirano			
	0	1	0	1	0	0	0	0
Bajt 2	Preostala dužina (2)							
	0	0	0	0	0	0	1	0

Tabela 12.10 Fiksno zaglavlje PUBREC paketa [10]

12.1.10. Struktura varijabilnog zaglavlja PUBREC paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Identifikator poruke MSB							
Bajt 2	Identifikator poruke LSB							

Tabela 12.11 Varijabilno zaglavlje PUBREC paketa [10]

12.1.11. Struktura fiksnog zaglavlja PUBREL paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (6)				Rezervirano			
	0	1	1	0	0	0	1	0
Bajt 2	Preostala dužina (2)							

	0	0	0	0	0	0	1	0

Tabela 12.12 Fiksno zaglavlje PUBREL paketa [10]

12.1.12. Struktura varijabilnog zaglavlja PUBREL paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Identifikator poruke MSB							
Bajt 2	Identifikator poruke LSB							

Tabela 12.13 Varijabilno zaglavlje PUBREL paketa [10]

12.1.13. Struktura fiksnog zaglavlja PUBCOMP paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (7)				Rezervirano			
	0	1	1	1	0	0	1	0
Bajt 2	Preostala dužina (2)							
	0	0	0	0	0	0	1	0

Tabela 12.14 Fiksno zaglavlje PUBCOMP paketa [10]

12.1.14. Struktura varijabilnog zaglavlja PUBCOMP paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Identifikator poruke MSB							
Bajt 2	Identifikator poruke LSB							

Tabela 12.15 Varijabilno zaglavlje PUBCOMP paketa [10]

12.1.15. Struktura fiksnog zaglavlja SUBSCRIBE paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (8)				Rezervirano			
	1	0	0	0	0	0	1	0
Bajt 2	Preostala dužina							

Tabela 12.16 Fiksno zaglavlje SUBSCRIBE paketa [10]

12.1.16. Struktura varijabilnog zaglavlja SUBSCRIBE paketa

	Opis	7	6	5	4	3	2	1	0
Identifikator poruke									
Bajt 1	Identifikator poruke MSB	0	0	0	0	0	0	0	0

Bajt 2	Identifikator poruke LSB	0	0	0	0	0	0	1	0

Tabela 12.17 Varijabilno zaglavlje SUBSCRIBE paketa [10]

12.1.17. Struktura tereta SUBSCRIBE paketa

Naziv teme	**"a/b"**
Zahtijevani nivo usluge	1
Naziv teme	**"c/d"**
Zahtijevani nivo usluge	2

Tabela 12.18 primjer tereta u SUBSCRIBE paketu [10]

	Opis	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Naziv teme									
Bajt 1	Dužina MSB (0)	0	0	0	0	0	0	0	0
Bajt 2	Dužina LSB (3)	0	0	0	0	0	0	1	1
Bajt 3	‘a’(0x61)	0	1	1	0	0	0	0	1
Bajt 4	‘/’(0x2F)	0	0	1	0	1	1	1	1
Bajt 5	‘b’(0x62)	0	1	1	0	0	0	1	0
Zahtijevani nivo usluge									
Bajt 6	Zahtijevani nivo usluge (QoS): 1	0	0	0	0	0	0	0	1
Naziv teme									
Bajt 7	Dužina MSB (0)	0	0	0	0	0	0	0	0
Bajt 8	Dužina LSB (3)	0	0	0	0	0	0	1	1
Bajt 9	‘c’(0x63)	0	1	1	0	0	0	1	1
Bajt 10	‘/’(0x2F)	0	0	1	0	1	1	1	1
Bajt 11	‘d’(0x64)	0	1	1	0	0	1	0	0
Zahtijevani nivo usluge									
Bajt 12	Zahtijevani nivo usluge (QoS): 2	0	0	0	0	0	0	1	0

Tabela 12.19 Varijabilno zaglavlje SUBSCRIBE paketa prema primjeru iz tabele 12.18 [10]

12.1.18. Struktura fiksnog zaglavlja SUBACK paketa

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bajt 1	Tip poruke (9)				Rezervirano			
	1	0	0	1	0	0	0	0

Bajt 2	Preostala dužina

Tabela 12.20 Fiksno zaglavlje SUBACK paketa [10]

12.1.19. Struktura varijabilnog zaglavlja SUBACK paketa

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bajt 1	Identifikator poruke MSB							
Bajt 2	Identifikator poruke LSB							

Tabela 12.21 Varijabilno zaglavlje SUBACK paketa [10]

12.1.20. Struktura tereta SUBACK paketa

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Povratni kod							
Bajt 1	x	0	0	0	0	0	x	x

Tabela 12.22 Format tereta SUBACK paketa [10]

Garantirani nivo kvalitete usluge	0
Garantirani nivo kvalitete usluge	2
Greška slanja	128

Tabela 12.23 Primjer tereta SUBACK paketa [10]

	Opis	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bajt 1	Garantirani nivo kvalitete usluge(0)	0	0	0	0	0	0	0	0
Bajt 2	Garantirani nivo kvalitete usluge(2)	0	0	0	0	0	0	1	0
Bajt3	Greška slanja	1	0	0	0	0	0	0	0

Tabela 12.24 Format tereta SUBACK paketa na primjeru iz tablice 12.23 [10]

12.1.21. Struktura fiksnog zaglavlja UNSUBSCRIBE paketa

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bajt 1	Tip poruke (10)				Rezervirano			
	1	0	1	0	0	0	1	0
Bajt 2	Preostala dužina							

Tabela 12.25 Fiksno zaglavlje UNSUBSCRIBE paketa [10]

12.1.22. Struktura varijabilnog zaglavlja UNSUBSCRIBE paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Identifikator poruke MSB							
Bajt 2	Identifikator poruke LSB							

Tabela 12.26 Varijabilno zaglavlje UNSUBSCRIBE paketa [10]

12.1.23. Struktura tereta UNSUBSCRIBE paketa

Naziv teme	»a/b«
Naziv teme	»c/d«

Tabela 12.27 Primjer tereta UNSUBSCRIBE paketa [10]

