VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Samuli Oja-Kaukola

DATAN LÄHETYS IOT-LAITTEELTA AWS-PIL-VIPALVELUUN

Tekniikka
2022

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Samuli Oja-Kaukola
Opinnäytetyön nimi	Datan lähetys IoT-laitteelta AWS-pilvipalveluun
Vuosi	2022
Kieli	suomi
Sivumäärä	55 + 5 liitettä
Ohjaaja	Jukka Matila

Tässä opinnäytetyössä luotiin IoT-järjestelmä, joka lähettää mitattuja lämpötila-arvoja AWS:ään. Työn tarkoituksena oli tutustua avain komponentteihin ja menetelmiin, joiden avulla IoT-järjestelmän ja AWS:än välinen yhteys saataisiin muodostettua.

IoT-järjestelmän mittaama lämpötila välitetään AWS-pilvipalvelimeen käyttäen MQTT-viestintäprotokollaa. Järjestelmälle luodaan matkapuhelinverkko HL7802 cellular-moduulia käyttäen, jonka avulla järjestelmä muodostaa yhteyden AWS:ään. Järjestelmä on kehitetty kustomoidulle prototyyppilevylle.

Opinnäytetyö toi minulle osaamista AWS-pilvipalvelin ympäristöön liittyen, sekä tutustutti minut matkapuhelinjärjestelmiin. Opinnäytetyön raportointivaiheessa ilmeni myös jatkokehitysideoita, joita järjestelmään voisi vielä lisätä.

Opinnäytetyön lopputuloksena, kustomoidulle prototyyppilevylle kehitettiin toimiva lämpötilaa mittaava IoT-järjestelmä, joka julkaisee dataa AWS-pilvipalveluun.

Avainsanat matkapuhelinverkot, AWS, MQTT-viestintäprotokolla, IoT

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Samuli Oja-Kaukola
Title	Sending Data from an IoT Device to the AWS Cloud Service
Year	2022
Language	Finnish
Pages	55 + 5 Appendices
Name of Supervisor	Jukka Matila

In this thesis, an IoT system was created that sends the measured temperature values to the AWS. The purpose of this work was to get acquainted with the key components and methods that would be used to establish a connection between the IoT system and AWS.

The temperature measured by the IoT system is transmitted to the AWS cloud server using the MQTT communication protocol. A cellular network is established for the system using the HL7802 cellular module, which allows the system to connect to the AWS. The system was developed for a custom prototype board.

The thesis brought me knowledge related to the AWS cloud server environment, as well as introduced me to mobile network systems. During the reporting phase of the thesis, further development ideas also emerged, which could be added to the system.

As a result of the thesis, a functional temperature measuring IoT system was developed for a custom prototype board, which publishes data to the AWS cloud service.

Keywords Cellular, AWS, MQTT communication protocol, IoT

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

LYHENTEET JA TERMIT

IoT	Internet of Things
AWS	Amazon Web Service
IDE	Integrated Development Environment (Kehitysympäristö)
MQTT	Message Queueing Telemetry Transport
LPWA	low-power wide-area (Pienitehoinen suuralueverkko)
LTE-M	Long Term Evolution for Machines
NB-IoT	Narrowband IoT
GNSS	Global Navigation Satellite System
USART	universal synchronous and asynchronous receiver-transmitter (sarjaliikenteen lähteys- ja vastaanottopiiri)
RTS	Ready to Send
CTS	Clear to Send
ESP	Encapsulating Security Payload (pakettivirtojen salaus)
APN	Access Point Name (yhdistyskäytävän nimi)
PDN	Public Data Network (julkinen tietoverkko)
Cellular	Matkapuhelinverkko
GSM	Global System for Mobile (digitaalinen matkapuhelinjärjestelmä)
2G	Second generation (toinen sukupolvi)

LTE	Long Term Evolution (laajakaistainen internet-yhteys)
MTSO	mobile telecommunications switching office
Taajuusspektri	signaalin sisältämä taajuusalue
TCP	Transmission Control Protocol (tietoliikenne protokolla)
TLS	Trasport Layer Security (kuljetuskerroksen suojaus)
IP	Internet Protocol (verkkokerroksen protokolla)
Soketti	Sokettia käytetään tunnistautuessaan muille ulkoisille isäntälaitteille soket-osoitteen perusteella.
Lippu	Muuttuja, joka indikoi tilan muutosta.
RAM	Random Access Memory (keskusmuisti)
ROM	Read-Only Memory (lukumuisti)
API	Application programming interface (ohjelmiston rajapinta)
AD-muunnin	ADC (Analog to Digital Converter)
IT	Information technology (tietotekniikka)
IssA	Infrastructure as a Service (palvelimien ja palvelinsalien ulkoistaminen)
Paas	Platform as a Service (palvelualustanulkoistaminen)
SaaS	Software as a Service (ohjelmiston jakelumalli)

LIITELUETTELO

1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli kehittää reaaliaikainen IoT-järjestelmä, joka mittaa lämpötilaa ja lähettää lämpötila-arvoja AWS-pilvipalveluun ja tallentaa ne AmazonWebService:n tarjoamaan DynamoDB-tietokantaan. Jotta mitatut lämpötila-arvot saadaan lähetettyä AWS-pilvipalveluun, on pystytettävä matkapuhelinverkkoyhteys, jonka avulla voidaan muodostaa yhteys laitteen ja pilvipalvelun välille. Opinnäytetyössä käydään läpi myös matkapuhelinverkkojen, MQTT-protokollan, FreeRTOS-reaaliaikakäyttöjärjestelmän ja AWS-pilvipalvelun teoreettinen toiminnallisuus, sekä niiden osat projektin kokonaisuutta.

