Tampereen yliopisto

Janne Siltanen

MQTT-PROTOKOLLAN SOVELTUVUUS IOT-JÄRJESTELMIIN

Tekniikan ja luonnontieteiden tiedekunta
Kandidaatintyö
Marraskuu 2019

TIIVISTELMÄ

Janne Siltanen: MQTT-protokollan soveltuvuus IoT-järjestelmiin
Kandidaatintyö
Tampereen yliopisto
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Marraskuu 2019

Tässä työssä tutkitaan MQTT-tiedonsiirtoprotokollan soveltuvuutta IoT-järjestelmiin. Työ koostuu teoriaosuudesta sekä käytännön työstä. Teoriaosuudessa käsitellään protokollalle ominaisia käsitteitä kuten julkaisija/tilaaja-periaatetta, aihekäsitteitä, viestityyppejä, salausta sekä välittäjiä. Teoriaosuuteen sisältyy myös esimerkkejä, jotka kuvaavat protokollan toimintaa reaalimaailmassa.

Työn käytännön osuudessa luodaan yksinkertainen kodin automaatiojärjestelmän seurantasovellus käyttäen tiedonsiirtoon MQTT-protokollaa. Sovellukseen luodaan kaksi asiakasohjelmaa eri ohjelmointikielillä ja siitä tehdään salattu käyttäen TSL/SSL-salausta. Toinen ohjelmista toimii järjestelmän graafisena käyttöliittymänä ja toinen anturidatan tuottajana. Sovelluksesta tehdään myös stabiili, joten sitä pystytään käyttämään myös epävakailla Internet-yhteyksillä.

Avainsanat: Esineiden Internet, IoT, MQTT, TSL/SSL, Tiedonsiirto, Käyttöliittymä

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

GPIO	General Purpose Input/Ouput, yleiskäyttöinen pinnirima pientietokoneissa
IoT	Internet of Things, esineiden Internet
IP-osoite	Internet Protocol-osoite, numerosarja, jolla yksilöidään verkossa olevat laitteet
MQTT	Message Queuing Telemetry Transport, tiedonsiirtoprotokolla
M2M	Machine to Machine, koneiden välinen kommunikointi
Python3	ohjelmointikieli Pythonin versio 3
QoS	engl. Quality of Service, palvelun laatutaso, MQTT-viestien lähetyspriorisointi
Qt	Alustariippumaton graafisten käyttöliittymien käyttöliittymäkirjasto ja ohjelmointiympäristö
RP	Raspberry Pi, kaupallinen pienoistietokone
SCADA	Supervisory Control and Data Acquisition, valvomo-ohjelmisto
SSH	Secure Shell, salattuun etäyhteyteen käytettävä tietoliikenneprotokolla
TCP/IP	Transmission Control Protocol/Internet Protocol, tiedonsiirtoprotokolla
TSL/SSL	Transport Secure Layer/Secure Sockets Layer, salausprotokolla
UTF-8	Unicode Transformation Format, tietokonejärjestelmiä varten kehitetty merkistöstandardi
WiFi	Langaton tietoliikenneteknologia

1. JOHDANTO

Tämän päivän automaatio ei ylety pelkästään teollisuuteen vaan autot, kodit, rakennukset ja useat palvelut ovat laajasti automatisoituja. Internet-verkkojen laajuus ja toimintavarmuus ovat edesauttaneet automaation kehitystä. On harvinaista löytää automatisoitu järjestelmä, joka ei olisi lainkaan kytketty Internet-verkkoon. Useat kodin laitteet kykenevät yhdistämään verkkoon ilman minkäänlaisia lisäkomponentteja ja tämän takia Internet sekä automaatio on osana monen ihmisen elämää. Niin sanottu esineiden Internet (engl. IoT, Internet of Things) on luonut laajat puitteet ihmisille luoda ja käyttää sovelluksia vapaasti ja helposti, mutta samalla luonut myös kysymyksiä tietoturvasta ja tietosuojasta. Laitteet keräävät suuret määrät dataa, joka luo kysymyksiä yksityisyydestä ja turvallisuudesta. [1]

Tässä työssä tutkitaan MQTT-protokollan soveltuvuutta esineiden Internet -järjestelmiin. MQTT on yksi yleisimmistä esineiden Internet -tiedonsiirtoprotokollista. Työ on jaettu kahteen osaan, joista ensimmäisessä tutkitaan ja käsitellään protokollaa ja sen ominaisuuksia teorian kannalta ja toisessa tehdään soveltuvuusselvitys (engl. PoC, Proof of Concept), jossa todistetaan sen toiminta käytännön sovelluksen kautta.

Ensimmäisessä osassa tarkoituksena on esitellä tiedonsiirtoprotokollille ominainen tilaaja/julkaisija-periaate sekä siihen liittyvät aiheet, käsitteet, viestit sekä tietoturvaseikat. Tarkoituksena on käsitellä aiheet perusteiden ja toimintamallien kautta paneutumatta liikaa tiedonsiirtoprotokollien pakettiarkkitehtuuriin. Käytännössä tämä tarkoittaa, että työssä käsiteltävät aiheet ovat ymmärrettävissä ilman syvää tietämystä tietoliikennetekniikasta ja protokollista.

Soveltuvuusselvityksessä otetaan käyttöön protokollaa hyödyntävä lämpötilan ja kosteuden seurantasovellus. Sovellus toteutetaan Raspberry Pi -ympäristössä. Työssä esitellään komponenttien toimintaperiaate, tietoliikenneympäristön pystytys sekä konfigurointi. Asiakasohjelmia varten kirjoitettavien koodien toiminta esitellään pääpiirteittäin. Käytännön työstä on tavoitteena luoda täysin suojattu käyttämällä TSL/SSL-salausta.

2. MQTT-PROTOKOLLA

MQTT eli Message Queuing Telemetry Transport on yleisesti käytetty tiedonsiirtoprotokolla laitteiden välisessä kommunikoinnissa. Se on hyvin suosittu IoT-sovelluksissa sen laajan ohjelmointikielituen, avoimen lähdekoodin sekä yksinkertaisuuden ansiosta. Sen kevyt toiminta mahdollistaa vakaan toiminnan myös hitailla ja epävakailla Internet-yhteyksillä. MQTT:lla voidaan luoda suuria järjestelmiä sen suuren skaalautuvuuden vuoksi. [2, Luku 1]

MQTT on hyvin suosittu niin sanotuissa tee-se-itse-projekteissa, joissa luodaan yksinkertaisia kodin automaatiosovelluksia esineiden Internetin avulla. Verkosta löytyy miltei loputon määrä ohjeita pienten sovellusten tekemiseen tämän protokollan avulla. MQTT:ta käytetään myös esimerkiksi Facebookin pikaviestimessä, joten se on suosittu protokolla myös automaation ulkopuolella. [3]

Tässä luvussa käsitellään protokollan toimintaa, käsitteitä sekä sen syntyperää. Alaluvuissa 2.2 ja 2.3 käsitellään tiedonsiirtoprotokollille ominainen tilaaja/julkaisija-periaate sekä MQTT:n aiheet ja viestityypit. Alaluvussa 2.6 keskitytään protokollalle tärkeisiin ominaisuuksiin eli viimeiseen tahtoon ja viestin laatutasoon.

2.1 Protokollan historia

Andy Stanford-Clarkin (IBM) ja Arlen Nipperin (Cirrus Link) kehittivät MQTT-protokollon vuonna 1999. Idea syntyi tarpeesta kehittää sopiva tiedonsiirtomenetelmä energiayhtiö ConocoPhillipsin öljyputkiston SCADA-järjestelmille. 2000-luvun lopun aikaan Internet-yhteyksien laajuus sekä tiedonsiirron hinta loivat rajoitteet ja vaatimukset uuden protokollan kehitykselle. Protokollalle luotiin viisi tavoitetta: sen on oltava yksinkertainen, sen tulee sisältää pakettien priorisointi, sen pitää olla kevyt ja käyttää mahdollisimman vähän kaistaa, sen pitää olla riippumaton datan tyypistä sekä jatkuvasti tietoinen kyseisen session tilasta. [4]

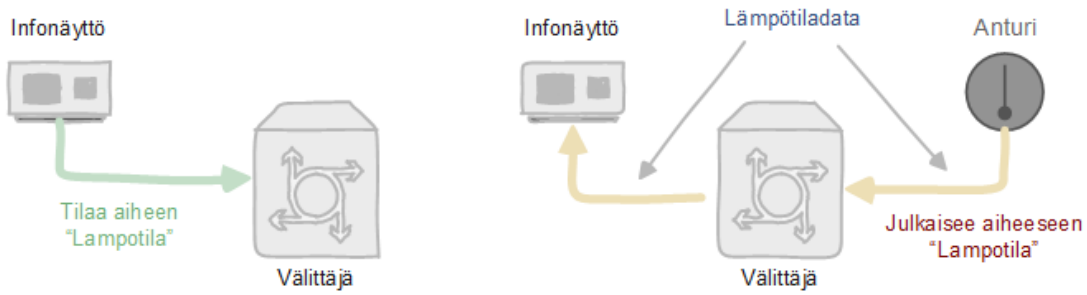
Protokollaa aloitettiin rakentamaan IBM:n sen aikaisen MQ-väliohjelmistotuoteperheen päälle, mistä protokollan nimikin on peräisin. MQTT:sta luotiin protokolla, jonka välittäjä näkyy verkossa omana laitteenaan ja joka pystyy keskustelemaan muiden laitteiden kanssa TCP/IP-kielelllä.

IBM käytti protokollaa vain yrityksen omissa projekteissa vuoteen 2010 asti, kunnes vapautti 3.1-version tekijänoikeusvapaaseen käyttöön. Tästä lähtien protokollan tuki ja suosio on kasvanut huomattavasti. Nykyään MQTT:n tukea ylläpitää Eclipse-säätiön Paho-

projekti, jonka sivuilta löytyy dokumentaatio sekä lähdekoodi useille eri ohjelmointikielille. [2, Luku 1]

2.2 Julkaisija/tilaaja-periaate

MQTT:lle sekä monille muille tiedonsiirtoprotokollille on ominaista niin sanottu julkaisija/tilaaja-periaate. Tällä periaatteella voidaan korvata perinteisempi asiakasohjelma/palvelin-tiedonsiirtotapa, missä data liikkuu suoraan sen kuluttajalta tai luojalta palvelimen analysoitavaksi. Julkaisija/tilaaja-periaatteella palvelin eli välittäjä toimii vain porttina, jolla data ohjataan oikeaan osoitteeseen. Suurin hyöty tästä on, että datan tuottaja ja tilaaja eivät ole koskaan yhteydessä toisiinsa eivätkä näin ollen ole tietoisia toistensa olemassaolosta. Erottelun takia tuottajan ja tilaajan ei koskaan tarvitse tietää toistensa IP-osoitteita tai portteja, niiden ei tarvitse olla samaan aikaan käynnissä ja niiden ei tarvitse toimia synkronoidusti. Erottelulla voidaan helposti yksinkertaistaa monimutkaisiakin järjestelmiä. [5, s. 11]