	Opis	7	6	5	4	3	2	1	0
Naziv teme									
Bajt 1	Dužina MSB (0)	0	0	0	0	0	0	0	0
Bajt 2	Dužina LSB (3)	0	0	0	0	0	0	1	1
Bajt 3	'a' (0x61)	0	1	1	0	0	0	0	1
Bajt 4	'/' (0x2F)	0	0	1	0	1	1	1	1
Bajt 5	'b' (0x62)	0	1	1	0	0	0	1	0
Naziv teme									
Bajt 6	MSB (0)	0	0	0	0	0	0	0	0
Bajt 7	LSB (3)	0	0	0	0	0	0	1	1
Bajt8	'c' (0x63)	0	1	1	0	0	0	1	1
Bajt 9	'/' (0x2F)	0	0	1	0	1	1	1	1
Bajt 10	'd' (0x64)	0	1	1	0	0	1	0	0

Tabela 12.28 Teret UNSUBSCRIBE paketa prema tabeli 12.27 [10]

12.1.24. Struktura fiksnog zaglavlja UNSUBACK paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (11)				Rezervirano			
	1	0	1	1	0	0	0	0
Bajt 2	Preostala dužina (2)							
	0	0	0	0	0	0	1	0

Tabela 12.29 Fiksno zaglavlje UNSUBACK paketa [10]

12.1.25. Struktura varijabilnog zaglavlja UNSUBACK paketa

Bit	7	6	5	4	3	2	1	0

Bajt 1	Identifikator poruke MSB
Bajt 2	Identifikator poruke LSB

Tabela 12.30 Varijabilno zaglavlje UNSUBACK paketa [10]

12.1.26. Struktura fiksnog zaglavlja PINGREQ paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (12)				Rezervirano			
	1	1	0	0	0	0	0	0
Bajt 2	Preostala dužina (0)							
	0	0	0	0	0	0	0	0

Tabela 12.31Fiksno zaglavlje PINGREQ paketa [10]

12.1.27. Struktura fiksnog zaglavlja PINGRESP paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (13)				Rezervirano			
	1	1	0	1	0	0	0	0
Bajt 2	Preostala dužina (0)							
	0	0	0	0	0	0	0	0

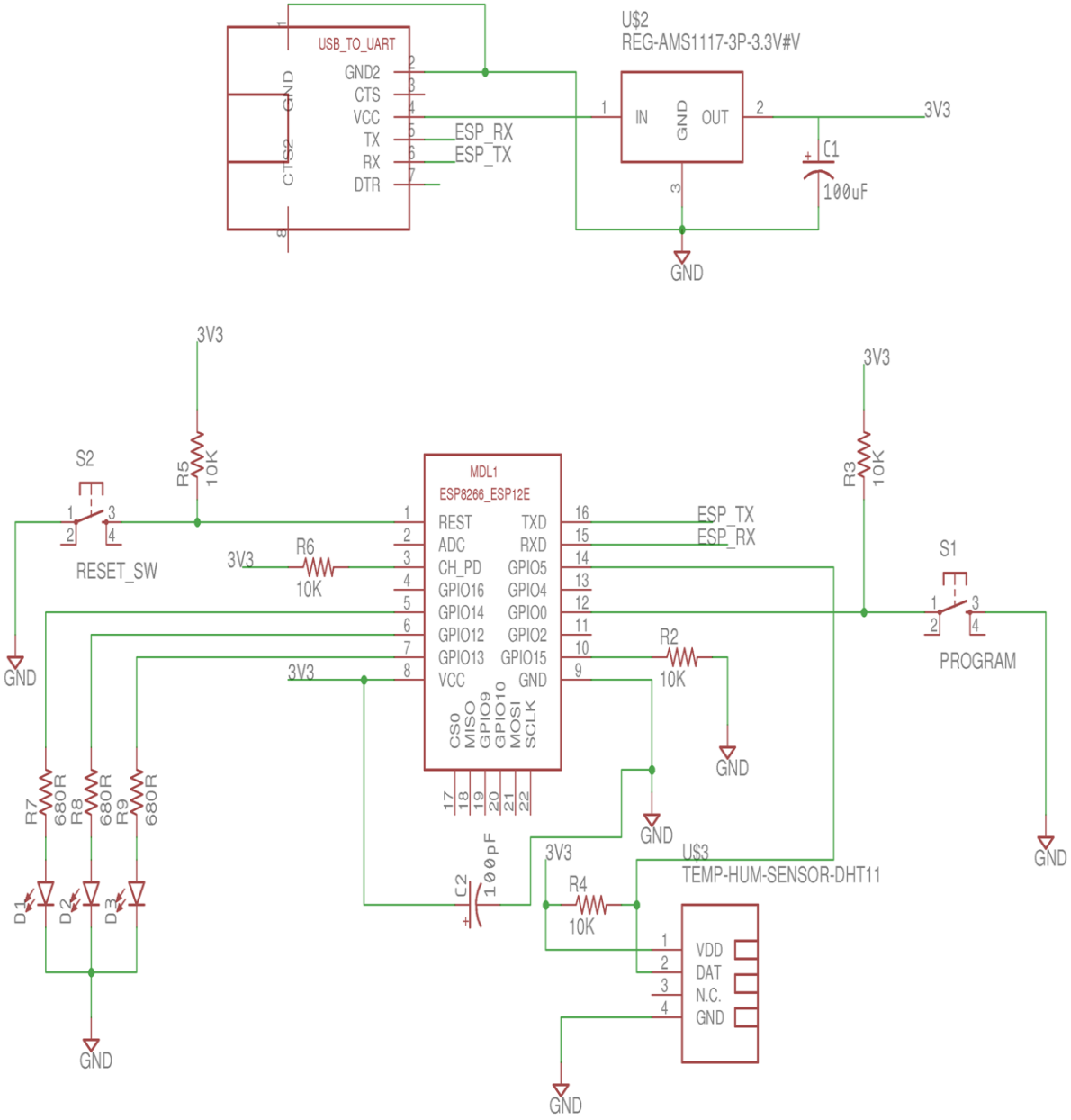
Tabela 12.32 Fiksno zaglavlje PINGRESP paketa [10]