Opinnäytetyöprojekti on kehitetty kustomoidulle prototyyppilevylle, mikä toi haastavuutta kirjastojen ja toimintojen implementointiin. Prototyyppilevy sisälsi HL7802-cellular-moduulin, jonka avulla matkapuhelinverkko voitiin pystyttää. Cellular-moduulia ohjattiin käyttämällä AT-komentoja.

Lämpötilan mittaamiseen käytettiin PT-1000-vastuslämpötila-anturia, joka mittaa ympäristön lämpötilan suhteessa resistanssiin. MQTT-viestintä muodostettiin käyttämällä coreMQTT-kirjastoa. Prototyyppilevyllä olevaa STM32F479IIT6 mikroprosessoria ohjelmoitiin C-kielellä käyttämällä STM32CubeIDE-kehitysympäristöä.

Työn toimeksiantajana toimi Xedi Ky, joka myy tilaustyönä asiakkaille suunniteltuja sulautettujen järjestelmien ohjelmistoja. Xedi Ky on perustettu vuonna 2019. Opinnäytetyöprojekti aloitettiin tammikuussa 2022 ja se valmistui maaliskuussa 2022.

2 MATKAPUHELINVERKKO

Tässä luvussa käydään läpi matkapuhelinverkosto toiminnallisuutta.

2.1 Sukupolvet

Matkapuhelinverkostot ovat kehittyneet ajan myötä ja jatkavat kehittymistä. Verkostojen kehittymisessä on tapahtunut suuria harppauksia, ja tämän johdosta uusia laitteita ja palveluita on voinut syntyä. Jokaisen sukupolven tarkoituksena on ollut tarjota uutta toiminnallisuutta ja kehittää matkapuhelinverkoston kokonaisuutta. Kehitettyjä sukupolvia tähän päivään mennessä ovat 1G, 2G, 3G, 4G ja 5G, joista jokainen on toistaan suurempi harppaus verkkojen kehityksessä. Projektissa käytetty HL7802 cellular-moduuli ohjattiin käyttämään GSM-verkkoyhteyttä.[8, 26]

2.2 2G-verkosto

1G verkosto oli käytössä vuoteen 1991 asti, jonka jälkeen se korvattiin toisen sukupolven (2G) verkolla. Uusi 2G-verkosto ei enää toiminut analogisella radiosignaalilla, vaan digitaalisella signaalilla. Tämä nostatti huomattavasti salauksen turvallisuutta, lähetettävän signaalin laatua, signaalin kantamaa sekä verkoston kapasiteettia. 2G-verkon tavoitteena oli saavuttaa GSM (Global System for Mobile communication) standardi.

Nämä standardit toivat uusia ominaisuuksia käyttäjille, kuten tekstiviestien (SMS), multimediaviestien (MMS) lähettämisen ja vastaanottamisen, sekä WAP (Wireless Application Protocol) internetprotokollan, joka salli pääsyn matkapuhelimella tiettyyn Internet-sisältöön. Vuonna 1997 2G-verkkon tarjoamalla GPRS-tiedonsiirtoyhteydellä voitiin lähettää, sekä vastaanottaa sähköposteja. Toisen sukupolven kehittymisen puolessa välissä (2.5G) GPRS päivittyi EDGE (Enhanced Data rates for Global Evolution) teknologiaan, mikä nostatti yhteysnopeuksia perustuen uuteen pakettikytkentä (Packet switching) tiedonsiirtomenetelmään.[8, 26]