Kuva 1. *Julkaisija/tilaaja-periaate*

Kuvan 1 esimerkissä on esitelty yksinkertainen julkaisija/tilaaja-sovellus, jossa infonäyttö toimii lämpötiladatan tilaajana ja sensori sen tuottajana. Kyseiset komponentit ovat tässä sovelluksessa välittäjään yhdistetyt asiakkaat (engl. clients). Kuvan vasemmassa laidassa infonäyttö yhdistää välittäjään ja tilaa aiheen ”Lampotila”. Tässä vaiheessa välittäjä ei vielä välitä infonäytön asiakasohjelmalle mitään, sillä kyseiseen aiheeseen ei ole kukaan julkaissut mitään. Kuvan toisessa vaiheessa lämpötila-anturin asiakasohjelma alkaa julkaisemaan samaan aiheeseen lämpötilatietoa. Välittäjä ottaa datan vastaan ja huomaa, että kyseistä aihetta on tilattu. Välittäjä alkaa lähettämään anturin lämpötilatietoa eteenpäin sen tilanneille yksiköille. Viestien sisältö on tarkemmin esitelty alaluvussa 2.4. Tämä on hyvin yksinkertainen sovellus, jolla voidaan helposti havainnollistaa protokollan toimintaa. Usein reaalimaailman toteutuksissa sovellukset ovat huomattavasti monimutkaisempia ja moniosaisempia. Tämän takia ei olisi mitenkään mahdollista käyttää

näin yksinkertaisia aiheita. Erilaisia lämpötilatietoja voi jossain sovelluksissa olla tuhansia, joten aiheet on eroteltava tarkemmin. Aiheita on käsitelty tarkemmin alaluvussa 2.3

2.3 Aiheet

MQTT:ssa viestintä ohjataan niin sanottujen aiheiden (engl. topic) avulla. Aiheiden avulla voidaan jaotella verkossa olevia komponentteja hierarkiatyyppiseen järjestykseen. Aiheet ovat UTF-8-tyyppisiä merkkijonoja, joiden avulla välittäjä pystyy ohjaamaan viestit tilaajille. Aiheet voidaan jaotella tasojen mukaan eri ala-aiheisiin kauttaviivojen avulla. Esimerkiksi aihe ”koti/kylpyhuone/lampotila” voisi esittää kodin kylpyhuoneen lämpötilaa. [5, s. 42]

MQTT-protokollan mukaan aiheiden nimien tulee olla vähintään yhden merkin pituisia merkkijonoja ja ne saavat sisältää välilyöntejä. Aiheita käsiteltäessä tulee ottaa huomioon, että välittäjä tulkitsee isot ja pienet kirjaimet erikseen, joten esimerkiksi ”autotalli/kosteus” on eri aihe kuin ”Autotalli/kosteus”.

Aiheita voidaan tilata protokollan mukaisten villikorttien (engl. wild card) kautta. Tällä voidaan yksinkertaistaa usean samantyyppisen aiheen tilausta yhdellä komennolla. Yhden aihetason korvaus tapahtuu sijoittamalla plusmerkki korvattavan aiheen tilalle. [5, s. 44] Esimerkiksi tilaamalla aihe ”koti/+/lampotila”, voidaan tilata kaikki aiheet missä ensimmäisenä aiheena on ”koti” ja kolmantena ala-aiheena on ”lampotila”. Taulukossa 1 on esitelty esimerkki villikortin käytöstä kodin kylpyhuoneen lämpötilojen avulla.

Taulukko 1. *Yhden aihetason korvaus villikortilla*

`Koti/Kylpyhuone/+/Lampotila`

✓	`Koti/Kylpyhuone/Sauna/Lampotila`
✓	`Koti/Kylpyhuone/Kylpytila/Lampotila`
⊘	`Koti/Kylpyhuone/Sauna/`**`Kosteus`**
⊘	`Koti/`**`Keittio`**`/Jaakaappi/Lampotila`
⊘	`Koti/Kylpyhuone/`**`Sauna/Lattia`**`/Lampotila`

Taulukosta huomataan, että plusmerkki korvaa vain yhden aihetason, ja loput aiheesta on oltava täsmälleen samassa muodossa. Virheelliset osat on taulukossa esitetty lihavoituna.

Usean aiheen villikortin symbolina käytetään risuaitaa. Risuaitaa käyttämällä voidaan korvata monta aihetasoa ja sitä voidaan hyödyntää suurien kokonaisuuksien tilaamisessa.

***Taulukko 2**. Monen aihetason korvaus villikortilla*

`Koti/Kylpyhuone/#`

✓	`Koti/Kylpyhuone/Sauna/Lampotila`
✓	`Koti/Kylpyhuone/Kylpytila/Lampotila`
✓	`Koti/Kylpyhuone/Sauna/Kosteus`
⊘	`Koti/`**`Keittio/Jaakaappi/Lampotila`**
✓	`Koti/Kylpyhuone/Sauna/Lattia/Lampotila`

Taulukossa 2 on esitelty risuaidan käyttöä viestien tilauksessa. Sitä käytettäessä on tärkeää huomata, että merkin tulee olla aihemerkkijonon viimeisenä. Merkkijonon alun tulee olla täsmälleen samassa muodossa kuin tilauskomennon. Mikäli jostain syystä tilaaja haluaa kaikki välittäjälle menevät viestit, sen voi tehdä tilaamalla pelkän risuaidan. Tätä ei kuitenkaan suositella, sillä se voi olla äärimmäisen kuormittavaa välittäjälle.[5, s. 44]

MQTT-protokollan mukaan aiheet voi nimetä vapaasti, mutta tietyt merkkijonot on varattu välittäjän datankeräykseen. ”$SYS”-alkuisia aiheita on mahdollista pelkästään tilata, mutta niihin ei ole mahdollista julkaista mitään. Tällaisia välittäjälle varattuja aiheita ovat muun muassa

```
$SYS/broker/clients/connected
$SYS/broker/messages/received
$SYS/broker/subscriptions/count.
```

Näistä aiheista voi kukin asiakasohjelma tilata välittäjältä muun muassa sen versiotiedot, viestimäärät sekä siihen yhteydessä olevien muiden asiakkaiden määrän. [6]

2.4 Viestit ja viestityypit

Tähän asti MQTT:sta on esitelty protokollan toimintaperiaatetta ja aiheiden käsitteitä. Tämän alaluvun tarkoituksena on käsitellä MQTT-viestejä, niiden tyyppejä ja tarkoituksia. MQTT:n yleisimmät peruskomentotyypit ovat yhteyskokeilu (engl. ping), julkaisu, tilaus, yhdistäminen sekä yhteyden katkaisu. Lisäksi MQTT:ssa on ominaista, että komentojen vastineena lähetetään tiedostusviesti vastaanotetusta komennosta lähettäjälle. Nämä viestityypit ovat hyvin olennaisia jatkuvan yhteyden ylläpidon kannalta. Tiedostusviestit sisältävät tiedot siitä, saatiinko komento suoritettua ja mikä lopputulos sillä

oli. Näiden lisäksi protokollasta löytyy lisäkomentoja julkaisun eri tiedostustoiminnoille ja ne on käsitelty viestien laatutasoa käsittelevässä alaluvussa 2.6. [2, Luku 2]

Koska MQTT on rakennettu TCP/IP-protokollan päälle, on sekä välittäjän että asiakaslaitteiden kyettävä ymmärtämään TCP/IP-protokollan kieltä. Viestit eivät kuitenkaan ole perinteisiä TCP/IP-paketteja vaan protokollan tarpeisiin luotuja MQTT-paketteja. [2, Luku 3]

2.4.1 Yhteyteen liittyvät komennot

Tämän alaluvun aiheena ovat yleisimmät MQTT-yhteyteen liittyvät peruskomentotyypit, joihin kuuluvat CONNECT, CONNACK, DISCONNECT, PINGREQ ja PINGRESP. Jokaiselle paketille on oma tietokehys. Nämä kehykset ja niiden tietokentät esitellään tässä alaluvussa.

Peruspaketeista laajin paketti on yhdistäminen (engl. connect). Tämä paketti sisältää tiedot yhteyttä yrittävästä asiakkaasta. Sen on oltava ensimmäinen paketti asiakasohjelmalta, jotta välittäjä suostuu aloittamaan kommunikoinnin asiakasohjelman kanssa. CONNECT-paketti on esitelty kuvassa 2 [7, Luku 3.1].

Kuva 2. *CONNECT-paketti*

Paketin tiedoista tärkein on *clientID.* Se on jokaiselle asiakasohjelmalle yksilöllinen tunniste merkkijonon muodossa, jolla välittäjä pystyy erottelemaan eri asiakasohjelmat toisistaan. Välittäjä tarkistaa tunnisteen ja mikäli se on jo käytössä jollain toisella asiakkaalla, yhteysyritys katkaistaan. Toinen pakollinen tieto asiakasohjelmalta on *cleanSession*-tieto. Se on BOOL-muodossa oleva muuttuja, mikä kertoo välittäjälle, haluaako asiakasohjelma aloittaa kokonaan uuden istunnon vai tulisiko välittäjän palauttaa kyseisen asiakkaan mahdollinen viestihistoria yhteydenluonnin yhteydessä. [7, Luku 3.1.2]

Käyttäjänimi ja salasana ovat välittäjän konfigurointitietoihin asetetut tunnisteet, joiden avulla välittäjä voi tunnistaa asiakasohjelman yhteyspyynnön. Ne eivät ole pakollisia, mutta hyvin suositeltavia turvallisuuden parantamiseksi. Tunnistamiseen liittyy olennaisesti TSL/SSL-salaus, mikä on käsitelty tarkemmin alaluvussa 2.7. [7, Luku 3.1.2]

Välittäjän on tarkistettava säännöllisin väliajoin yhteyden toimivuus asiakasohjelmaan. Tähän käytetään *keepAlive*-muuttujaa. Se on maksimiaika sekunteina, minkä välittäjä sietää olla erossa asiakasohjelmasta. Mikäli kyseinen aika ylittyy ilman viestejä, asettaa välittäjä asiakasohjelman yhteyden katkaistuksi. Tämänlaisissa yllättävissä yhteydenkatkaisuissa välittäjä julkaisee asiakasohjelman asettaman viimeisen tahdon (engl. last will). Tällä voidaan stabiloida järjestelmiä mahdollisesti epävakaisissa yhteyksissä. Viimeistä toivoa on käsitelty tarkemmin alaluvussa. 2.6. [7, Luku 3.1.2.10]

***Kuva 3**. CONNACK-paketti ja palautuskoodit*

Jokaisen yhteydenmuodostusyrityksen jälkeen välittäjän on lähetettävä CONNACK-paketti takaisin asiakasohjelmalle. Sen on oltava ensimmäinen paketti välittäjän suunnalta, jotta yhteys voidaan muodostaa. Yhteydenmuodostuksen tiedostus (engl. Connect acknowledgement) sisältää kaksi muuttujaa, joilla ilmaistaan, onnistuiko yhteys. *SessionPresent*-muuttujalla välittäjä kertoo asiakasohjelmalle, löytyykö välittäjän historiatiedoista edellisiä istuntoja. Mikäli asiakasohjelma on yhteydenmuodostuksessa halunnut täysin uuden istunnon, tämän muuttujan arvoksi tulee epätosi. Jos asiakasohjelma haluaa palauttaa edellisen istunnon ja sellainen löytyy välittäjän historiasta, tämän muuttujan arvoksi asettuu tosi. CONNACK-paketti sisältää myös tiedon yhteydenmuodostuksen