12.1.28. Struktura fiksnog zaglavlja DISCONNECT paketa

Bit	7	6	5	4	3	2	1	0
Bajt 1	Tip poruke (14)				Rezervirano			
	1	1	1	0	0	0	0	0
Bajt 2	Preostala dužina (0)							
	0	0	0	0	0	0	0	0

Tabela 12.33 Fiksno zaglavlje DISCONNECT paketa [10]

12.2. Elektronički sklop za rad sa ESP 8266-12F mikroupravljačem



Slika 12.1 Elektronički sklop za rad sa ESP 8266-12F mikroupravljačem

12.3. Java programski kod za Android aplikaciju

12.3.1. Programski kod Login Activity

```
1.  package com.example.dario.mqttapp_test;
2.  import android.content.Intent;
3.  import android.content.pm.ActivityInfo;
4.  import android.os.AsyncTask;
5.  import android.os.Bundle;
6.  import android.support.v7.app.AppCompatActivity;
7.  import android.view.View;
8.  import android.widget.Button;
9.  import android.widget.CheckBox;
10. import android.widget.EditText;
```

```
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import org.apache.commons.lang3.StringUtils;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;


public class LoginActivity extends AppCompatActivity {
    String URL_prefix="tcp://";   //ako korisnik ne odabere SSL zadano je tcp (bez kriptirane veze)!!
    String Broker_URL;
    String port;
    String kor_ime;
    String lozinka;
    boolean zahtjev_autroizacije=false;
    TextView naslov_kor_ime;
    TextView naslov_lozinka;
    EditText Broker_URL_t;
    EditText port_t;
    EditText kor_ime_t;
    EditText lozinka_t;
    Button spoji_gumb;
    CheckBox SSL;
    CheckBox Autorizacija;
    ProgressBar login_napredak;
    Intent login_intent;
    MqttAndroidClient test_client;
    MqttConnectOptions test_options;


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

        setRequestedOrientation (ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);


        login_intent =new Intent(LoginActivity.this,MainActivity.class);


        SSL = (CheckBox) findViewById(R.id.checkBoxSSL);
        Autorizacija=(CheckBox)findViewById(R.id.Autorizacija_id);
        Broker_URL_t = (EditText) findViewById(R.id.brokerurl);
        port_t = (EditText) findViewById(R.id.port);
        kor_ime_t = (EditText) findViewById(R.id.koris_ime);
        lozinka_t = (EditText) findViewById(R.id.lozinka);
        spoji_gumb = (Button) findViewById(R.id.povzi_gumb);
        naslov_kor_ime=(TextView)findViewById(R.id.textView6);
        naslov_lozinka=(TextView)findViewById(R.id.textView7);
        login_napredak=(ProgressBar)findViewById(R.id.progressBar2);


        SSL.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (SSL.isChecked()) {
                    URL_prefix = "ssl://";
                } else {
                    URL_prefix = "tcp://";
                }
            }
        });
        Autorizacija.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
82.             public void onClick(View v) {
83.                 if (Autorizacija.isChecked()){
84.                     lozinka_t.setVisibility(View.VISIBLE);
85.                     kor_ime_t.setVisibility(View.VISIBLE);
86.                     naslov_kor_ime.setVisibility(View.VISIBLE);
87.                     naslov_lozinka.setVisibility(View.VISIBLE);
88.                     zahtjev_autroizacije=true;
89.                 }else {
90.                     lozinka_t.setVisibility(View.INVISIBLE);
91.                     kor_ime_t.setVisibility(View.INVISIBLE);
92.                     naslov_kor_ime.setVisibility(View.INVISIBLE);
93.                     naslov_lozinka.setVisibility(View.INVISIBLE);
94.                     zahtjev_autroizacije=false;
95.                 }
96.
97.             }
98.         });
99.
100.     }
101.     public void Povezi (View view){
102.
103.         Broker_URL = Broker_URL_t.getText().toString();
104.         port = port_t.getText().toString();
105.         kor_ime = kor_ime_t.getText().toString();
106.         lozinka = lozinka_t.getText().toString();
107.
108.         login_intent.putExtra("ključ_Broker_URL",Broker_URL);
109.         login_intent.putExtra("ključ_port",port);
110.         login_intent.putExtra("ključ_kor_ime",kor_ime);
111.         login_intent.putExtra("ključ_lozinka",lozinka);
112.         login_intent.putExtra("ključ_URL_prefiks",URL_prefix);
113.         login_intent.putExtra("ključ_zahtjev_autorizacije",zahtjev_autroizacije);
114.
115.
116.         if (zahtjev_autroizacije&& (StringUtils.isBlank(kor_ime)||StringUtils.isBlank(lozink
    a))){
117.      Toast.makeText(LoginActivity.this, "Podaci: korisničko ime ili lozinka nisu upisani!!",
            Toast.LENGTH_SHORT).show();
118.         }else{
119.             new Test_u_pozadini().execute();
120.         }
121.
122.
123.     }
124.
125.     public void test_odspoji_funkcija() {
126.         try {
127.                 IMqttToken token = test_client.disconnect();
128.                 token.setActionCallback(new IMqttActionListener() {
129.                     @Override
130.                     public final void onSuccess(IMqttToken asyncActionToken) {
131.                     }
132.                     @Override
133.                     public void onFailure(IMqttToken asyncActionToken, Throwable exception)
    {
134.                     }
135.                 });
136.             } catch (MqttException e) {
137.                 e.printStackTrace();
138.             }
139.     }
140.
141.
142.     public class Test_u_pozadini extends AsyncTask<Void,Integer,Integer>{
143.
144.         @Override
145.         protected void onPreExecute() {
146.             super.onPreExecute();
147.         login_napredak.setMax(3);
148.         login_napredak.setProgress(0);
149.         spoji_gumb.setClickable(false);
150.
```

```
        }

        @Override
        protected void onProgressUpdate(Integer... values) {
            super.onProgressUpdate(values);
            login_napredak.setProgress(values[0]);

        }

        @Override
        protected Integer doInBackground(Void... voids) {

            String clientId = MqttClient.generateClientId();
            test_client = new MqttAndroidClient(getBaseContext(),URL_prefix+Broker_URL+":"+po
rt, clientId);
            test_options = new MqttConnectOptions();
            if (zahtjev_autroizacije) {
                test_options.setUserName(kor_ime);
                test_options.setPassword(lozinka.toCharArray());
            }
            test_options.setCleanSession(false);
            test_options.setAutomaticReconnect(false);


                try {
                    publishProgress(1);
                    IMqttToken token = test_client.connect(test_options);
                    publishProgress(2);
                    token.setActionCallback(new IMqttActionListener() {
                        @Override
                        public void onSuccess(IMqttToken asyncActionToken) {
                            publishProgress(3);
                            startActivity(login_intent);
                            test_odspoji_funkcija();
                            spoji_gumb.setClickable(true);
                        }
                        @Override
                        public void onFailure(IMqttToken asyncActionToken, Throwable exceptio
n) {
                            publishProgress(3);
                            Toast.makeText(LoginActivity.this, "Pogrešni podaci ili greška In
ternet veze", Toast.LENGTH_SHORT).show();
                            spoji_gumb.setClickable(true);
                        }
                    });
                } catch (MqttException e) {
                    e.printStackTrace();
                }
            return null;
        }
    }

 }
```