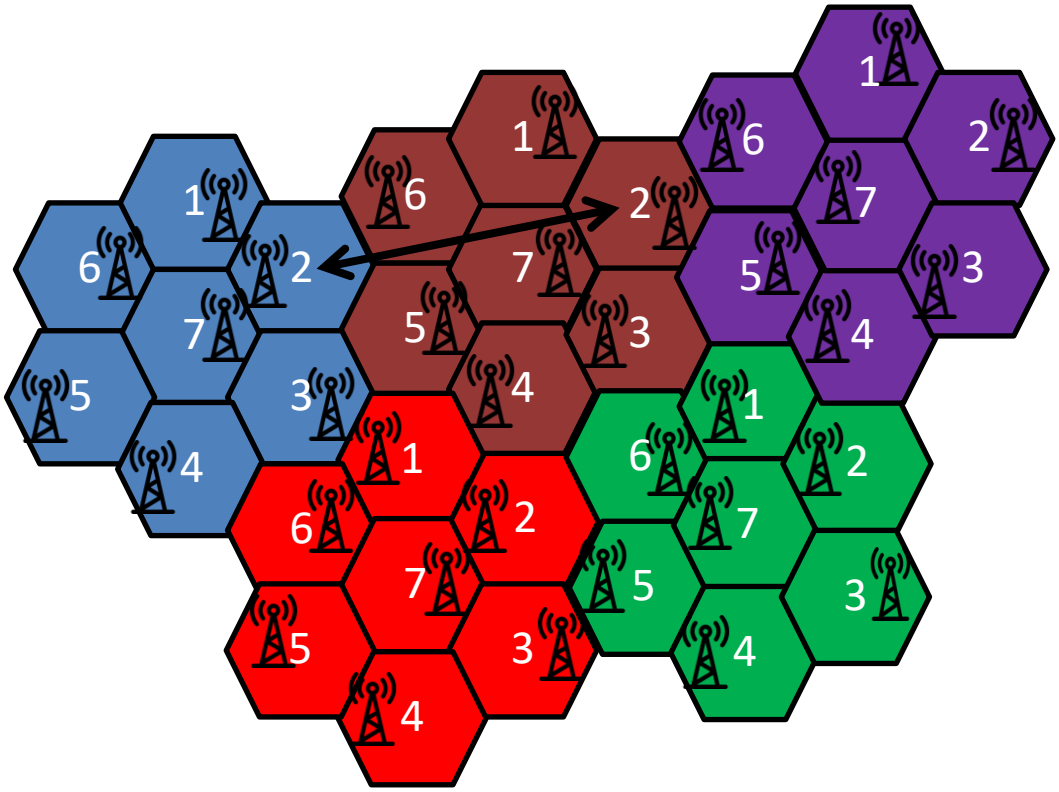
2.3 Yleistä matkapuhelinverkosta

Matkapuhelinverkkosysteemit ovat laajassa käytössä nykypäivänä, ja niiden tehtävänä on tarjota tehokkaasti saatavilla olevia taajuusspektrejä. Nykypäivänä, kun mobiililaitteita on käytössä jopa miljardi ympäri maailmaa, on taajuuksien uudelleenkäyttö välttämätöntä ilman, että mobiililaitteet häiritsevät toisiaan.

Matkapuhelinverkosto mahdollistaa puhelinyhteyden jakamalla palvelinalueen soluasemiin. Jokaisella soluryhmittymällä on oma lähetin, joka hallitsee, lähettää ja vastaanottaa liikennettä omilla maantieteellisellä alueellaan olveilta mobiililaitteilta matkapuhelinkeskukseen. Soluasemat hyödyntävät puhelinmastoja ja niiden antenneja tarjotakseen yhteyden etäiseen kytkimeen, jota kutsutaan matkaviestinvälitystoimistoksi (MTSO). MTSO:n tehtävinä on yhdistää puhelinyhteyksiä käyttäjille, vaihtaa puhelut solujen välillä matkapuhelimen ylittäessä sen hetkisen soluaseman rajan sekä varmentaa yhteyden luojan ennen yhdistämistä.

Matkapuhelinverkostoa esitetään usein kuusikulmiona, jotka ovat liitettynä toisiinsa. Matkapuhelinverkosto muodostuu soluista ja nimi ”Cellular” syntyy englanninkielisestä sanasta ”cell” eli solu. Kuusikulmainen muoto ei kuitenkaan todellisuudessa kuvaa verkoston kantavuutta.

On otettava huomioon ympäristö, kuten vuoret, rakennukset ja muut vastaavat rakennelmat, jotka voivat vaikuttaa signaalin kantavauuteen. Solun päättymisrajaa on siis hyvin vaikea määritellä. Signaalin vahvuus heikkenee huomattavasti, mitä kauemmaksi solun vastaanotinta edetään. Myös mobiililaitteet itsessään eroavat siinä, kuinka vahvana ne pystyvät signaalia pitämään.[3, 5]



Kuva 1. Matkapuhelinverkon soluasemaverkosto.

2.4 Taajuuden uudelleenkäyttö

Matkapuhelinverkkojärjestelmä pyrkii hyödyntämään käytettävissä olevia kanavia tehokkaasti käyttämällä pienitehoisia radiolähettimiä. Jokaisella radiolähettimellä on tietty kuuluvuusalue. Nämä lähettimet mahdollistavat taajuuksien uudelleenkäytön lisätäkseen huomattavasti palveltavien käyttäjien määrää. Kuva 1 havainnollistaa taajuuden uudelleenkäytön soluasemasta toiseen. Pienitehoiset matkaviestimet ja radiolaitteet jokaisessa soluasemassa mahdollistavat samojen radiotaajuuksien uudelleenkäytön eri soluissa, joka moninkertaistaa yhteyskapasiteetin, aiheuttamatta minkäänlaisia häiriöitä. Kuvassa nähdään myös, kuinka vastaanottimet ovat jaettu soluverkoston eri soluryhmittymille.[4]

2.5 Soluryhmittymät

Matkapuhelinverkosto muodostuu soluryhmittymistä. Soluryhmittymien infrastruktuuria suunnitellessa on otettava huomioon vierekkäiset soluryhmittymät. Vierekkäisille soluryhmittymille allokoidaan eri taajuuskaistoja ja kanavia siten, että solut voivat olla hieman toistensa päällä ilman häiriöitä. Tämä minimoi taajuuksien välisiä häiriöitä. Yleisintä on, että soluryhmittymä koostuu seitsemästä solusta, mutta on olemassa kokoonpanoja, jotka ovat ristiriidassa seitsemänsoluisen ryhmittymän kanssa.

Soluverkon konfiguroinnilla pyritään minimoimaan häiriöitä saman taajuuden omaavien solujen välillä. Mitä suurempi solujen määrä ryhmittymässä on, sitä suurempi etäisyys samojen taajuuksien omaavien solujen välillä on. Käytännössä paras vaihtoehto olisi luoda ryhmittymä, missä on suuri määrä soluja. Tämä ei kuitenkaan ole mahdollista, sillä saatavilla on vain rajoitettu määrä kanavia. Tämä tarkoittaa sitä, että mitä suurempi solujen määrä ryhmittymässä on, sitä pienempi on kunkin solun käytettävissä oleva lukumäärä, mikä taas vähentää kapasiteettia. Tämä tarkoittaa, että ryhmitelmän solujen lukumäärän, häiriötasojen ja kapasiteetin välillä on oltava tasapaino.

Solujen koko soluryhmittymissä mahdollistaa suuremman käyttäjä kapasiteetin, mutta solut vaatisivat yhä enemmän lähettimen vastaanottimia tai tukiasemia, ja tämä lisäisi huomattavasti lisäkustannuksia operaattorille. Eli esimerkiksi kaupunkialueilla vastaavasti, missä käyttäjiä on enemmän, asennetaan paljon pienitehoisia tukiasemia.[4]

3 MQTT

Tässä luvussa käsitellään yleistä MQTT-protokollan toiminnallisuutta ja protokollan historiaa. Lisäksi luvussa tutkitaan MQTT julkaise-tilaa-verkkoprotokollan arkkitehtuuria, sekä toiminnallisuutta.