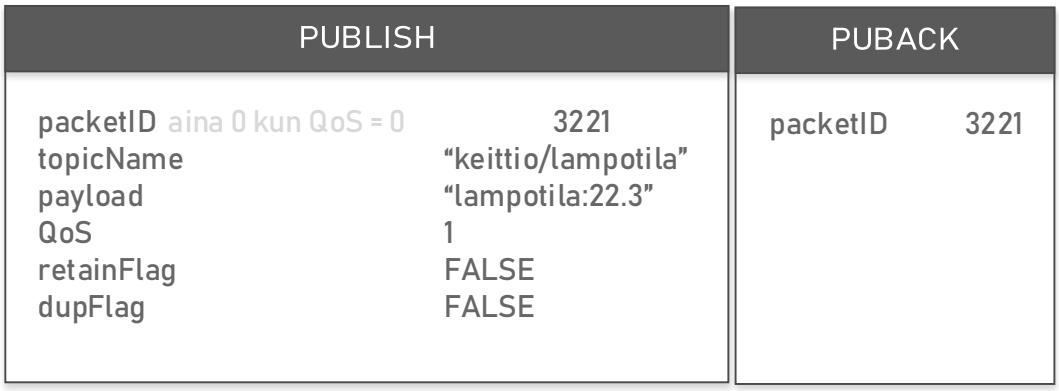
onnistumisesta. Eri palautuskoodit on numeroitu nollasta viiteen, ja ne kertovat asiakasohjelmalle syyn yhteyden mahdollisesta epäonnistumisesta. Eri palautuskoodit on esitelty kuvassa 3. [7, Luku 3.2]

Yhteyteen liittyviä paketteja on käsiteltyjen lisäksi DISCONNECT, PINGREQ ja PINGRESP. Nämä paketit eroavat yhteydenmuodostuspaketeista siten, että niillä on vain tyyppi, mutta ei lainkaan kuormaa. Yhteydenkatkaisu (engl. disconnect) on paketti, jonka asiakasohjelma voi lähettää halutessaan ilmaista niin sanotun puhtaan katkaisun. Tässä tapauksessa välittäjä tietää, että asiakasohjelma katkaisee yhteyden tietoisesti, eikä esimerkiksi heikon yhteyden takia. [2, Luku 3]

Yhteyden ylläpitoon käytetyt paketit ovat yhteyskokeilupyyntö (engl. ping request) ja yhteyskokeiluvastaus (engl. ping response). Välittäjä varmistaa yhteyden asiakasohjelmaan säännöllisin väliajoin yhteyskokeilun avulla. CONNECT-paketissa määritelty *keepAlive*-muuttuja määrittelee säännöllisen yhteyskokeilun maksimiajan. Välittäjä määrittelee yhteyden katkenneeksi, mikäli se ei saa yhteyskokeiluvastausta asetetun ajan kuluttua. [2, Luku 3]

2.4.2 Julkaisuun ja tilaukseen liittyvät komennot

Tässä alaluvussa käsitellään komennot, jotka liittyvät aiheiden tilaukseen ja julkaisuun. Tämänlaisia komentoja ovat muun muassa SUBSCRIBE, SUBACK, PUBLISH ja PUBACK. Tilaus- ja julkaisukomennoissa on yhdistämiskomentojen tavoin omat kuittausviestit, jotka sisältävät tiedot mahdollisista virhetilanteista. Kaikki paketit sisältävät *packetID*-muuttujan, millä jokainen komento voidaan yksilöidä ja näin ollen osoittaa kuittaus täysin oikealle viestille. Asiakasohjelmat voivat lähettää tilaus- ja julkaisukomentoja välittömästi yhdistämisen jälkeen. [2, Luku 4]



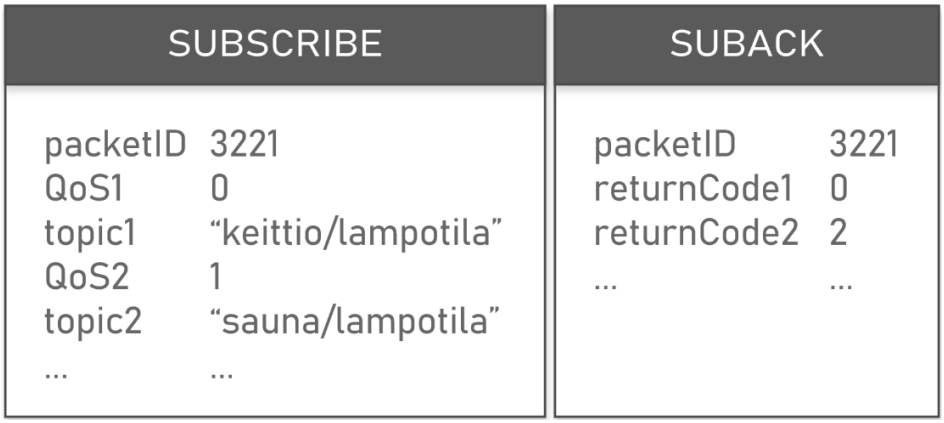
Kuva 4. *PUBLISH-ja PUBACK-paketti*

PUBLISH-paketti sisältää pakettitunnisteen, aiheen, viestisisällön, palvelun laatutason sekä säilytys- ja kaksoiskappaletiedot. MQTT:n datariippumattomuuden ansiosta viestit

voivat olla tyypiltään mitä vain. Tieto siirtyy verkon yli bittimuodossa ja asiakasohjelmien on huolehdittava, siitä miten tieto loppujen lopuksi käsitellään. Datan muutos bittimuotoon ja takaisin on usein huolehdittu valmiiden asiakasohjelmapohjien ohjelmakoodeissa. [5, s. 37]

Julkaistaville komennoille voidaan asettaa eri palvelun laatutasot (engl. Quality of service) viestin tärkeyden mukaan. Jokaisella eri laatutasolla on omat kuittausviestit. Niin sanottua QoS-tasoa käsitellään tarkemmin alaluvussa 2.6. [5, s. 37]

Säilytys-muuttujalla (engl. Retain) lähettävä asiakasohjelma voi määritellä haluaako viestin jäävän välittäjän tietoihin. Näin ollen uudet asiakasohjelmat saavat tilaushetkellä automaattisesti välittäjälle säilötyt viestit ilman että lähettävän asiakasohjelman on julkaistava niitä uudestaan. Kaksoiskappalemuuttujalla (engl. duplicate) asiakasohjelma voi kertoa lähettävänsä saman viestin toista kertaa. Tämä tapahtuu usein silloin kun ensimmäistä viestiä ei ole kuitattu ja asiakasohjelma haluaa varmistaa, että viesti menee perille. [5, s. 37]

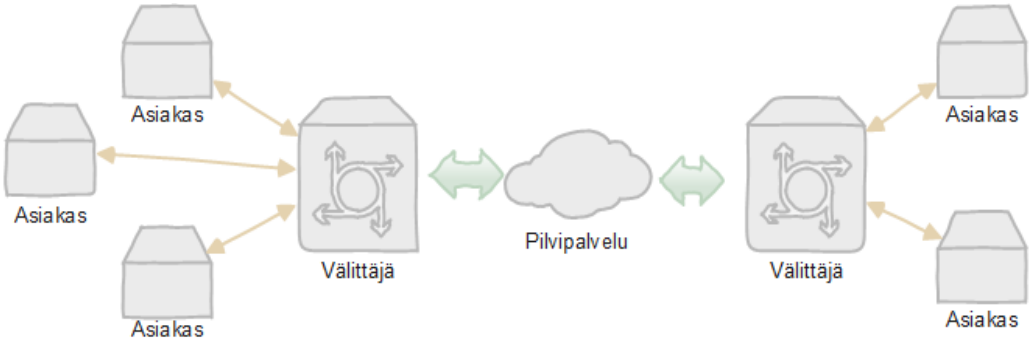


***Kuva 5**. SUBSCRIBE- ja SUBACK-paketti*

Tilauspaketti ja tilauksenkuittauspaketti on esitetty kuvassa 5. Tilauspaketti sisältää yhden tai useamman tärkeystaso/aiheparin sekä yhden pakettitunnisteen. Asiakasohjelma voi siis yhdellä tilauskomennolla tilata useita aiheita. Vastauksena tilaava asiakasohjelma saa SUBACK-komennon, joka sisältää pakettitunnisteen sekä palautuskoodit jokaiselle tilatulle aiheelle. Onnistuneen tilauksen palautuskoodi on 0,1 tai 2 tilatun aiheen tärkeystason mukaan. Mikäli tilaus epäonnistuu, protokollassa on tälle asetettu palautuskoodiksi numero 128. [5, s. 33]

2.5 Välittäjä

MQTT:n toiminnan kannalta kriittisin osa on välittäjä (engl. broker). Välittäjän tehtävänä on toimia kommunikoinnin keskipisteenä ja välittää viestikomennot eteenpäin. Viestinnän ohjauksen lisäksi välittäjän tärkeimpiä tehtäviä on järjestelmän ylläpito ja hallinta. Yksinkertaisimmillaan se on komentoriviltä ajettava ohjelma, jonka konfigurointitiedostoon asetetaan halutut asetukset. Järjestelmän pystytysvaiheessa välittäjälle voidaan asettaa useita asetuksia, joilla määrätään asiakasohjelmien oikeuksia ja toimintatapoja. Sille voidaan asettaa esimerkiksi suuri mahdollinen viestikoko, autentikointivaatimukset sekä lokintallennusasetukset. Välittäjän asetuksiin tutustutaan käytännön työn alaluvussa 3.3. [2, Luku 3]



***Kuva 6**. Pilvipalvelulla ketjutettu MQTT-järjestelmä*

Välittäjiä voidaan ketjuttaa useita sarjaan siltauksen avulla, joten järjestelmistä voidaan luoda erittäin laajoja kokonaisuuksia. Välittäjän ketjutuksissa voidaan hyödyntää esimerkiksi pilvipalveluita. Monet IoT-palveluntarjoajat tarjoavat valmiita alustoja MQTT-protokollan käyttöön. Siltauksen yhteydessä välittäjistä tulee itsekin asiakasohjelmia, ja ne lähettävät normaalisti viestejä toisten välittäjien välillä. Tämän lisäksi on tarjolla myös suoraan pilvessä toimivia välittäjiä. [8]

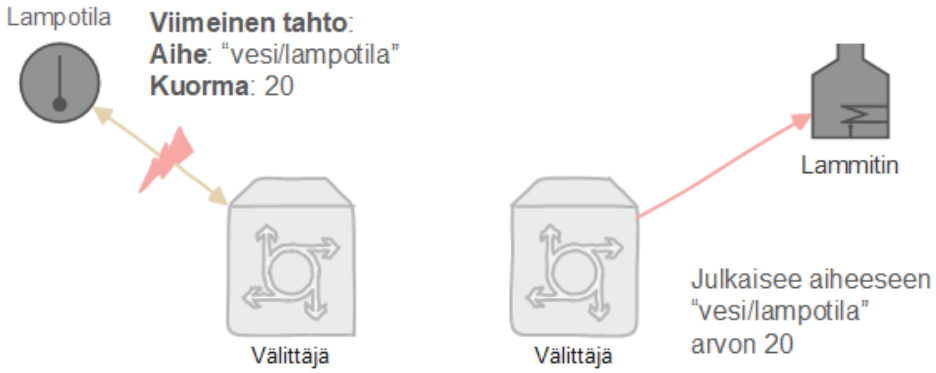
Markkinoilta löytyy sekä kaupallisia että avoimen lähdekoodin välittäjäohjelmistoja. Yleisimpiä kaupallisista ovat HiveMQ ja CloudMQTT. Avoimen lähdekoodin välittäjistä ylivoimaisesti suosituin on Eclipse-projektin Mosquitto-välittäjä, jota aiotaan käyttää myös käytännön työn toteuttamisessa. Sen lisäksi monissa projekteissa on käytössä RabbitMQ-välittäjä, joka tukee myös muita tiedonsiirtoprotokollia. Se on erityisen suosittu esimerkiksi startup-yrityksissä sen laajan käyttöjärjestelmätuen ansiosta [9]. Suurimpana erona kaupallisten ja ei-kaupallisten välittäjien välillä on asiakastuki sekä käyttömukavuus. Kaupallisissa välittäjissä on usein graafinen käyttöliittymä asetusten ja datan seu-

rantaan sekä kattava asiakastuki mahdollisten ongelmien selvittämiseen. Ei-kaupallisissa on ominaista karumpi komentorivikäyttöliittymä sekä asetusten asettaminen konfigurointitiedostoilla. [10]

2.6 LWT ja QoS

Tässä alaluvussa käsitellään MQTT-protokollan kaksi olennaista ominaisuutta: viimeinen tahto (engl. Last will) sekä palvelun laatutaso (engl. Quality of service). Nämä ominaisuudet ovat yksi osittainen syy, miksi MQTT on kasvattanut suosiotaan IoT-sovelluksissa. Niillä pyritään stabiloiman sovelluksia epävakailla Internet-yhteyksillä sekä samalla priorisoimaan viestejä niiden tärkeyden perusteella.