12.3.2.Programski kod Main Activity

```
package com.example.dario.mqttapp_test;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.os.Vibrator;
import android.support.annotation.RequiresApi;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
```

```
16. import android.widget.Button;
17. import android.widget.ProgressBar;
18. import android.widget.Spinner;
19. import android.widget.TextView;
20. import android.widget.Toast;
21. import android.widget.ToggleButton;
22.
23. import org.apache.commons.math3.util.Precision;
24. import org.eclipse.paho.android.service.MqttAndroidClient;
25. import org.eclipse.paho.client.mqttv3.IMqttActionListener;
26. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
27. import org.eclipse.paho.client.mqttv3.IMqttToken;
28. import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
29. import org.eclipse.paho.client.mqttv3.MqttClient;
30. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
31. import org.eclipse.paho.client.mqttv3.MqttException;
32. import org.eclipse.paho.client.mqttv3.MqttMessage;
33. import java.math.BigDecimal;
34. import java.util.ArrayList;
35. import java.util.List;
36.
37. public class MainActivity extends AppCompatActivity implements AdapterView.OnItemSelectedListe
    ner {
38.     String URL_prefix;
39.     String Broker_URL;
40.     String port;
41.     String kor_ime;
42.     String lozinka;
43.     String topicLED = "LED";
44.     String topicVLAGA = "vlaga";
45.     String topicTEMP = "temp";
46.     String topicODAZIV = "odaziv";
47.     String topicStanje = "Stanje";
48.     String topicSINK = "SINK";
49.     String topicSINK_POV1 = "SINK_POVRATNO1";
50.     String topicSINK_POV2 = "SINK_POVRATNO2";
51.     String topicSINK_POV3 = "SINK_POVRATNO3";
52.     String topicOPORUKA = "ESP_oporuka";
53.     String sadržaj_oporuke;
54.     boolean zahtjev_autorizacije;
55.     int trenutno_stanje_LED;
56.     int SINK_LED_STANJE_crv;
57.     int SINK_LED_STANJE_zel;
58.     int SINK_LED_STANJE_plv;
59.     int sadržaj_odaziva;
60.     int QoS_odaziva;
61.     List<Double> lista_rezultata;
62.     TextView led_crv_stanjetxt;
63.     TextView led_zel_stanjetxt;
64.     TextView led_pla_stanjetxt;
65.     TextView prikazvlaga;
66.     TextView prikaztemp;
67.     TextView prikazodaziv;
68.     TextView prikaz_pros;
69.     TextView prikaz_st_dev;
70.     Button odaziv_gumb;
71.     ToggleButton on_off_crvena;
72.     ToggleButton on_off_zelena;
73.     ToggleButton on_off_plava;
74.     double t_trenutno_poslano;
75.     double djelitelj = 1000000;
76.     MqttAndroidClient client;
77.     MqttConnectOptions options;
78.     Vibrator vibrator;
79.     Ringtone myringtone;
80.     ProgressBar traka_napratka;
81.     Spinner qos_izbor;
82.
83.
84.
85.     @Override
86.     protected void onCreate(Bundle savedInstanceState) {
```

```
87.         super.onCreate(savedInstanceState);
88.         setContentView(R.layout.activity_main);
89.         setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
90.
91.
92.         Intent main_intent = getIntent();
93.         Broker_URL = main_intent.getExtras().getString("ključ_Broker_URL");
94.         port = main_intent.getExtras().getString("ključ_port");
95.         kor_ime = main_intent.getExtras().getString("ključ_kor_ime");
96.         lozinka = main_intent.getExtras().getString("ključ_lozinka");
97.         URL_prefix = main_intent.getExtras().getString("ključ_URL_prefiks");
98.         zahtjev_autorizacije = main_intent.getExtras().getBoolean("ključ_zahtjev_autorizacije"
    );
99.
100.          led_crv_stanjetxt= (TextView) findViewById(R.id.textledcrv);
101.          led_zel_stanjetxt = (TextView) findViewById(R.id.textledzel);
102.          led_pla_stanjetxt = (TextView) findViewById(R.id.textledplav);
103.          lista_rezultata = new ArrayList<>();
104.          prikazvlaga = (TextView) findViewById(R.id.vlagaTxt);
105.          prikaztemp = (TextView) findViewById(R.id.tempText);
106.
107.          prikazodaziv = (TextView) findViewById(R.id.odazivTxt);
108.          prikaz_pros = (TextView) findViewById(R.id.odazivTxt2);
109.          prikaz_st_dev = (TextView) findViewById(R.id.odazivTxt3);
110.
111.          vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
112.          odaziv_gumb = (Button) findViewById(R.id.button3);
113.          odaziv_gumb.setVisibility(View.INVISIBLE);
114.
115.          on_off_crvena=(ToggleButton)findViewById(R.id.toggleButton3);
116.          on_off_zelena = (ToggleButton)findViewById(R.id.toggleButton2);
117.          on_off_plava = (ToggleButton)findViewById(R.id.toggleButton);
118.
119.          Uri uri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
120.          myringtone = RingtoneManager.getRingtone(getApplicationContext(), uri);
121.          traka_napratka = (ProgressBar) findViewById(R.id.progressBar);
122.          if (Broker_URL.equals("mqtt-projekt.tk")){
123.              led_crv_stanjetxt.setVisibility(View.VISIBLE);
124.              led_zel_stanjetxt.setVisibility(View.VISIBLE);
125.              led_pla_stanjetxt.setVisibility(View.VISIBLE);
126.              prikaztemp.setVisibility(View.VISIBLE);
127.              prikazvlaga.setVisibility(View.VISIBLE);
128.              on_off_crvena.setVisibility(View.VISIBLE);
129.              on_off_zelena.setVisibility(View.VISIBLE);
130.              on_off_plava.setVisibility(View.VISIBLE);
131.          } else {
132.              led_crv_stanjetxt.setVisibility(View.INVISIBLE);
133.              led_zel_stanjetxt.setVisibility(View.INVISIBLE);
134.              led_pla_stanjetxt.setVisibility(View.INVISIBLE);
135.              prikaztemp.setVisibility(View.INVISIBLE);
136.              prikazvlaga.setVisibility(View.INVISIBLE);