3.1 MQTT-protokollalle yleistä

MQTT on M2M julkaise-tilaa arkkitehtuuriin perustuva viestintäprotokolla, joka on kevyt, julkinen ja helppo implementoida. MQTT:n implementoinnin helppous mahdollistaa protokollan käytön IoT-järjestelmien kehityksessä. Esimerkiksi kotiautomaatiojärjestelmät hyötyvät protokollasta suunnattomasti, sillä se mahdollistaa laitteiden välisen kommunikoinnin nettiyhteyden välityksellä varaamatta liikaa kaistanleveyttä tai muita applikaatioita. Ensisijaisesti protokolla on kehitetty yhdistämään kaistanleveyden ja tehonrajoittamia laitteita langattomien verkkojen kautta. MQTT-viestintäprotokollaa on käytetty vuodesta 1999 lähtien ja se on suunniteltu käyttämään TCP/IP-verkkosokettia tunnistautuakseen ulkoisille isäntälaitteille.

MQTT-järjestelmät pitävät julkaisijan ja tilaajan erillään toisistaan. Vastaanottaakseen -tai lähettääkseen viestejä tilaajan ja julkaisijan tulee tietää vain välittäjän isäntänimi tai IP-osoite. MQTT-järjestelmät kommunikoivat asynkronisesti eli osapuolet eivät ole ajallisesti toisistaan riippuvaisia. Suurella osalla MQTT-järjestelmistä on myös mahdollista välittää viestejä lähes reaaliajassa, mikäli sellaiselle on tarvetta. [15, 18, 24]

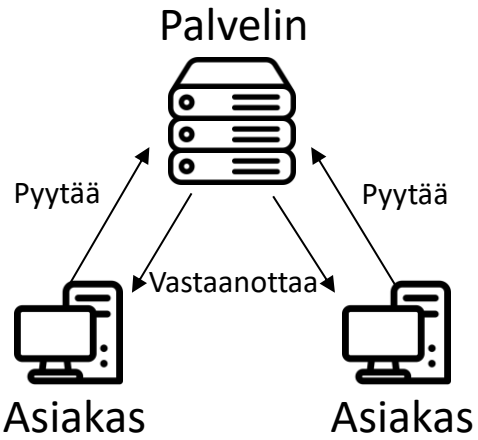
3.2 MQTT-protokollan historia

Ensimmäisen MQTT-viestintäprotokollan kehittivät Andy Stamford-Clark ja Arlen Nipper vuonna 1999. He määrittelivät, mitä protokollan tulisi sisältää tulevaisuudessa. Protokollan tuli olla helposti implementoitavissa, jotta sitä pystyttiin hyödyntämään mahdollisimman paljon ohjelmistokehityksessä. Protokollan tavoitteena oli myös parantaa vastaanotettavan datan laatua sekä sen oli oltava kevyt.

Protokollan tulisi vastaanottaa dataa sen formaatista tai lähteestä huolimatta, mutta prosessoida se silti tehokkaasti. Vuonna 2010 julkaistiin lisenssimaksuvapaa versio MQTT 3.1, josta lähtien protokolla oli kaikkien käytettävissä ilmaiseksi.[11, 15]

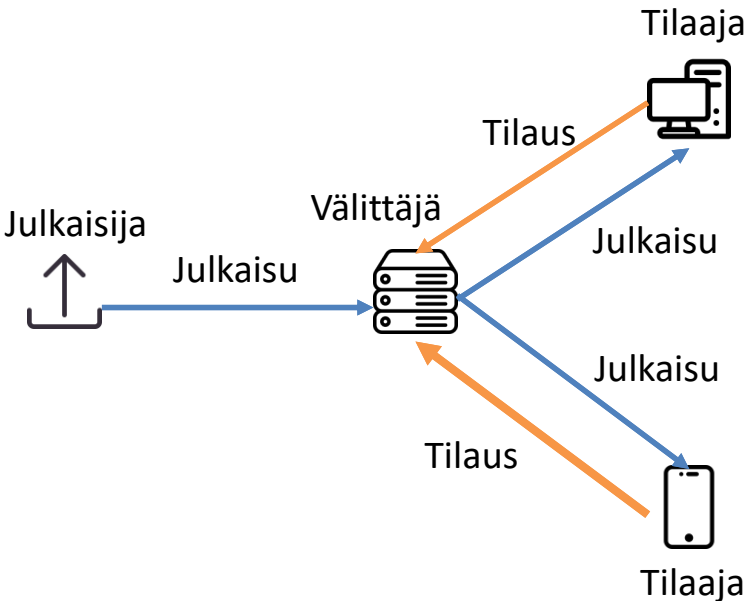
3.3 Julkaise-tilaa arkkitehtuuri

Julkaise-tilaa (publish-subscribe) arkkitehtuuri on vaihtoehtoinen malli perinteiselle asiakas-palvelinarkkitehtuurille.



Kuva 2. Asiakas-palvelin arkkitehtuuri.

Asiakas-palvelin arkkitehtuuri perustuu pyytää-vastaanottaa toiminallisuusperiaatteeseen. Kuva 2 havainnollistaa, kuinka asiakas lähettää pyynnön palvelimelle ja palvelin vastaa siihen. Asiakas-palvelin mallissa asiakas kommunikoi siis suoraa päätepisteen kanssa, kun taas julkaise-tilaa mallissa viestit kulkevat välittäjän kautta. Tällöin tilaaja, joka vastaanottaa viestin ei ole tietoinen viestin lähettäjästä, eli julkaisijasta.



Kuva 3. Julkaise-tilaa arkkitehtuuri.