Asiakasohjelma pystyy yhteydenmuodostuksessa asettamaan itselleen viimeiseen tahtoon liittyvät parametrit. Se on aihe ja arvo, jonka asiakasohjelma haluaa välittäjän julkaisevan, kun yhteys näiden välillä katkeaa yllättävästi. [2, Luku 9]



***Kuva 7**. Viimeisen tahdon toiminta*

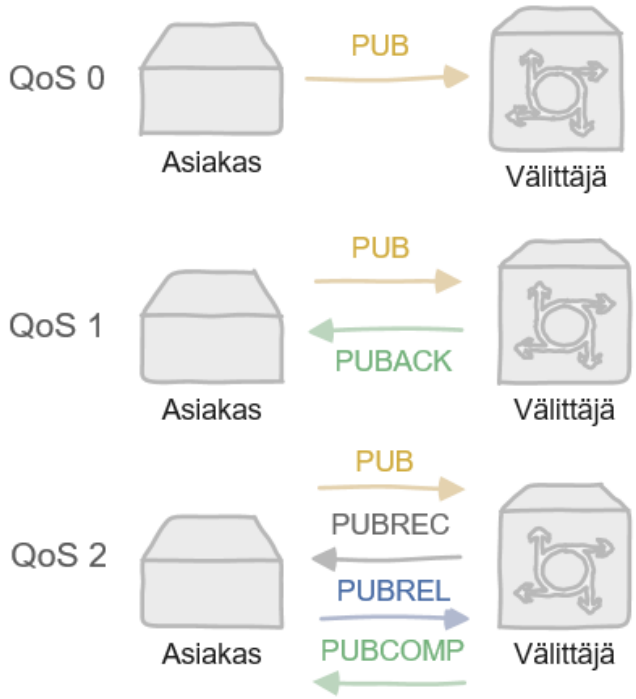
Kuvan 7 esimerkissä esitellään mahdollinen reaalimaailman esimerkki viimeisen tahdon käytöstä. Kuvassa lämpötila-anturi on asettanut yhteydenmuodostuksessa oman viimeisen tahtonsa välittäjälle. Lämmitin on tilannut nämä aiheet. Välittömästi kun anturin ja välittäjän välinen yhteys katkeaa yllättävästi, välittäjä julkaisee tähän aiheeseen anturin asettaman arvon. Tässä tapauksessa anturi pystyy viimeisen tahdon avulla määrittämään oman viimeisen lämpötila-arvonsa, vaikka yhteys olisi katkennut.

Palvelun laatutaso on viestin lähettäjän ja vastaanottajan välinen sopimus siitä, kuinka tärkeä yksittäinen viesti on. Viestin lähetyksen yhteydessä lähettävä osapuoli pystyy määrittämään, miten kyseinen viesti tulisi käsitellä. Palvelun laatutaso on yksi MQTT-protokollan avainominaisuuksista, sillä sen avulla voidaan erotella elintärkeät viestit tavallisista viesteistä. Viestille voidaan lähetysvaiheessa antaa yksi kolmesta laatutasosta. Ne on esitelty taulukossa 3. [5, s. 46]

***Taulukko 3**. Palvelun laatutasot*

QoS	Merkitys
0	Lähetys enintään kerran
1	Lähetys vähintään kerran
2	Lähetys täsmälleen kerran

Pienin mahdollinen laatutaso on 0. Siinä tapauksessa viestin lähettäjä on määrittänyt, että viesti tulisi lähettää resurssien salliessa parhaalla mahdollisella tavalla. Viestin lähettäjä ei kuitenkaan odota vastaanottajan lähettävän takaisin tiedostusta eli PUBACK-viestiä, jolla varmistetaan, että viesti olisi vastaanotettu. Pienintä laatutasoa käytetään yleensä hyvin yleisten ja usein lähetettävien viestin yhteydessä. Esimerkiksi jos lämpötilaa lähetetään sekunnin välein, ei ole tarpeellista kuitata jokaista viestiä. Tällä tavoin vähennetään järjestelmän kuormaa huomattavasti. [5, s. 46]



***Kuva 8**. Viestityypit QoS-tasoille*

Mikäli viestistä halutaan kuittaus, voidaan käyttää laatutasoa 1. Sillä voidaan määrätä vastaanottaja lähettämään kuittaus, eli PUBACK-viesti. Lähettäjä säilyttää ja yrittää lähettää viestiä, kunnes se on saanut varmistuksen viestin toimituksesta. Tässä tapauksessa saadaan takuu, että viesti on mennyt perille ainakin kerran, mutta ei voida taata, ettei se ole mennyt perille monta kertaa. Tämän takia järjestelmän on kyettävä käsittelemään mahdollisia duplikaattiviestejä. Tätä laatutasoa käytettäessä on huomioitava, että

järjestelmässä liikkuvien viestien määrä tuplaantuu jokaisen kuittausviestin takia. [2, Luku 6]

Suurin palvelun laatutaso on 2. Sillä lähettäjä voi varmistaa, että kyseinen viesti menee perille, mutta vain yhden kerran. Se on laatutasoista raskain ja hitain, mutta samalla myös turvallisin. Raskaus johtuu siitä, että alkuperäinen viesti halutaan lähettää vain kerran, mutta samalla halutaan varmistua, että se on mennyt perille. Koska alkuperäistä viestiä ei tässä tilanteessa voida lähettää kuin kerran, on protokollan kyettävä muilla tavoin varmistamaan, että se on saapunut kohteeseen. Tämän takia MQTT-protokollaan on rakennettu kuvassa 8 näkyvät PUBREL, PUBREC sekä PUBCOMP tietokehykset. Näiden avulla asiakasohjelmat voivat tiedustella toisiltaan mikä tilanne viestin lähetyksessä on koskematta alkuperäiseen viestiin. Tiedusteluviestit ovat muutoin tyhjiä mutta sisältävät pelkästään 2.4.2 alaluvussa esitellyt *packetID*-muuttujan. [2, Luku 6]

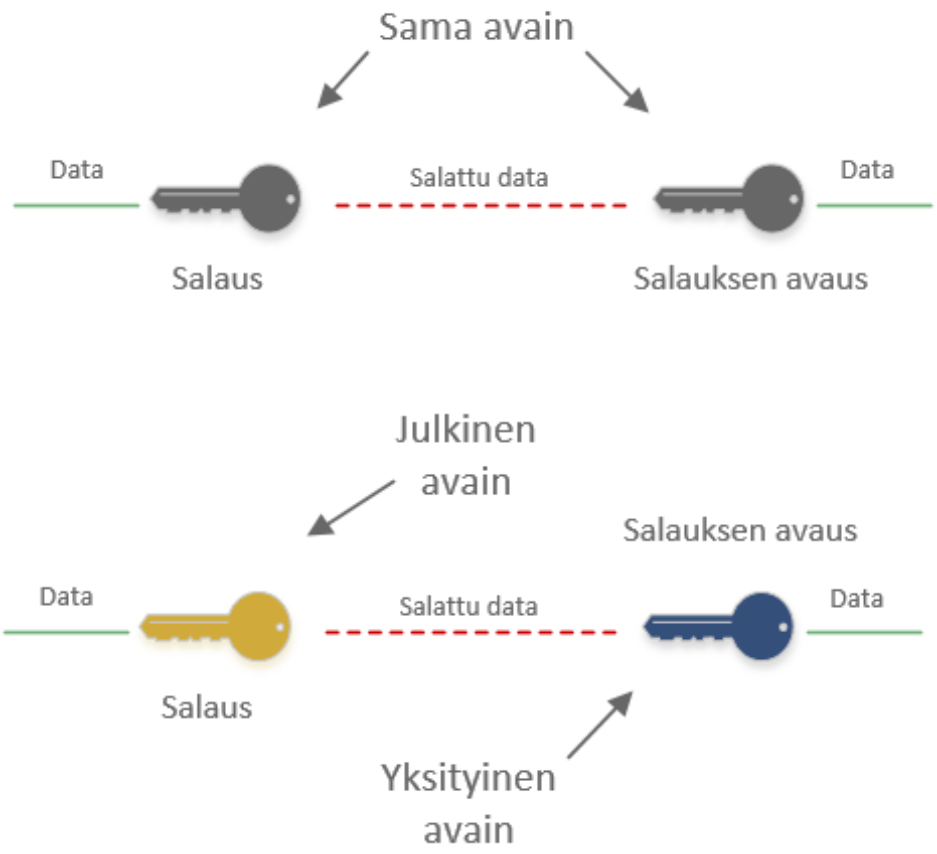
2.7 Salaus ja turvallisuus

Tietoturva on yksi merkittävimmistä asioista, mikä tulee ottaa huomioon IoT-järjestelmiä suunniteltaessa. Voidaan ajatella, että mikäli tietoliikenne olisi täysin salaamaton, voisi kolmas osapuoli hyvin helposti lukea ja modifioida verkossa liikkuvaa dataa ja näin ollen täysin romahduttaa järjestelmän toiminnan. Salauksen ja turvallisuuden tärkeyttä on siis vaikea vähätellä. Tämän kappaleen tarkoituksena on esitellä, miten suojaus toimii ja miten sitä voidaan hyödyntää MQTT-sovelluksissa.