137.              on_off_crvena.setVisibility(View.INVISIBLE);
138.              on_off_zelena.setVisibility(View.INVISIBLE);
139.              on_off_plava.setVisibility(View.INVISIBLE);
140.          }
141.
142.
143.
144.          qos_izbor = (Spinner) findViewById(R.id.spinner);
145.          ArrayAdapter adapter = ArrayAdapter.createFromResource(this, R.array.kvaliteta_uslug
    e, android.R.layout.simple_spinner_item);
146.          adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
147.          qos_izbor.setAdapter(adapter);
148.          qos_izbor.setOnItemSelectedListener(this);
149.
150.
151.          String clientId = MqttClient.generateClientId();
152.          client = new MqttAndroidClient(this.getApplicationContext(), URL_prefix + Broker_URL
     + ":" + port, clientId);
153.          options = new MqttConnectOptions();
154.          if (zahtjev_autorizacije) {
155.              options.setUserName(kor_ime);
```

```
156.                options.setPassword(lozinka.toCharArray());
157.            }
158.            options.setCleanSession(false);
159.            options.setAutomaticReconnect(true);
160.
161.
162.            try {
163.                IMqttToken token = client.connect(options);
164.                token.setActionCallback(new IMqttActionListener() {
165.                    @Override
166.                    public void onSuccess(IMqttToken asyncActionToken) {
167.                        Toast.makeText(MainActivity.this, "spojeno!!", Toast.LENGTH_LONG).show()
    ;
168.                        pretplata();
169.                        sinkronizacija();
170.                        odaziv_gumb.setVisibility(View.VISIBLE);
171.                    }
172.
173.                    @Override
174.                    public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
175.                        Toast.makeText(MainActivity.this, "greška prilikom povezivanja!!!!", Toa
    st.LENGTH_LONG).show();
176.                    }
177.                });
178.            } catch (MqttException e) {
179.                e.printStackTrace();
180.
181.            }
182.            client.setCallback(new MqttCallbackExtended() {
183.
184.                @Override
185.                public void connectComplete(boolean reconnect, String serverURI) {
186.                }
187.                @Override
188.                public void connectionLost(Throwable cause) {
189.                }
190.
191.                @Override
192.                public void messageArrived(String topic, MqttMessage message) throws Exception {

193.                    if (topic.equals(topicTEMP)) {
194.                        prikaztemp.setText("TEMPERATURA: " + new String(message.getPayload()) +
    "°C");
195.                    }
196.                    if (topic.equals(topicVLAGA)) {
197.                        prikazvlaga.setText("VLAŽNOST ZRAKA: " + new String(message.getPayload()
    ) + "%");
198.                    }
199.                }
200.
201.                    if (topic.equals(topicODAZIV)) {
202.                        sadržaj_odaziva = Integer.parseInt(new String(message.getPayload(), "UTF
    8"));
203.
204.
205.                    }
206.
207.
208.                    if (topic.equals(topicSINK_POV1)) {
209.                        String poruka_sink_pov1 = new String(message.getPayload(), "UTF8");
210.                        ToggleButton tipka_crvena = (ToggleButton) findViewById(R.id.toggleButto
    n3);
211.                        SINK_LED_STANJE_crv = Integer.parseInt(poruka_sink_pov1);
212.                        switch (SINK_LED_STANJE_crv) {
213.                            case 1:
214.                                led_crv_stanjetxt.setText("Crvena ON");
215.                                tipka_crvena.setChecked(true);
216.                                break;
217.                            case 0:
218.                                led_crv_stanjetxt.setText("Crvena OFF");
219.                                tipka_crvena.setChecked(false);
220.                                break;
```

```
221.                     }
222.                 } else if (topic.equals(topicSINK_POV2)) {
223.                     String poruka_sink_pov2 = new String(message.getPayload(), "UTF8");
224.                     ToggleButton tipka_zelena = (ToggleButton) findViewById(R.id.toggleButto
    n);
225.                     SINK_LED_STANJE_zel = Integer.parseInt(poruka_sink_pov2);
226.                     switch (SINK_LED_STANJE_zel) {
227.                         case 1:
228.                             led_zel_stanjetxt.setText("Zelena ON");
229.                             tipka_zelena.setChecked(true);
230.
231.                             break;
232.                         case 0:
233.                             led_zel_stanjetxt.setText("Zelena OFF");
234.                             tipka_zelena.setChecked(false);
235.                             break;
236.                     }
237.                 } else if (topic.equals(topicSINK_POV3)) {
238.                     String poruka_sink_pov3 = new String(message.getPayload(), "UTF8");
239.                     ToggleButton tipka_plava = (ToggleButton) findViewById(R.id.toggleButton
    2);
240.                     SINK_LED_STANJE_plv = Integer.parseInt(poruka_sink_pov3);
241.                     switch (SINK_LED_STANJE_plv) {
242.                         case 1:
243.                             led_pla_stanjetxt.setText("Plava ON");
244.                             tipka_plava.setChecked(true);
245.                             break;
246.                         case 0:
247.                             led_pla_stanjetxt.setText("Plava OFF");
248.                             tipka_plava.setChecked(false);
249.                     }