Kuva 3 esittää, kuinka julkaise-tilaa arkkitehtuuri käytännössä toimii. Julkaisija julkaisee viestin, joka välittyy välittäjälle. Välittäjä vastaanottaa julkaisut ja tilaukset ja välittää julkaisut tilaajille. Julkaise-tilaa mallin välittäjän tärkein tehtävä on erotella julkaistut viestit ja välittää ne oikeille tilaajille. Välittäjä erottelee elementtejä, joita tilaaja ei tarvitse julkaisijoilta, kuten IP-osoitteet ja portit. Näin julkaisija tai tilaaja eivät kommunikoi suoranaisesti toistensa kanssa.[6, 11, 19]

3.3.1 Viestien suodatus

Välittimellä tapahtuva viestien suodatus on julkaise-tilaa mallin tärkeimpiä rooleja. Viestien suodattamiseen käytettäviä toimintatapoja on kolme erilaista. Aiheen perusteella toimiva suodatin, joka suodattaa viestin aiheen perusteella. Tilaaja ilmoittaa välittimelle tilaamansa aiheen ja aiheeseen liitetty viesti välittyy tilaajalle.

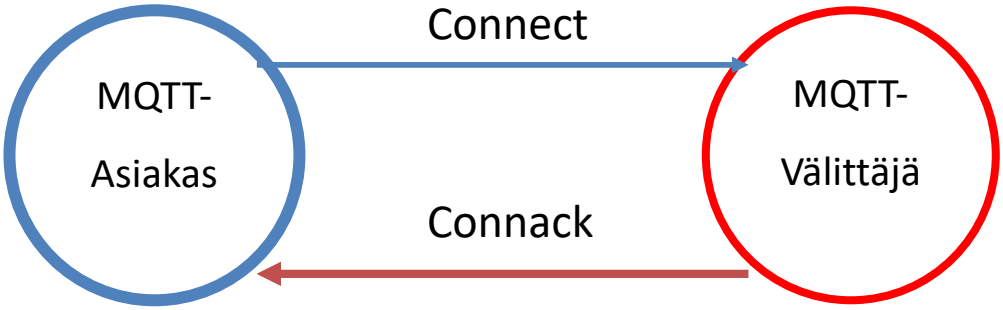
Toisena suodatintapana on sisällön perusteella toimiva suodatus. Sisällön perusteella toimiva suodatus suodattaa viestin sisällönsuodatuskielen perusteella. Tilaajat tilaavat heitä kiinnostavien viestien suodatuskyselyt ja välittäjä välittää viestin tilaajalle. Tämän suodatusmenetelmän heikkous on viestin salausmahdollisuudet, sillä viesti on tiedettävä etukäteen, jotta sen voi suodattaa.

Kolmantena suodatintapana on tyypin perusteella toimiva suodatus. Tätä suodatusmenetelmää käytetään, kun kyseessä on oliosuuntautunut kieli. Tällöin viestien sisällöt voidaan suodattaa tyypin tai luokan perusteella. Tilaaja tilaa tyyppiä tai luokkaa ja vastaanottaa koko sisällön mitä tyyppi tai luokka sisältää.[19]

3.4 MQTT-viestintä MQTT-asiakkaan ja MQTT-välittäjän välillä

MQTT-viestintäprotokollaa käyttävästä julkaisijasta tai tilaajasta käytetään käsitettä asiakas (client). MQTT-asiakas voi olla mikä tahansa laite, joka pystyy käsittelemään MQTT-palvelinta ja luomaan yhteyden välittäjään netin välityksellä. Jokaista laitetta, joka kommunikoi MQTT-viestintäprotokollan avulla TCP/IP-tietoliikenneprotokollapinon ylitse voidaan kutsua asiakkaaksi.

MQTT-protokolla perustuu TCP/IP toiminnallisuuteen. TCP/IP on tietoliikenneprotokollapino, joka mahdollistaa turvallisen ja tehokkaan tiedonsiirron verkossa. TCP täsmentää, kuinka sovellukset voivat luoda viestintäkanavia verkossa, ja miten viestit jaetaan pieniin paketteihin ennen lähetystä, sekä kuinka paketit kootaan oikeassa järjestyksessä pakettien päästyä perille. IP määrittää reitin ja varmistaa, että paketit pääsevät perille. MQTT-viestinnän mahdollistamiseksi molemmat asiakas ja välittäjä tarvitsevat TCP/IP-pinon.[16, 23]



Kuva 4. MQTT-asiakas muodostaa yhteyden MQTT-välittäjään.

Kuva 4 havainnollistaa, kuinka asiakkaan muodostaessa yhteyden välittäjän kanssa. Asiakas lähettää yhdistysviestinpaketin (connect) välittäjälle ja välittäjä vastaa yhteyden hyväksynnällä (connack). Kun yhteys on muodostettu, se pysyy niin kauan, kunnes erillinen katkaisuviestipaketti (disconnect) annetaan tai yhteys katkeaa. Connect-viestipaketin sisällön täytyy olla tietyssä formaatissa sillä väärin muotoillut viestit, joiden vastaanottamisessa kuluu liian kauan luokitellaan haitta-ohjelmiksi tai viesteiksi, jotka tukkivat liikennettä ja näin ollen hidastavat välittäjän toimintaa.

Connect-viestipaketin tulee pitää sisällään vähintään ClientID, Clean Session-lipun ja keepAlive-ajan. ClientID:n tulee olla asiakaskohtainen, koska tällä asiakas tunnistautuu MQTT-välittäjälle. Clean session-lippu kertoo välittäjälle, mikäli asiakas haluaa kestävän yhteyden. Jos connect-viestipaketti sisältää *cleanSession = false*, välittäjä säilyttää kaikki tilaukset ja viestit mitä asiakas ei ole vastaanottanut. Jos connect-viestipaketti sisältää *cleanSession = true*, välittäjä tyhjentää kaiken informaation edellisiltä yhteys kerroilta. Keepalive-aika määrittää ajan, kuinka monta sekuntia asiakas ja välittäjä pitävät yhteyden yllä toistensa välillä ilman, että he viestittelevät tai lähettävät PING-komentoja toisilleen.