MQTT:ssa ja IP-verkkosovelluksissa yleensä käytetään TSL/SSL-salausprotokollaa. Protokolla on hyvin tunnettu ja sitä käytetään muun muassa web-selaamisen suojaukseen erityisesti Internetissä käytävässä kaupankäynnissä. Web-selaimissa suojaus on usein käyttäjältä piilossa ja toimii automaattisesti. Verkossa liikkuva tieto on salattu, siten että sitä ei ole mahdollista lukea tai muokata ilman oikeita avaimia. [11]

Verkkosovelluksissa käytettävä kryptografia eli salaustekniikka voidaan jakaa kahteen kategoriaan: symmetriseen ja asymmetriseen tekniikkaan. Symmetrisessä salaustekniikassa kommunikoinnin molemmille osapuolille on jaettu sama avain, jolla viestin voi sekä salata että avata. Data salataan lähettäjän päässä ja liikkuu verkon läpi salattuna. Symmetrisessä salauksessa ongelmana on, että mikäli kolmas osapuoli saa avaimen haltuunsa, pystyy se tulkitsemaan salattua liikennettä. Huomattavasti turvallisemmaksi todettu ja yleisemmin käytetty on epäsymmetrinen eli asymmetrinen salaustekniikka. Siinä käytetään avainpareja, jotka koostuvat julkisesta (engl. public key) sekä yksityisestä (engl. private key) avaimesta. Avaimet ovat erilaisia, mutta niillä on matemaattinen yh-

teys. Tämä yhteys on hyvin monimutkainen ja on käytännössä mahdoton avata. Nimensä mukaisesti julkinen avain voidaan jakaa kenelle vain ja sitä käytetään pelkästään viestin salaamiseen. Viestin avaamiseen ei ole mahdollista käyttää samaa avainta, vaan se on tehtävä yksityisellä avaimella. [12, ss. 3–5]



***Kuva 9**. Symmetrinen ja asymmetrinen salaus*

Salausta käytettäessä tulee ottaa huomioon järjestelmän nopeus. Vaikka salaus onkin hyvin tärkeää, se ei saa hidastaa järjestelmän toimintaa liikaa. Asymmetrisen salauksen tapauksessa järjestelmän on aina tarkastettava julkisen ja yksityisen avaimen matemaattinen yhteys, mikä itsessään vaatii paljon laskentatehoa. [13]

TSL/SSL-salaus käyttää symmetrisen ja asymmetrisen salaustekniikan yhdistelmää. Järjestelmä tarkistaa matemaattiset yhteydet vain yhteydenmuodostuksen alkuvaiheessa ja luo näiden perusteella kyseiselle istunnolle omat istuntoavaimet. Tällä tavalla järjestelmä säästää suuren määrän resursseja ja salattu liikenne voidaan saavuttaa kevyemmin. Tämänlaista yhteydenmuodostuksessa tapahtuvaa salauksen määritystä kutsutaan TSL/SSL-kättelyksi (engl. handshaking). [14, s. 145]

MQTT-protokollassa TSL/SSL-salaus on toteutettu web-sivujen tapaan varmenteiden avulla. Se on elektroninen dokumentti, jolla voidaan todistaa, että osapuoli on se mikä

väittää olevansa. Varmenne sisältää edellä mainittujen avainten lisäksi elektronisen allekirjoituksen, jonka on allekirjoittanut joku luotettavaksi todettu palvelu. Varmenteita tarjoavia palveluita on Internetissä runsaasti, mutta ne voidaan luoda myös itse avoimen lähdekoodin avulla. MQTT:ssa välittäjälle ja asiakasohjelmille asetetaan omat varmenteet ja välittäjä tarkistaa yhteydenmuodostuksessa varmenteiden aitouden. [11]

MQTT:ssa on mahdollista asettaa jokaiselle asiakasohjelmille myös yksilölliset käyttäjätunnukset sekä salasanat. Tällä voidaan lisätä matalan tason turvallisuutta, jos ei haluta käyttää varmenteita. Välittäjälle asetetaan käyttäjänimi-salasana pareja, jotka se säilyttää salattuina tiedoissaan. Kun uusi asiakas pyrkii ottamaan yhteyttä välittäjään, se vertaa asiakkaan toimittamaa käyttäjänimi-salasana paria omissa tiedoissaan oleviin pareihin. Jos parit ovat yhdenmukaiset, välittäjä sallii asiakasohjelman yhteydenmuodostuksen. [2, Luku 3]

MQTT-protokollaan on rakennettu myös pääsynhallinta (engl. access control). Tällä voidaan estää asiakkaita julkaisemasta tai tilaamasta dataa määritettyihin aiheisiin. Sen avulla voidaan luoda järjestelmä turvallisemmaksi ja stabiilimmaksi, kun tietyillä asiakkailla on oikeus vain niille tärkeisiin aiheisiin. Tällaisia aiheita ovat muun muassa välittäjän omaan datankeruuseen tarkoitetut aiheet, joista mainittiin alaluvussa 2.3. Tässä työssä toteutettavassa käytännön työssä pääsynhallintaa ei ole käytetty, mutta suuremmissa järjestelmissä sen käyttö on lähes välttämätöntä. Pääsynhallintaan liittyvät konfiguraatiot asetetaan käyttöönotossa suoraan välittäjälle ja asiakasohjelmat eivät saa tietoa näistä asetuksista. Hallintaan liittyvät estot voidaan asettaa myös alaluvussa 2.3 esitettyjen villikorttien muodossa. [15]

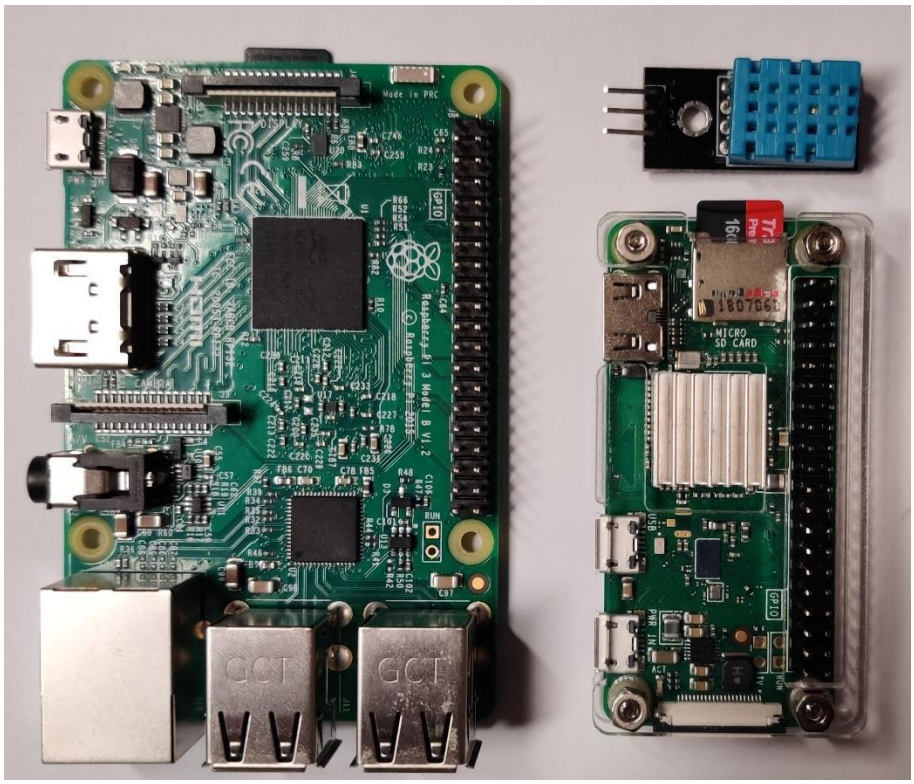
3. KÄYTÄNNÖN TYÖN TOTEUTUS

3.1 Käytännön työn tavoite

Käytännön työn tavoitteena on luoda ja ottaa käyttöön yksinkertainen kodin valvontajärjestelmä kuvan 1 mukaisesti. Laitteiden välinen kommunikointi toteutetaan käyttäen MQTT-protokollaa Raspberry Pi-pienoistietokoneiden välityksellä. Sen tarkoituksena on selvittää protokollan soveltuvuus yksinkertaisiin IoT-järjestelmiin ja todentaa millainen prosessi MQTT:n perustuvan sovelluksen käyttöönottaminen on. Sovellukseen ohjelmoidaan kaksi asiakasohjelmaa kahta eri ohjelmointikieltä käyttäen. Tällä halutaan todentaa protokollan ohjelmointikielituki käytännössä. Lämpötiladataa lähettävä asiakasohjelma on tarkoituksena ohjelmoida Pythonilla ja siitä on tarkoitus tehdä hyvin yksinkertainen ja kevyt. Dataa tilaava asiakasohjelma on tavoitteena tehdä Qt-ohjelmointiympäristössä hyödyntäen C++ ohjelmointikieltä. Sille on tarkoitus luoda graafinen näkymä, jossa lämpötila näkyy reaaliaikaisena. Järjestelmästä on tarkoitus tehdä modulaarinen ja sellainen, että sen mahdollinen laajentaminen olisi yksinkertaista. Järjestelmästä on tarkoitus luoda salattu käyttämällä TSL/SSL-salausta

3.2 Komponentit

Komponenteiksi soveltuvuusselvitykseen valikoituivat Raspberry Pi 3B, Raspberry Pi Zero W sekä DHT11-anturi. Kuvassa 10 vasemmalla oleva Raspberry Pi 3B valikoitui sen kokoonsa nähden tehokkaan suorituskyvyn ja laajan kosketuspaneelituen ansiosta. Kortista löytyvien yleiskäyttöpinnien (engl. General Purpose Input Output, GPIO) ansioista siihen voi kytkeä erilaisia laitteita ja antureita. Tässä työssä kyseisiin pinneihin kytkettiin kosketuspaneeli graafista käyttöliittymää varten. Tähän tietokoneeseen asennettiin Raspberry Pi -tietokoneille tarkoitettu Linuxiin pohjautuva Raspbian-käyttöjärjestelmä.



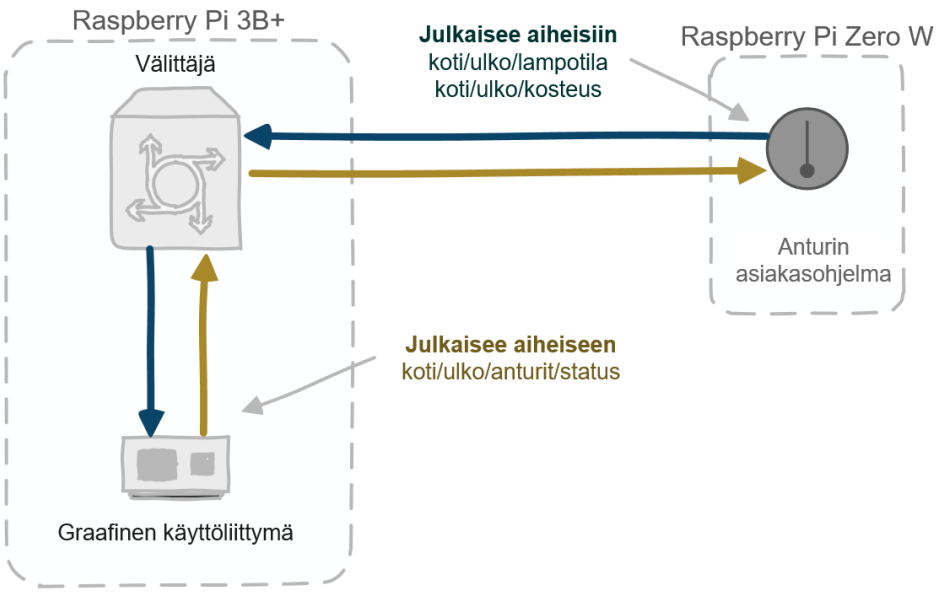
***Kuva 10**. Raspberry Pi 3B, Raspberry Pi Zero W ja DHT11-anturi*

Kuvassa 10 oikealla alhaalla oleva Raspberry Pi Zero W otettiin käyttöön lämpötila-anturin asiakasohjelmaa varten. Se soveltui tähän tarkoitukseen hyvin pienen kokonsa, WiFi-tuen ja yleiskäyttöpinnien ansiosta. Myös tähän tietokoneeseen asennettiin Raspbian-käyttöjärjestelmä. Tietokoneen pinneihin kytkettiin työssä käytettävä DHT11-anturi, jolla voidaan mitata lämpötilaa sekä kosteutta. Kyseinen anturi on esitelty kuvassa 10 oikealla yläkulmassa.