250.                 }
251.
252.                 if (topic.equals(topicStanje)) {
253.                     String poruka_stanja = new String((message.getPayload()), "UTF8");
254.                     ToggleButton tipka_crvena = (ToggleButton) findViewById(R.id.toggleButto
    n3);
255.                     ToggleButton tipka_zelena = (ToggleButton) findViewById(R.id.toggleButto
    n);
256.                     ToggleButton tipka_plava = (ToggleButton) findViewById(R.id.toggleButton
    2);
257.                     trenutno_stanje_LED = Integer.parseInt(poruka_stanja);
258.                     switch (trenutno_stanje_LED) {
259.                         case 41:
260.                             led_crv_stanjetxt.setText("Crvena ON");
261.                             tipka_crvena.setChecked(true);
262.                             break;
263.                         case 40:
264.                             tipka_crvena.setChecked(false);
265.                             led_crv_stanjetxt.setText("Crvena OFF");
266.                             break;
267.                         case 51:
268.                             led_zel_stanjetxt.setText("Zelena ON");
269.                             tipka_zelena.setChecked(true);
270.                             break;
271.                         case 50:
272.                             led_zel_stanjetxt.setText("Zelena OFF");
273.                             tipka_zelena.setChecked(false);
274.                             break;
275.                         case 61:
276.                             led_pla_stanjetxt.setText("Plava ON");
277.                             tipka_plava.setChecked(true);
278.                             break;
279.
280.                         case 60:
281.                             led_pla_stanjetxt.setText("Plava OFF");
282.                             tipka_plava.setChecked(false);
283.                             break;
284.                     }
285.                 }
286.
287.                 if (topic.equals(topicOPORUKA)) {
```

```
288.                     sadržaj_oporuke = new String(message.getPayload(), "UTF8");
289.                     if (sadržaj_oporuke.equals("RIP")) {
290.                         new čekaj().execute();
291.                     }
292.                 }
293.             }
294.
295.             @Override
296.             public void deliveryComplete(IMqttDeliveryToken token) {
297.
298.             }
299.         });
300.     }
301.     @Override
302.     public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
303.         switch (position) {
304.             case 0:
305.                 QoS_odaziva = 0;
306.                 break;
307.             case 1:
308.                 QoS_odaziva = 1;
309.                 break;
310.             case 2:
311.                 QoS_odaziva = 2;
312.                 break;
313.         }
314.     }
315.
316.     @Override
317.     public void onNothingSelected(AdapterView<?> parent) {
318.         QoS_odaziva = 0;
319.     }
320.
321.     public void pali_gasi_crvenu(View view) {
322.         Boolean stanje_on = ((ToggleButton) view).isChecked();
323.         if (stanje_on) {
324.             String topic = topicLED;
325.             String message = "11";
326.             try {
327.                 client.publish(topic, message.getBytes(), 0, false);
328.                 vibrator.vibrate(150);
329.             } catch (MqttException e) {
330.                 e.printStackTrace();
331.             }
332.         } else {
333.             String topic = topicLED;
334.             String message = "10";
335.             try {
336.                 client.publish(topic, message.getBytes(), 0, false);
337.                 vibrator.vibrate(150);
338.             } catch (MqttException e) {
339.                 e.printStackTrace();
340.             }
341.         }
342.     }
343.
344.     public void pali_gasi_zelenu(View view) {
345.         boolean stanje_on = ((ToggleButton) view).isChecked();
346.         if (stanje_on) {
347.             String topic = topicLED;
348.             String message = "21";
349.             try {
350.                 client.publish(topic, message.getBytes(), 0, false);
351.                 vibrator.vibrate(150);
352.             } catch (MqttException e) {
353.                 e.printStackTrace();
354.             }
355.
356.         } else {
357.             String topic = topicLED;
358.             String message = "20";
359.             try {
```

```
                client.publish(topic, message.getBytes(), 0, false);
                vibrator.vibrate(150);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        }
    }

    public void pali_gasi_plavu(View view) {
        boolean stanje_on = ((ToggleButton) view).isChecked();
        if (stanje_on) {
            String topic = topicLED;
            String message = "31";
            try {

                client.publish(topic, message.getBytes(), 0, false);
                vibrator.vibrate(150);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        } else {
            String topic = topicLED;
            String message = "30";
            try {

                client.publish(topic, message.getBytes(), 0, false);
                vibrator.vibrate(150);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        }
    }

    public void odaziv_funkcija(View view) {
        try {
            client.subscribe(topicODAZIV, QoS_odaziva);
        } catch (MqttException e) {
            e.printStackTrace();
        }
        new Radi_u_pozadini().execute();
    }


    public void sinkronizacija() {
        String topic = topicSINK;
        String message = ("101");
        try {
            client.publish(topic, message.getBytes(), 0, false);