Connect-viestipaketti voi myös halutessaan pitää sisällään käyttäjänimen ja salasanan, sekä LWT (Last Will and Testament) viestin. MQTT:n on mahdollista lähettää tunnistautumista varten asiakkaalle käyttäjänimen ja salasana. Näitä viestejä ei kuitenkaan ole salattu, joten se ei ole suositeltavaa. Monet MQTT-välittäjät voivat todentaa asiakkaat SSL (Secure Sockets Layer) salausprotokollan avulla. LWT-

viesti ilmoittaa muille käyttäjille, jos joku käyttäjistä menettää yhteyden yhtäkkisesti. Connect-viestipaketin sisältäessään LWT-viestin välittäjä voi yhteyden katketessa lähettää välittäjän viimeisen viestin.

Connack-viestipaketti, minkä välittäjä lähettää asiakkaalle connect-viestipaketin saatuaan sisältää sessionPresent-lipun ja returnCode-lipun. SessionPresent-lippu kertoo asiakkaalle, onko tälle jo olemassa vanha istunto eli onko edellinen sessio tyhjätty. ReturnCode-lippu kertoo asiakkaalle, onnistuiko yhteyden muodostaminen MQTT-välittäjän kanssa.[16]

3.5 Julkaisu MQTT-välittäjälle

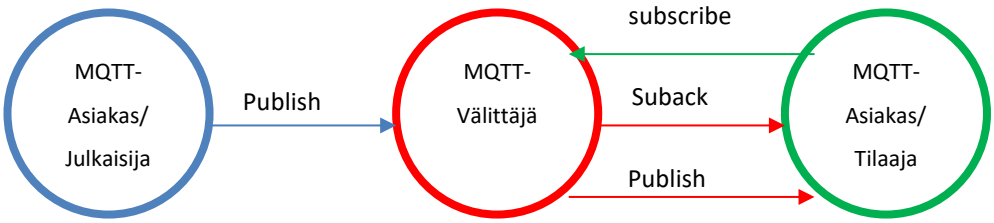
Tässä kappaleessa asiakas viittaa julkaisijaa. Asiakkaan muodostettua yhteyden välittäjän kanssa, asiakas on valmis julkaisemaan (Publish) viestejä välittäjälle. Jokainen julkaisu, minkä asiakas lähettää välittäjälle tulee sisältää aiheen, jotta julkaisu on mahdollista välittää tilaajalle.

Asiakkaan julkaisema MQTT-viestintäpaketti sisältää useita attribuutteja. Olennaisimmat ovat viestipaketintunnus (PacketID), aiheen nimi (Topicname) ja sisältö (Payload). PacketID identifioi viestipaketin kokonaisuuden, jotta se voi kulkea asiakkaan ja välittäjän välillä. Aiheen nimi, jota tarvitaan paketin vastaanottamiseen. Aihe on merkkijono, jonka rakenne on yleensä jaoteltu ”/” symbolilla. Näin tilausta on helpompi kuvailla tarkemmin. Payload-kohta pitää sisällään itse julkaisun sisällön, minkä julkaisija on lähettänyt. Julkaisun formaatin voi olla kuva, tekstiä kaikissa muodissa, salattua dataa tai binäärisesti mitä vain.

Asiakkaan julkaistua MQTT-viestintäpaketin välittäjälle välittäjä lukee julkaisun ja hyväksyy sen. Julkaisun hyväksyttyä välittäjä välittää julkaisun tilaajille, jotka ovat tilanneet kyseisen viestipaketin aiheen. Julkaisija ei saa minkäänlaista varmennetta, kun viestipaketti on saapunut tilaajalle.[17]

3.6 Tilaus MQTT-välittäjälle

Tässä kappaleessa asiakas viittaa tilaajaan. Asiakkaan tilatessaan välittäjää asiakas lähettää tilausviestipaketin (subscibe) välittäjälle. Tilauspaketti sisältää aiheet, joita asiakas haluaa tilata ja PacketID:n samaan tapaan, kuin julkaisussa lähetetyssä viestipaketissa.



Kuva 5. Julkaisu välittyy tilaajalle.

Kuva 5 havainnollistetaan, kuinka MQTT-asiakas/julkaisija julkaisee viestipaketin MQTT-välittäjälle. MQTT-asiakas/tilaaja lähettää tilausviestipaketin välittäjälle ja välittäjä varmentaa tilattavan aiheen olemassaolon, jonka jälkeen välittäjä lähettää tilauksen hyväksynnän (suback) asiakkaalle. Tämän jälkeen välittäjä välittää kaikki tilatun aiheen alla olevat julkaisut tilaajalle.[17]

3.6.1 Tilauksen lopettaminen

Tilauksen vastaanotettuaan asiakas lähettää tilauksen lopetuspaketin (unsubscribe) välittäjälle. Tämä paketti sisältää tilattujen aiheiden nimet, jotta välittäjä tietää poistaa tämän tilauksen kyseisen tilaajan tilauslistalta. Unsubcibe-viestipaketin saatuaan välittäjä lähettää varmennuksen tilauksen lopettamisesta (unsuback). Kun asiakas vastanottaa unsuback-viestipaketin, asiakas tietää, että tilaus on lopetettu.[17]

4 FREERTOS-REAALIAIKAKÄYTTÖJÄRJESTELMÄ

Tässä luvussa käydään läpi FreeRTOS-reaaliaikakäyttöjärjestelmän perusteet sekä mihin FreeRTOS:iä käytetään, ja miksi se on hyödyllinen ohjelmistokehityksessä. Kappaleessa käsitellään myös mikä on kerneli ja mitä se tarjoaa.