3.3 Ympäristö, asennus ja konfigurointi

Järjestelmä rakennettiin kodin langattomaan sisäverkkoon. Lämpötila-anturin tietokoneeseen asennettiin Python3-tulkki sekä kappaleessa 2.1 mainittu avoimen lähdekoodin Paho-projektin MQTT-kirjasto Pythonille. Näiden lisäksi järjestelmästä aktivoitiin SSH-yhteys konfigurointia varten. Tietokoneen pinneihin kytkettiin DHT11-anturi. Välittäjätietokoneseen asennettiin Qt-ympäristö, QMQTT-kirjasto sekä mosquitto-välittäjä. Myös tähän tietokoneeseen aktivoitiin SSH-yhteys.

Ennen varsinaista ohjelmointia järjestelmälle oli luotava hierarkia aiherakennetta varten. Vaikka järjestelmä on kooltaan hyvin pieni, siitä on tarkoitus tehdä helposti kasvatettava.



Kuva 11. *Järjestelmän rakenne ja aiheet*

Järjestelmän rakenne ja valitut aiheet on esitelty kuvassa 11. Kuvasta voidaan huomata, että anturin arvoa lähettävä asiakasohjelma julkaisee kahteen aiheeseen anturin arvoa. Kyseisten viestien kuormana on anturin arvoa vastaava liukuluku. Qt:en perustuvan graafisen käyttöliittymän asiakasohjelma on tilannut nämä aiheet visualisointia varten. Käyttöliittymästä löytyvistä painikkeista voidaan myös ohjata antureiden tilaa. Mikäli jostain syystä ei haluta, että anturiohjelma lähettää viestejä, se voidaan ohjata pois päältä lähettämällä antureiden status-aiheeseen viesti ON tai OFF halutun tilan mukaan.

Järjestelmälle luotiin varmenteet OpenSSL-ohjelmalla. Se on avoimeen lähdekoodin perustuva ohjelmakirjasto, jolla voi luoda vapaasti omaan käyttöön TLS/SSL-varmenteita. Välittäjän konfiguraatiotiedostosta muutettiin seuraavat arvot:

```
cafile /etc/mosquitto/certs/ca.crt
keyfile /etc/mosquitto/certs/server.key
certfile /etc/mosquitto/certs/server.crt
port 8883.
```

Varmenteille asetettiin oikeat tiedostopolut ja MQTT-liikenteen portti vaihdettiin porttiin 8883. Tämä portti on yleisesti vakiintunut salatun MQTT-liikenteen käyttöön. Salaamattomassa liikenteessä käytetään porttia 1883.

3.4 Asiakasohjelmien esittely

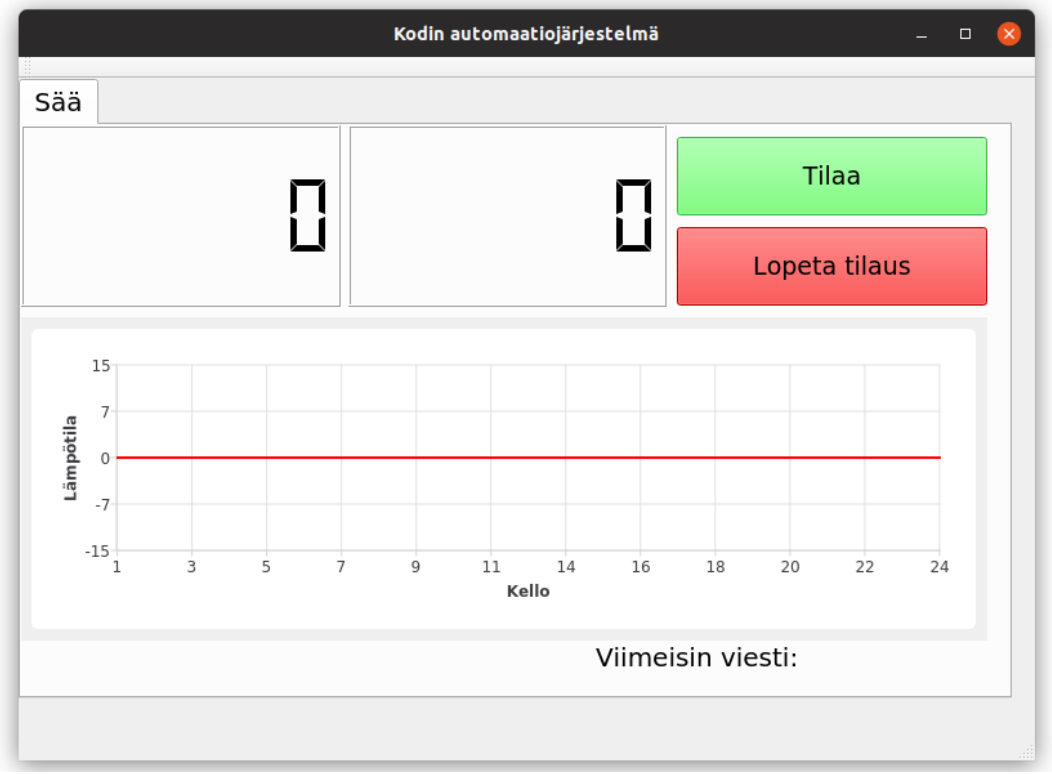
Tässä alaluvussa esitellään soveltuvuusselvitystä varten kirjoitetut asiakasohjelmat. Ohjelmat ovat perusrakenteeltaan samanlaiset, vaikka niissä on käytetty eri ohjelmointikieliä. Ohjelmien esittelyssä on keskitytty käsittelemään MQTT-protokollan kannalta tärkeimmät ohjelmien osat. Molemmat ohjelmakoodit on esitetty liitteissä. Qt-ohjelmistoympäristössä luotu asiakasohjelma löytyy liitteestä A ja Pythonilla tehty ohjelma löytyy liitteestä B. Tässä luvussa viitataan liitteiden ohjelmakoodeista löytyviin rivinumeroihin. Käyttöliittymäohjelma on C-kielien tapaan jaettu otsikko- ja kooditiedostoon. Tähän ohjelmaan viitattaessa tarkoitetaan varsinaisen kooditiedoston rivinumeroita, jotka alkavat liitteen A sivulta 3. Ohjelmakoodien laajempi toiminta on kommentoitu itse koodin sekaan.

Molemmissa ohjelmissa ohjelman suoritus alkaa alkuarvojen asetuksilla ja varmenteiden lukemisella. Käyttöliittymäohjelman Qt-koodissa varmenteet luetaan riveillä 82-89. Asiakasohjelmalle luotu varmennetiedosto luetaan ja ajetaan Qt-kirjastosta löytyvään varmennelistaan. Tässä tapauksessa varmenne löytyy ohjelmaprojektin juuresta, joten tiedosto-oliolle voidaan antaa parametriksi pelkästään tiedoston nimi. Lopuksi luodaan asiakasohjelmaolio, joka ottaa parametrikseen välittäjän IP-osoitteen, portin sekä luodun konfiguraatio-olion.

Anturin Python-koodissa asiakasohjelmaolio luodaan rivillä 34 ja sille annetaan vastaavasti varmennetiedosto rivillä 42. Tässä tapauksessa asiakasohjelmalle on jo annettu osoite sekä portti alkuarvojen yhteydessä. Pythonin MQTT-kirjastoja käytettäessä on huomioitava TLS/SSL-salauksen versio. Valmis kirjasto on hyvin tarkka siitä, että versionumero pätee välittäjän käyttämään salausversioon.

Ennen kuin kumpikaan asiakasohjelma ottaa yhteyden välittäjään, ne asettavat itselleen viimeisen tahdon. Qt:ssa viimeinen tahto asetetaan koodin rivillä 96. Tällä halutaan varmistaa, että mikäli käyttöliittymän yhteys välittäjään katkeaa, tulee myös antureiden lopettaa viestin lähetys. Anturin ohjelma on tilannut tämän kyseisen aiheen ja reagoi siihen julkaistavaan viestiin lopettamalla omien viestin lähettämisen. Vastaavasti Python-koodissa viimeiseksi tahdoksi asetetaan rivillä 41, jolloin ohjelma ilmoittaa yhteydenkatkaisussa oman lämpötilansa olevan tyhjä merkkijono.

Qt-koodissa saapuvan viestin käsittely on toteutettu funktiossa rivillä 102. Tässä funktiossa ohjelma tarkistaa saako se viestin lämpötila- tai kosteusaiheeseen. Näissä tapauksissa ohjelma päivittää käyttöliittymässä olevat numerotaulut oikeisiin arvoihin. Ohjelman graafinen käyttöliittymä on esitelty kuvassa 12.



***Kuva 12**. Graafinen käyttöliittymä*

Käyttöliittymästä löytyy vastaavasti kaksi painiketta, joiden funktiot löytyvät riveiltä 154 ja 171. Mikäli käyttäjä painaa kuvasta käyttöliittymästä löytyvää Tilaa-painiketta, ohjelma lähettää anturiohjelmalle aktivointiviestin. Aktivointiviesti lähetetään palvelun laatutasolla 1, sillä sen perillemeno on hyvin oleellista. Kun anturiohjelma saa aktivointiviestin, se alkaa lähettämään anturidataa palvelun laatutasolla 0 rivien 81-85 mukaisesti. Vastaavasti kun käyttäjä painaa Lopeta Tilaus- painiketta, anturidatan lähettäminen keskeytetään. Anturiohjelman toiminta aktivointitilanteissa löytyy koodiriviltä 46 alkavassa funktiossa. Käyttöliittymä on toteutettu siten, että siihen voidaan tarvittaessa lisätä välilehtiä, kun järjestelmää laajennetaan. Tällä hetkellä siihen on toteutettu vain tämän kokonaisuuden välilehti.

4. YHTEENVETO

Tämän päivän automaatiossa tiedonsiirron nopeus, turvallisuus ja järjestelmän kevyt toiminta ovat tärkeässä asemassa. Tämän työn tarkoitus oli tutkia MQTT-protokollan soveltuvuutta esineiden Internet -protokollaksi. Työ koostui protokollaa tutkivasta teoriaosuudesta sekä pienestä käytännön sovelluksesta.

Teoriaosuutta lähdettiin tutkimaan syistä, miksi protokolla on luotu ja mitä tavoitteita sille on annettu. Näiden perusteella tutkittiin protokollaan liittyviä perusperiaatteita kuten julkaisija/tilaaja-periaatetta, aihekäsitteitä sekä palvelun laatutasoa. Salaus ja turvallisuus nousivat myös hyvin tärkeiksi käsittelykohteiksi. Käytännön sovelluksesta oli alun alkaen tavoite tehdä pienestä koostaan huolimatta tietoturvallinen ja stabiili järjestelmä ja siinä myös onnistuttiin. Avoimen lähdekoodin ohjelmistot sekä kirjastot ovat luoneet perusteet tämänkaltaisille projekteille sekä tiedonsiirrolle. Omien asiakasohjelmien kirjoittaminen avoimeen lähdekoodiin perustuvista kirjastoista onnistui hyvin ja samalla voitiin todeta MQTT-protokollan ohjelmointikielituki kahdella eri ohjelmointikielellä.