        } catch (MqttException b) {
            b.printStackTrace();
        }
    }

    public void pretplata() {
        try {
            client.subscribe(topicStanje, 0);
            client.subscribe(topicTEMP, 0);
            client.subscribe(topicVLAGA, 0);
            client.subscribe(topicSINK_POV1, 0);
            client.subscribe(topicSINK_POV2, 0);
            client.subscribe(topicSINK_POV3, 0);
            client.subscribe(topicOPORUKA, 0);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onBackPressed() {
        try {
            IMqttToken token = client.disconnect();
            token.setActionCallback(new IMqttActionListener() {
```

```
                    @Override
                    public final void onSuccess(IMqttToken asyncActionToken) {
                        Toast.makeText(MainActivity.this, "odspojeno!!", Toast.LENGTH_SHORT).sho
    w();

                    }

                    @Override
                    public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                        Toast.makeText(MainActivity.this, "nije moguče odspojiti!!!!", Toast.LEN
    GTH_SHORT).show();
                    }
                });
            } catch (MqttException e) {
                e.printStackTrace();
            }
            this.finish();
        }

        class Radi_u_pozadini extends AsyncTask<Void, Integer, Boolean> {
            @Override
            protected void onPreExecute() {
                super.onPreExecute();
                lista_rezultata.clear();
                sadržaj_odaziva = 100;
                traka_napratka.setMax(9);
            }
            @Override
            protected void onProgressUpdate(Integer... values) {
                super.onProgressUpdate(values);
                traka_napratka.setProgress(values[0]);
                odaziv_gumb.setClickable(false);
            }
            @Override
            protected Boolean doInBackground(Void... voids) {
                for (int i = 0; i < 10; i++) {
                    try {
                        Thread.sleep(500);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    String topic = topicODAZIV;
                    String message = Integer.toString(i);

                    try {
                        client.publish(topic, message.getBytes(), QoS_odaziva, false);
                    } catch (MqttException e) {
                        e.printStackTrace();
                    }
                    t_trenutno_poslano = System.nanoTime() / djelitelj;
                    while (i != sadržaj_odaziva) {
                    }
                    double odaziv = (System.nanoTime() / djelitelj) - t_trenutno_poslano;
                    lista_rezultata.add(i, odaziv);
                    publishProgress(i);
                }
                return null;
            }
            @RequiresApi(api = Build.VERSION_CODES.N)
            @Override
            protected void onPostExecute(Boolean aBoolean) {
                super.onPostExecute(aBoolean);

                List<Double> lista_rez_zaokr =new ArrayList<>();
                for (int j=0;j<lista_rezultata.size();j++){
                    double zaokr_vrj =Precision.round(lista_rezultata.get(j),2,BigDecimal.ROUND_
    HALF_UP);
                    lista_rez_zaokr.add(j,zaokr_vrj);
                }
                double sum = 0;
                for (int k = 0; k < lista_rez_zaokr.size(); k++) {
                    sum += lista_rez_zaokr.get(k);
```

```
501.             }
502.             double prosjek = sum / lista_rez_zaokr.size();
503.             List<Double> lista_razlike = new ArrayList<>();
504.             double SD_razlika;
505.             for (int x = 0; x < lista_rez_zaokr.size(); x++) {
506.                 SD_razlika = lista_rez_zaokr.get(x) - prosjek;
507.                 lista_razlike.add(x, SD_razlika);
508.             }
509.             List<Double> lista_kvadrata = new ArrayList<>();
510.             double kvadrati;
511.             for (int z = 0; z < lista_razlike.size(); z++) {
512.                 kvadrati = lista_razlike.get(z) * lista_razlike.get(z);
513.                 lista_kvadrata.add(kvadrati);
514.             }
515.             double zbroj_kvadrata = 0;
516.             for (int y = 0; y < lista_kvadrata.size(); y++) {
517.                 zbroj_kvadrata += lista_kvadrata.get(y);
518.             }
519.             double rez = zbroj_kvadrata / (lista_rez_zaokr.size() - 1);
520.             double st_devijacija = Math.sqrt(rez);
521.             double pros_zaokr = Precision.round(prosjek,2,BigDecimal.ROUND_HALF_UP);
522.             double st_dev_zakr = Precision.round(st_devijacija,2,BigDecimal.ROUND_HALF_UP);

523.             prikazodaziv.setText("REZULTATI ODAZIVA [ms]: "+"\n"+lista_rez_zaokr);
524.             prikaz_pros.setText("PROSJEK ODAZIVA: "+pros_zaokr+" ms");
525.             prikaz_st_dev.setText("STANDARDNA DEVIJACIJA: "+st_dev_zakr);
526.             odaziv_gumb.setClickable(true);
527.         }
528.     }
529.     class čekaj extends AsyncTask<Void,Integer,Integer>{
530.         @Override
531.         protected Integer doInBackground(Void... voids) {
532.             try {
533.                 Thread.sleep(5000);
534.             } catch (InterruptedException e) {
535.                 e.printStackTrace();
536.             }
537.             return null;
538.         }
539.         @Override
540.         protected void onPostExecute(Integer integer) {
541.             super.onPostExecute(integer);
542.             if (sadržaj_oporuke.equals("RIP")) {
543.                 Toast.makeText(MainActivity.this, "ESP GREŠKA !!", Toast.LENGTH_LONG).show()
    ;
544.             }
545.         }
546.     }
547. }
```