4.1 Yleistä

FreeRTOS on ilmainen, täysin avoimeen lähteeseen perustuva käyttöjärjestelmä. FreeRTOS on suunniteltu toimimaan pienillä mikroprosessoreilla, joten käyttöjärjestelmä itsessään on hyvin pienikokoinen. FreeRTOS-reaaliaikakäyttöjärjestelmällä on mahdollista säästää RAM-muistia kehittäessään ohjelmistoa mikroprosessoreilla, jotka kärsivät pienestä RAM- ja Flash-mustista.

4.2 Toiminnallisuus

FreeRTOSin toiminnallisuus perustuu skeduleriin (scheduler). FreeRTOS mahdollistaa prosessien käsittelyn samanaikaisesti kernelin tarjoaman skedulerin avulla. FreeRTOS käyttää perinteistä reaaliaikaista skeduleria, joka päättää mitä taskeja (task) prosessori suorittaa taskeille määritettyjen prioriteettien perusteella. Skeduleri toimii siis sille määrättyyn tahtiin. Tätä suositaan sulautettujenjärjestelmien kehityksessä, sillä sulautetuissa järjestelmissä tietyt tapaukset vaativat tarkalleen sille määritellyn ajanjakson, jonka skeduleri mahdollistaa.[25]

4.2.1 Kerneli

Reaaliaikakäyttöjärjestelmän ydin on kerneli. Kerneli tarjoaa skedulerin, joka mahdollistaa prosessien samanaikaisen käsittelyn. Kernelin avulla järjestelmän on mahdollista käyttää eri muistinvarausmenetelmiä ja luoda täysin staattisesti allokoituja järjestelmiä. Taskien välinen kommunikointi onnistuu myös kernelin avulla.

Reaaliaikajärjestelmä, joka käyttää kernelin tarjoamaa skeduleria suorittaa taskeja käyttämällä pelkästään taskien omia sisältöjä, eikä se ole riippuvainen muista tas-

keista, ellei ohjelmaa ole erikseen niin suunniteltu. Jokaisella taskilla on oma pinonsa (stack) ja prioriteettinsa. Skeduleri antaa taskeille suoritusaikaa niiden prioriteettien perusteella. Skedulerin vaihtaessa suoritettavaa taskia, taskin sisältö säilötään taskin pinoon. Taskin saadessaan skedulerilta suoritusvuoron, taskia voidaan jatkaa kohdasta, minne se viimeksi jäi käyttämällä taskin pinoa. Mikäli useampi taski omaa saman prioriteetin ja ovat valmiina suoritettavaksi jakavat he prosessointiajan.[7]

4.2.2 Samanaikainen suorittaminen

Mikäli järjestelmän kehityksessä käytettävä prosessori omaa useamman ytimen niin skeduleria ei tarvita, sillä jokainen taski voidaan suorittaa omalla ytimellä. Sulautettujen järjestelmien kehityksessä käytetään kuitenkin yleensä yksiytimisiä prosessoreja. FreeRTOS-reaaliaikakäyttöjärjestelmän avulla perinteinen yksiytiminen prosessori, joka suorittaa yhtä taskia kerrallaan saadaan vaikuttamaan siltä, että taskit suoriutuisivat yhtäaikaisesti. Käyttöjärjestelmä ohjaa prosessoria hyppimään eri taskien välillä niin nopeasti.

Kuva 6. Yksiytiminen prosessori suorittaa useampaa taskia samanaikaisesti.

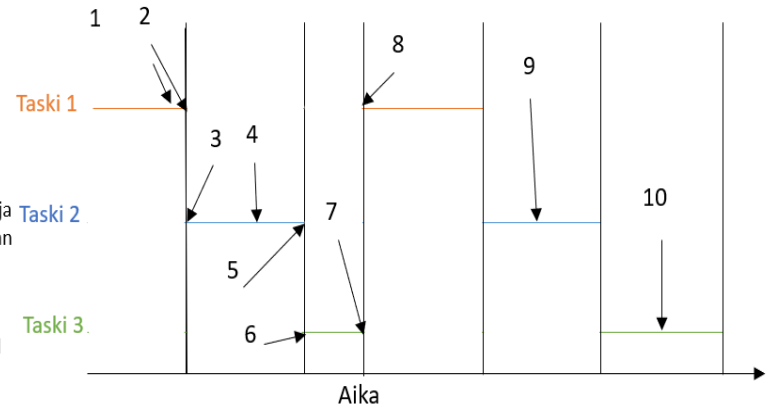
Kuvassa 6 nähdään, kuinka useampi taski suoriutuu yksiytimisellä prosessorilla ja kuinka se näyttää ihmissilmään suoriutuvan samanaikaisesti, sillä skeduleri vaihtaa

suoritettavaa taskia niin nopeasti. Samanaikainen suorittaminen mahdollistaa isojen ja monimutkaisten ohjelmistojen jaottelun pienemmiksi osiksi, jolloin niiden käsittely on helpompaa. Jaoteltua järjestelmää on myös helpompi testata ja sen koodia uudelleen käyttää.[14, 28]

4.2.3 Skeduleri

Kerneli voi keskeyttää taskin ja jatkaa taskia monesti taskin elinkaaren aikana. Näitä keskeytyksiä taskeille ovat unitila (sleep), jolla luodaan viivettä tai estotila (block), jota käytetään, mikäli taskille tarpeelliset resurssit eivät ole vapaina. Resurssien vapauduttua taski palaa suoritustilaan. Kernelin tehtävä on huolehtia, että skeduleri suorittaa tietyn taskin sille määrätyllä hetkellä.[7, 19]