Työn teoriaosuudessa tuli ilmi, että MQTT-protokollan perustana on sen stabiili käyttäytyminen heikoilla ja epävakailla Internet-yhteyksillä. Asiakasohjelma pystyy halutessaan asettamaan oman tilansa, jopa yhteydenkatkaisun jälkeen, mikä on tehokas tapa kontrolloida järjestelmän toimintaa helposti katkeavilla yhteyksillä. Tavoitteiksi protokollalle sen luomisvaiheessa oli myös asetettu, että tiedonsiirto ei saa riippua datan tyypistä. Tämä on toteutettu protokollassa siten, että kaikki lähetettävä data on bittimuodossa ja sen vastaanottajien on itse huolehdittava sen oikeaoppisesta hyödyntämisestä. Lähetettävien pakettien priorisointi oli myös tärkeässä asemassa protokollan kehitysvaiheessa. Priorisoinnilla voidaan antaa jokaiselle MQTT-paketille sen sisältöä vastaava tärkeystaso eli QoS-taso. Sen avulla voidaan erotella erityisen tärkeät paketit vähemmän tärkeistä.

Näiden protokollalle luotujen vaatimusten perusteella voidaan arvioida MQTT:n soveltuvuutta IoT-järjestelmiin. Protokolla soveltuu ohjelmointikielituen, laajan käyttäjäkannan sekä luotettavuutensa ansiosta hyvin esineiden internet -protokollaksi. Siihen sisäänrakennetut ominaisuudet luovat perustan tehokkaalle, mutta yksinkertaiselle IoT-tiedonsiirrolle.

LÄHTEET

[1] F. Mattern ja C. Floerkemeier, ”From the Internet of Computers to the Internet of Things”, in *From Active Data Management to Event-Based Systems and More: Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday*, K. Sachs, I. Petrov, and P. Guerrero, Toim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 242–259.

[2] ”MQTT Essentials - All Core Concepts explained", HiveMQ. [Verkossa]. Saatavissa: https://www.hivemq.com/mqtt-essentials/. [Viitattu: 15-loka-2019].

[3] ”Building Facebook Messenger”, *Facebook*. [Verkossa]. Saatavissa: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920. [Viitattu: 15-loka-2019].

[4] A. Nipper, ”MQTT’s role as an IoT message transport”, *Control Engineering; Barrington*, vsk. 66, nro 1, ss. 20–21, January 2019.

[5] G. C. Hillar, *MQTT essentials: a lightweight IoT protocol : the preferred IoT publish-subscribe lightweight messaging protocol*, 1st p. PACKT Publishing, 2017.

[6] ”MQTT System Topics Documentation”, *GitHub*. [Verkossa]. Saatavissa: https://github.com/mqtt/mqtt.github.io. [Viitattu: 16-loka-2019].

[7] ”MQTT Version 3.1.1 - OASIS Standard Documentation”, *OASIS*. [Verkossa]. Saatavissa: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718033. [Viitattu: 16-loka-2019].

[8] ”Publishing over the MQTT bridge | Cloud IoT Core Documentation”, *Google Cloud*. [Verkossa]. Saatavissa: https://cloud.google.com/iot/docs/how-tos/mqtt-bridge. [Viitattu: 16-loka-2019].

[9] ”Which protocols does RabbitMQ support?”, *RabbitMQ*. [Verkossa]. Saatavissa: https://www.rabbitmq.com/protocols.html. [Viitattu: 21-marras-2019].

[10] ”MQTT Brokers/Servers and Hosting Guide”, *Steve Cope*. [Verkossa]. Saatavissa: http://www.steves-internet-guide.com/mqtt-hosting-brokers-and-servers/. [Viitattu: 16-loka-2019].

[11] ”TLS/SSL - MQTT Security Fundamentals”, *HiveMQ*. [Verkossa]. Saatavissa: https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/. [Viitattu: 17-loka-2019].

[12] W. J. Buchanan, *Cryptography*. Aalborg, DENMARK: River Publishers, 2017.

[13] ”Symmetric vs. Asymmetric Ciphers”, *IT Pro*, 16-loka-2006. [Verkossa]. Saatavissa: https://www.itprotoday.com/strategy/symmetric-vs-asymmetric-ciphers. [Viitattu: 17-loka-2019].

[14] J. R. Vacca, *Computer and Information Security Handbook*, vsk. 2nd ed. Amsterdam: Morgan Kaufmann, 2013.

[15] ”Configuring and Testing Mosquitto MQTT Topic Restrictions”, *Steve Cope*. [Verkossa]. Saatavissa: http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/. [Viitattu: 21-marras-2019].

LIITE A: KÄYTTÖLIITTYMÄN ASIAKASOHJELMA

File: mainwindow.h

```
001: #ifndef MAINWINDOW_H
002: #define MAINWINDOW_H
003: /**
004: Tämän ohjelman tarkoituksena on toimia MQTT-asiakasohjelmana ja
005: kodin automaatiojärjestelmän
006: ohjauskeskuksena. Se sisältää käyttöliittymän, josta käyttäjä
007: voi katsoa eri osajärjestelmän osia välilehdiltä. Tässä vaiheessa
008: ohjelmaan on toteutettu vain Sää-välilehti, joka toimii MQTT-protokollan
009: välityksellä.
010:
011: Ohjelmassa käytetään Eqmx:n luomaa MQTT-luokkaa Qt-ohjelmointiympäristöi-
hin
012: Saatavissa: https://github.com/emqx/qmqtt
013:
014: **/
015: #include <QMainWindow>
016: #include <QDebug>
017: #include <QtCharts/QLineSeries>
018: #include <QtCharts/QChart>
019: #include <QtCharts/QChartView>
020: #include <QtCharts/QValueAxis>
021: #include <QDateTime>
022: #include <QTimer>
023: #include "qmqtt.h"
024: #include <QSslConfiguration>
025: namespace Ui {
026: class MainWindow;
027: }
028:
029: class MainWindow : public QMainWindow
030: {
031:     Q_OBJECT
032:
033: public:
034:     /**
035:      * @brief MainWindow luokka on ohjelman pääluokka,
036:      *        joka rakennetaan ohjelman käynnistyessä
037:      * @param Tälle luokalle ei aseteta parent muuttujaa.
038:      */
039:     explicit MainWindow(QWidget *parent = nullptr);
040:     ~MainWindow();
041:     /**
042:      * @brief CreateTemperatureChart funktiolla luodaan
043:      *        käyttöliittymässä näkyvä yhden vuorokauden
044:      *        lämpötilakäyrä.
045:      *
046:      */
047:     void CreateTemperatureChart();
048:     /**
049:      * @brief initializeMQTTConnection on funktio, jolla
050:      *        luodaan salattu yhteys MQTT-välittäjään.
051:      *
```

```
052:      */
053:     void initializeMQTTConnection();
054:     /**
055:      * @brief handleMessage-funktiota kutsutaan kun tämä
056:      *        asiakasohjelma saa MQTT-viestin johonkin sen
057:      *        tilaamaan aiheeseen.
058:      *
059:      * @param message on parametri joka sisältää tiedot
060:      *        saapuneesta viestistä.
061:      */
062:     void handleMessage(const QMQTT::Message &message);
063: public slots:
064:     /**
065:      * @brief slotTimerExpired on apufunktio, jolla päivitetään
066:      *        lämpötilakäyrää ajastfunktiolla käsitellään
067:      *        viestitimen avustuksella.
068:      */
069:     void slotTimerExpired();
070:
071: private slots:
072:     /**
073:      * @brief on_connectButton_clicked on funktio, jota kutsutaan
074:      *        kun käyttäjä painaa Tilaa-painiketta käyttöliittymässä.
075:      */
076:     void on_connectButton_clicked();
077:     /**
078:      * @brief on_disconnectButton_clicked on funktio, jota kutsutaan
079:      *        kun käyttäjä painaa Lopeta Tilaus-painiketta
080:      *        käyttöliitymässä
081:      */
082:     void on_disconnectButton_clicked();
083:
084: private:
085:     /**
086:      * @brief ui on tämän MainWindow ikkuna-luokan
087:      *        käyttöliittymäluokka joka sisältää tiedot
088:      *        käyttöliittymässä olevista osakomponenteista
089:      *        kuten painikkeista ja tekstikentistä.
090:      *
091:      */
092:     Ui::MainWindow *ui;
093:     /**
094:      * @brief m_series on QLineSeries-luokan instanssi,
095:      *        joka sisältää viimeisen 24 tunnin säätiedot
096:      *        int-arvoina.
097:      */
098:     QtCharts::QLineSeries *m_series;
099:     /**
100:      * @brief m_chart on taulukko, johon on lisätty
101:      *        24 tunnin m_series muuttuvassa oleva
102:      *        lämpötilasarja.
103:      */
104:     QtCharts::QChart *m_chart;
105:     /**
106:      * @brief m_chartView on taulukkonäkymä, johon
107:      *       m_chart taulukko on lisätty.
108:      */
109:     QtCharts::QChartView *m_chartView;
```

```
110:     /**
111:      * @brief m_currentTemperature muuttuja sisältää
112:      *        sen hetkisen lämpötilan int-muodossa.
113:      */
114:     int m_currentTemperature;
115:     /**
116:      * @brief m_lastUpdated sisältää luvun 0-24
117:      *        sen mukaan milloin lämpötilakäyrä
118:      *        on päivitetty viimeksi
119:      */
120:     int m_lastUpdated;
121:     /**
122:      * @brief m_launchUpdated bool-muotoinen apumuuttuja,
123:      *        jota käytetään lämpötilakäyrän ensimmäisessä
124:      *         päivityksessä
125:      *
126:      */
127:     bool m_launchUpdated;
128:     /**
129:      * @brief m_client on MQTT-asiakasluokka, jonka avulla
130:      *        voidaan kommunikoida MQTT-protokollalla
131:      */
132:     QMQTT::Client *m_client;
133:
134:
135: };
136:
137: #endif // MAINWINDOW_H
138:
```