12.4. C programski kod korišten za ESP 8266-12F mikroupravljač

```
1.  #include <ESP8266WiFi.h>
2.  #include <MQTTClient.h>
3.  #include "DHT.h"
4.  #define DHTPIN 5
5.  #define DHTTYPE DHT11
6.
7.
8.  const char ssid[] = "xxx";
9.  const char pass[] = "xxxx";
10.
11.
12.
13. const char broker_host[] = "mqtt-projekt.tk";
14. int broker_port = 8883;
15. const char broker_korisnickoime[] = "mqttuser";
16. const char broker_lozinka[] = "mqtttest";
17. char tema_oporuka[] = "ESP_oporuka";
18. char poruka_oporuka[] = "RIP";
```

```
19. char topic_c[50];
20. String vlaga_s;
21. String temp_s;
22. char temp_c[20];
23. char vlaga_c[20];
24. 
25. int Led_crv = 14;
26. int Led_zel = 12;
27. int Led_plv = 13;
28. int stanje_crv;
29. 
30. boolean sinkronizacija;
31. boolean sinkronizacija_pos;
32. 
33. char stanje_crv_ch[20];
34. String stanje_crv_s;
35. String stanje_zel_s;
36. String stanje_plv_s;
37. int stanje_zel;
38. char stanje_zel_ch[20];
39. int stanje_plv;
40. char stanje_plv_ch[20];
41. int stanje_crv_pos;
42. int stanje_zel_pos;
43. int stanje_plv_pos;
44. 
45. //DHT
46. DHT dht(DHTPIN, DHTTYPE);
47. 
48. WiFiClientSecure net;
49. MQTTClient client(512);
50. 
51. unsigned long lastMillis = 0;
52. 
53. void connect();
54. 
55. void setup() {
56.   pinMode(Led_crv, OUTPUT);
57.   pinMode(Led_zel, OUTPUT);
58.   pinMode(Led_plv, OUTPUT);
59. 
60. 
61.   Serial.begin(115200);
62.   WiFi.mode(WIFI_STA);
63.   WiFi.begin(ssid, pass);
64.   dht.begin();
65. 
66. 
67.   client.begin(broker_host, broker_port, net);
68.   client.onMessage(messageReceived);
69.   client.setWill(tema_oporuka,poruka_oporuka,1,2);
70.   connect();
71.   }
72. 
73. void connect() {
74.   Serial.print("\nPovezivanje na Wi-Fi AP:");
75.   while (WiFi.status() != WL_CONNECTED) {
76.     Serial.print(".");
77.     delay(1000);
78.   }
79.   Serial.print("\nPovezano sa pristupnom točkom: ");
80.   Serial.print(ssid);
81.   Serial.print("\nIP Adresa ESP8266 mikro-upravljaca: ");
82.   Serial.print(WiFi.localIP());
83.   Serial.print("\nPovezivanje sa MQTT brokerom...");
84.   while (!client.connect("ESP8266-12F", broker_korisnickoime, broker_lozinka)) {
85.     Serial.print(".");
86.     delay(1000);
```

```
  }
  Serial.print("\nPovezano sa MQTT brokerom!:");
  Serial.print(broker_host);
  client.publish(tema_oporuka,"",1,0);
  delay (10);
  client.subscribe("SINK");
  client.subscribe("LED");
  client.publish("SINK","101");
  delay (10);
  }

void loop() {
  client.loop();
   delay(10);
   if (!client.connected()) {
     connect();
   }
   stanje_crv = digitalRead (Led_crv);
   stanje_zel = digitalRead (Led_zel);
   stanje_plv = digitalRead (Led_plv);
   povratna_veza();
   if (millis() - lastMillis > 3000) {
     lastMillis = millis();
     float fvlaga = dht.readHumidity();
     float ftemp = dht.readTemperature();
     temp_s = String(ftemp);
     temp_s.toCharArray(temp_c, temp_s.length() + 1);
     vlaga_s = String(fvlaga);
     vlaga_s.toCharArray(vlaga_c, vlaga_s.length() + 1);
     client.loop();
     client.publish("temp", temp_c );
     Serial.print("\nobjavljujem temperaturu:" + temp_s + "°C");
     client.publish("vlaga", vlaga_c);
     Serial.print("\nobjavljujem vlažnost zraka:" + vlaga_s + "%");
   }
}

void messageReceived(String &topic, String &payload) {
  topic.toCharArray(topic_c, topic.length() + 1);
  if (strcmp (topic_c, "LED") == 0) {
    int i = payload.toInt();
    switch (i) {
      case 10:
        digitalWrite(Led_crv, LOW);
        Serial.print("\nLED crvena OFF");
        break;
      case 11:
        digitalWrite(Led_crv, HIGH);
        Serial.print("\nLED crvena ON");
        break;
      case 20:
        digitalWrite(Led_zel, LOW);
        Serial.print("\nLED zelena OFF");
        break;
      case 21:
        digitalWrite(Led_zel, HIGH);
        Serial.print("\nLED zelena ON");
        break;
      case 30:
        digitalWrite(Led_plv, LOW);
        Serial.print("\nLED plava OFF");
        break;
      case 31:
        digitalWrite(Led_plv, HIGH);
        Serial.print("\nLED plava ON");
        break;
    }
  }
```

```
  if (strcmp (topic_c,"SINK" )== 0){
    int i = payload.toInt();
    switch (i){
      case 101:
      sinkronizacija=true;
      sinkronizacija_pos=false;
      }
   }
}


void setWill(const char topic[], const char payload[],bool retained,int qos) {

}

void povratna_veza() {
  if (stanje_crv != stanje_crv_pos) {
    if (stanje_crv == HIGH) {
      client.publish("Stanje", "41");
    } else {
      client.publish("Stanje", "40");
    }
    delay (50);
    stanje_crv_pos = stanje_crv;
  }
  if (stanje_zel != stanje_zel_pos) {
    if (stanje_zel == HIGH) {
      client.publish("Stanje", "51");
    } else {
      client.publish("Stanje", "50");
    }
    delay (50);
    stanje_zel_pos = stanje_zel;
  }
  if (stanje_plv != stanje_plv_pos) {
    if (stanje_plv == HIGH) {
      client.publish("Stanje", "61");
    } else {
      client.publish("Stanje", "60");
    }
    delay (50);
    stanje_plv_pos = stanje_plv;
  }
  if (sinkronizacija!=sinkronizacija_pos){
   stanje_crv_s= String (stanje_crv);
   stanje_zel_s= String (stanje_zel);
   stanje_plv_s= String (stanje_plv);
   stanje_crv_s.toCharArray(stanje_crv_ch,stanje_crv_s.length()+1);
   stanje_zel_s.toCharArray(stanje_zel_ch,stanje_zel_s.length()+1);
   stanje_plv_s.toCharArray(stanje_plv_ch,stanje_plv_s.length()+1);
   client.publish("SINK_POVRATNO1",stanje_crv_ch);
   delay (50);
   client.publish("SINK_POVRATNO2",stanje_zel_ch);
   delay (50);
   client.publish("SINK_POVRATNO3",stanje_plv_ch);
   delay (50);
   sinkronizacija_pos=sinkronizacija;
   }
}
```