File: mainwindow.cpp

```
001: #include "mainwindow.h"
002: #include "ui_mainwindow.h"
003:
004: MainWindow::MainWindow(QWidget *parent) :
005:     QMainWindow(parent),
006:     ui(new Ui::MainWindow)
007: {
008:
009:     ui->setupUi(this);
010:     // Alustetaan ohjelma ja sen jäsenmuuttujat
011:     this->CreateTemperatureChart();
012:     this->initializeMQTTConnection();
013:     m_launchUpdated = false;
014:     m_currentTemperature = 0;
015:     m_lastUpdated =  QDateTime::currentDateTime().time().hour();
016:     //Ajastin lämpötilakäyrän päivitystä varten
017:     QTimer *m_timer = new QTimer(this);
018:     connect(m_timer, SIGNAL(timeout()), this, SLOT(slotTimerExpired()));
019:     m_timer->start(10000);
020:     // Lambda-funktio jota kutsutaan kun asiakasolio m_client saa MQTT-
viestin
021:     connect(m_client, &QMQTT::Client::received, this, [this](const
QMQTT::Message &message) {
022:         this->handleMessage(message);
023:     });
```

```
024:     //Lambda-funktio. jota kutsutaan kun asiakasoolio m_client yhteys vä-
littäjään muodostuu.
025:     connect(m_client, &QMQTT::Client::connected, this, [this]() {
026:         qDebug() << "Välittäjä yhdistetty";
027:     });
028:     //Lambda-funktio. jota kutsutaan kun asiakasoolio m_client yhteys vä-
littäjään katkeaa.
029:     connect(m_client, &QMQTT::Client::disconnected, this, [this]() {
030:         qDebug() << "Yhteys välittäjään katkennut";
031:     });
032: }
033:
034: MainWindow::~MainWindow()
035: {
036:     delete ui;
037: }
038: void MainWindow::CreateTemperatureChart()
039: {
040:     m_series= new QtCharts::QLineSeries();
041:
042:     QPen pen1;
043:     pen1.setWidth(2);
044:     pen1.setColor(Qt::red);
045:     //Asetetaan käyrälle sopiva väri ja paksuus
046:     m_series->setPen(pen1);
047:     //Luodaan lista, jossa 24 kappaletta 0-asteen lämpötiloja
048:     for(int i = 1; i < 25; i++) {
049:         m_series->append(i,0);
050:     }
051:     m_chart = new QtCharts::QChart();
052:     //Lisätään luotu sarja tilasto-olioon
053:     m_chart->addSeries(m_series);
054:     m_chart->legend()->hide();
055:     //Luodaan ja asetetaan sopivat akselit käyrälle
056:     auto axisX_ = new QtCharts::QValueAxis();
057:     axisX_->setTickCount(12);
058:     axisX_->setLabelFormat("%.0f");
059:     axisX_->setTickInterval(1);
060:     axisX_->setTitleText("Kello");
061:     m_chart->addAxis(axisX_, Qt::AlignBottom);
062:     m_series->attachAxis(axisX_);
063:     auto axisY_ = new QtCharts::QValueAxis();
064:     axisY_->setMin(-15);
065:     axisY_->setMax(15);
066:     axisY_->setLabelFormat("%i");
067:     axisY_->setTitleText("Lämpötila");
068:     m_chart->addAxis(axisY_, Qt::AlignLeft);
069:     m_series->attachAxis(axisY_);
070:     m_chartView = new QtCharts::QChartView(m_chart);
071:     m_chartView->setRenderHint(QPainter::Antialiasing);
072:     //Lisätään luotu kokonaisuus käyttöliittymään
073:     ui->ChartLayout->addWidget(m_chartView);
074:
075:
076: }
077:
078: void MainWindow::initializeMQTTConnection()
079: {
```

```
080:     //Luodaan TSL/SSL-konfiguraatio ja lisätään siihen
081:     //luotu sertifikaatti.
082:     QSslConfiguration sslConfig = QSslConfiguration::defaultConfigura-
tion();
083:     QFile file("ca.crt");
084:     file.open(QIODevice::ReadOnly);
085:     QByteArray data = file.readAll();
086:     QSslCertificate ca_cert(data);
087:     QList<QSslCertificate> certlist;
088:     certlist.append(ca_cert);
089:     sslConfig.setCaCertificates(certlist);
090:     //Otetaan salattu yhteys välittäjään ja asetetaan alkuarvot
091:     m_client = new QMQTT::Client("192.168.0.101", 8883, sslConfig);
092:     m_client->setClientId("TempScreen");
093:     m_client->setUsername("user");
094:     m_client->setPassword("password");
095:     //Määritetään viimeiseksi tahdoksi aihe ja arvo
096:     m_client->setWillTopic("koti/ulko/anturit/status");
097:     m_client->setWillMessage("OFF");
098:     m_client->connectToHost();
099:
100: }
101:
102: void MainWindow::handleMessage(const QMQTT::Message &message)
103: {
104:     qDebug() <<"Viesti saapunut aiheeseen: " +  message.topic();
105:     if (message.topic() == "koti/ulko/lampotila"){
106:
107:         //Jos viesti tulee lampotila-aiheesseen se näytetään käyttöliit-
tymän
108:         //lämpötilan LCD-näytössä. Samalla m_currentTemperature päivite-
tään
109:         ui->tempLCD->display(QString::fromUtf8(message.payload()) + "C");
110:         m_currentTemperature = QString::fromUtf8(message.pay-
load()).split(".")[0].toInt();
111:         ui->timeLabel->setText(QDateTime::current-
DateTime().time().toString());
112:     }
113:     else if (message.topic() == "koti/ulko/kosteus") {
114:         //Jos viesti tulee lampotila-aiheesseen se näytetään käyttöliit-
tymän
115:         //kosteuden LCD-näytössä.
116:         ui->humidLCD->display(QString::fromUtf8(message.payload()));
117:         ui->timeLabel->setText(QDateTime::current-
DateTime().time().toString());
118:     }
119:
120: }
121:
122: void MainWindow::slotTimerExpired()
123: {
124:     int toBeUpdated = QDateTime::currentDateTime().time().hour();
125:
126:     if (toBeUpdated == 0 && m_lastUpdated != 0) {
127:         //Spesiaalitapaus kun kello lyö 24 eli 0, päivitetään indeksissä
128:         // 23 oleva arvo oikeaan lämpötilaan.
129:         m_series->remove(23);
130:         m_series->insert(23, QPointF(24,m_currentTemperature));
```

```
131:         m_chartView->repaint();
132:         m_lastUpdated = 0;
133:         return;
134:     }
135:     if (!m_launchUpdated) {
136:         //Ensimmäinen arvo joka asetetaan ohjelman käynnistyessä.
137:         m_series->remove(toBeUpdated - 1);
138:         m_series->insert(toBeUpdated - 1, QPointF(toBeUpdated,m_cur-
rentTemperature));
139:         m_chartView->repaint();
140:         m_launchUpdated = true;
141:     }
142:     if (toBeUpdated != m_lastUpdated) {
143:         //Mikäli viimeksi päivitetty kellonajan arvo käyrässä on eri
144:         //kuin tämänhetkinen, päivitetään uusi arvo uuteen kohtaan.
145:         m_series->remove(toBeUpdated - 1);
146:         m_series->insert(toBeUpdated - 1, QPointF(toBeUpdated,m_cur-
rentTemperature));
147:         m_chartView->repaint();
148:         m_lastUpdated = toBeUpdated;
149:     }
150: }
151:
152:
153:
154: void MainWindow::on_connectButton_clicked()
155: {
156:     //Käynnistetään anturien lähetys ON-komennolla
157:     QString topic = "koti/ulko/anturit/status";
158:     QString value = "ON";
159:     QMQTT::Message *msg = new QMQTT::Message();
160:     msg->setTopic(topic);
161:     msg->setPayload(value.toUtf8());
162:     m_client->publish(*msg);
163:     //Tilataan antureiden arvot
164:     QString topic1 = "koti/ulko/lampotila";
165:     m_client->subscribe(topic1);
166:     QString topic2 = "koti/ulko/kosteus";
167:     m_client->subscribe(topic2);
168:
169: }
170:
171: void MainWindow::on_disconnectButton_clicked()
172: {
173:     //Sammutetaan anturien lähetys OFF-komennolla
174:     QString topic = "koti/ulko/anturit/status";
175:     QString value = "OFF";
176:     QMQTT::Message *msg = new QMQTT::Message();
177:     msg->setTopic(topic);
178:     msg->setPayload(value.toUtf8());
179:     m_client->publish(*msg);
180:     //Lopetetaan aiheiden tilaus
181:     QString topic1 = "koti/ulko/lampotila";
182:     m_client->unsubscribe(topic1);
183:     QString topic2 = "koti/ulko/kosteus";
184:     m_client->unsubscribe(topic2);
185:
186: }
```

LIITE B: ANTURIN ASIAKASOHJELMA

File: tempsensor.py

```
#!/usr/bin/ python3
'''
Tämän ohjelman tarkoituksena on toimia MQTT-asiakasohjelmana ja lämpötilasensorina.
Ohjelma muodostaa yhteyden välittäjään ja odottaa, että saa viestin aiheeseen
"koti/ulko/anturit/status", jonka kuormana on merkkijono "ON".
Tämän johdosta ohjelma alkaa lähettämään anturidataa aiheisiin:
"koti/ulko/lampotila"
"koti/ulko/kosteus"

Datan lähetys päättyy mikäli ohjelma saa "OFF" viestin
edellä mainittuun status-aiheeseen

'''

import sys
import time
import Adafruit_DHT
import paho.mqtt.client as paho
import ssl
#Pinnit, joihin anturi on kytketty
sensor = 11
pin = 17

class TempSensorClient:
    #Rakentaja ottaa parametrinaan yhteyteen liittyvät tiedot
    def __init__(self, client_name ,broker_address, port, tsl_credentials):
        self.client_name = client_name
        self.broker_address = broker_address
        self.port = port
        self.tsl_credentials = tsl_credentials
        self.transmitting = False

    def initialize_connection(self):
        self.client = paho.Client()
        #Yhdistetään valmiin asiakasohjelman funktiot tämän ohjelman funktioihin
        self.client.on_message = self.on_message
        self.client.on_connect = self.on_connect
        self.client.on_publish = self.on_publish
        self.client.on_subscribe = self.on_subscribe
        #Asetetaan viimeinen tahdo
        self.client.will_set("koti/ulko/lampotila", "", qos=0, retain=False)
        self.client.tls_set("ca.crt", tls_version=ssl.PROTOCOL_TLSv1_2)
        self.client.connect(self.broker_address, self.port, 60)
        self.client.loop_start()
    #Funktio, jota kutsutaan kun ohjelma saa viestin johonkin sen tilaamaan aiheeseen
    def on_message(self, client, userdata, message):
```

```
47:         msg = str(message.payload.decode("utf-8"))
48:         topic = message.topic
49:         #Tarkastetaan onko viesti käsky alkaa lähettämään dataa
50:         if topic == "koti/ulko/anturit/status":
51:             if msg == "ON":
52:                 self.transmitting = True
53:                 print("Aloitetaan anturidatan lähetys")
54:             elif msg == "OFF":
55:                 self.transmitting = False
56:                 print("Lopetetaan anturidatan lähetys")
57:         else:
58:             print("Viesti saapunut: "+ msg + " aiheeseen: " + topic)
59:         pass
60:     def on_connect(self, client, userdata, flags, rc):
61:         print("Yhteys välittäjään muodostettu")
62:     def on_publish(self, client,userdata,result):
63:         print("Viesti julkaistu")
64:         pass
65:     def on_subscribe(self, client, userdata, mid, granted_qos):
66:         print("Aiheen tilaus onnistunut")
67:
68: #Asetetaan ohjelman alkuarvot
69: TSC = TempSensorClient("TempSensor", "192.168.0.101", 8883, None)
70: TSC.initialize_connection()
71: TSC.client.subscribe("koti/ulko/anturit/status",1)
72: temp_topic = "koti/ulko/lampotila"
73: humid_topic = "koti/ulko/kosteus"
74:
75: #Silmukka, jossa ohjelma lähettää dataa mikäli sitä on pyydetty.
76: while True:
77:     time.sleep(1)
78:     try:
79:         #Luetaan anturilta arvot.
80:         humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
81:         if TSC.transmitting:
82:             TSC.client.publish(temp_topic, payload=str(temperature),
qos=0, retain=False)
83:             print("Yritetään julkaista aiheeseen:" + temp_topic + " ar-
voa:"+ str(temperature))
84:             TSC.client.publish(humid_topic, payload=str(humidity), qos=0,
retain=False)
85:             print("Yritetään julkaista aiheeseen:" + humid_topic + " ar-
voa:"+ str(humidity))
86:     except KeyboardInterrupt:
87:         TSC.client.loop_stop()
88:         print("Lopetetaan ohjelman suoritus")
89:         sys.exit()
90:
```