- (1) Suoritetaan taskia 1.
- (2) Kerneli keskeyttää taskin 1.
- (3) Kerneli palaa taskiin 2.
- (4) Taskia 2 suorittaessa taski varaa prosessorin oheislaitteet itselleen.
- (5) Kerneli keskeyttää taskin 2 ja palaa taskiin 3 (6).
- (7) Taski 3. yrittää päästä käsiksi varattuihin prosessorin oheislaitteisiin ja keskeyttää itsensä.
- (8) Kerneli palaa taskiin 1.
- (9) Taski 2. suoritetaan loppuun ja varatut oheislaitteet vaipautetaan
- (10) Taski 3. pääsee käsiksi oheislaitteisiin ja täten tehtävää voidaan suorittaa, kunnes kernel tauottaa sen.



Kuva 7. Skedulerin toiminta.[19]

4.3 Taskien välinen kommunikointi

FreeRTOS-reaaliaikakäyttöjärjestelmä käyttää eri menetelmiä taskien väliseen kommunikointiin. Näitä menetelmiä ovat jonot, semaforit ja mutexit. Näillä menetelmillä taskien on mahdollista jakaa sisäisiä resursseja toisilleen. Näitä menetelmiä käyttäessä taski siirtyy estotilaan, mikäli resurssi mitä taski kaipaa, ei ole vapaana. Resurssin vapauduttua taski voi suoriutua normaalisti.

4.3.1 Jonot

Jonot (queues) ovat ensisijainen menetelmä taskien keskeiseen kommunikaatioon. Jonojen avulla taskit voivat kommunikoida toistensa kanssa taskien ja keskeytyksien välillä. Useat jonomenetelmät toimivat thread safe-toimintaa käyttäen FIFO (First In First Out) periaatteella, eli keskeytyksen ei pitäisi liittyä jonopuskurin sisältöön. FIFO-periaatteella tuorein jonopuskuriin lisätty data menee vakiona jonon perälle, mutta data on myös mahdollista siirtää jonon kärkeen. Data kopioidaan osoittajaan jonopuskurissa, jolloin puskurin data ei muutu, vaikka alkuperäistä muuttujaa muutettaisiin.

Mikäli taski yrittää lähettää dataa täysinäiseen jonopuskuriin, taski menee estotilaan, kunnes jonopuskurista vapautuu tilaa. Sama pätee, kun taski yrittää lukea jonopuskurista, mikä on tyhjä. Estotila ei syö prosessointiaikaa, joten muut taskit voivat tällä välin suoriutua. Jos useampi estotilassa oleva taski yrittää lähettää dataa täynnä olevaan jonopuskuriin, niin taski, jolla on isoin prioriteetti, vapautetaan ensin, jonka avulla saadaan tila käyttöön.[7, 10, 12, 22]

4.3.2 Semaforit

Semaforit ovat muuttujia, jotka ohjaavat taskien pääsyä käsiksi yleisiin yhteisiin resursseihin. Semaforia käytetään synkronoinnin tuottamiseen taskien tai taskien ja keskeytyksien välillä. Semaforien toiminnot mahdollistavat estotilan ajanjakson määrittelyn, eli aikakatkaisun. Ajanjakson määrittelyllä tarkoitetaan sitä, kuinka monta kellojaksoa (tick) taski yrittää ottaa semaforin käyttöön sen ollessa varattu,

ennen estotilaan päätymistä. Mikäli useampi taski yrittää päästä käsiksi samaan semaforiin, isomman prioriteetin omaava taski saa sen käyttöön semaforin vapautuessa.

Binäärinen semafori on kuin jono, mutta se voi pitää sisällään vain yhtä kohdetta. Semaforin jonon tila on siis 1 (täynnä) tai 0 (tyhjä). Prosessorin resurssien säästämiseksi taski, joka käyttää oheislaiteita, pidetään estotilassa. Tällöin taski suoritetaan vain silloin, kun se on mahdollista. Tämä mahdollistetaan binäärisen semaforin avulla silloin, kun taski pyytää semaforia, semafori tietää asettaa taskin on estotilaan.

Keskeytysrutiinin avulla oheislaite vapauttaa (gives) semaforin silloin, kun oheislaite vaatii toimenpiteitä. Taski ottaa aina semaforin, eli lukee jonosta numeron yksi ja vaihtaa se nollaksi, mutta ei palauta sitä. Semafori palautetaan keskeytyksen avulla.

Laskevat semaforit toimivat samoin kuin binäärisetkin, eli ne ovat jonoja. Yleensä laskevaa semaforia käytetään kahteen eri tarkoitukseen: Laskevaa semaforia voidaan käyttää tapahtumien (event) laskemiseen tai resurssienhallinaan.

Laskeva semafori pitää yllään lukua siitä, kuinka monta tapahtumaa on tapahtunut. Tapahtuman hetkellä keskeytysohjelma (handler) antaa semaforin, jolloin semaforin lukumääräarvo nousee. Keskeytysohjelma ottaa semaforin tapahtumaa suorittaessa, joka laskee semaforin lukumääräarvoa. Lopussa semaforin lukumäärä kertoo, kuinka montako tapahtumaa on tapahtunut ja montako on suoritettu.

Resurssienhallinnatamenetelmässä semaforin lukumääräarvo indikoi vapaina olevia resursseja. Saadakseen hallintaan näitä resursseja, taski tarvitsee semaforin, jolloin lukumääräarvo laskee. Kun lukumääräarvo laskee nollaan, tarkoittaa se resurssien loppumista. Taskin suoriuduttua taski antaa semaforin takaisin, jolloin lukumääräarvo nousee.[7, 10, 12, 22]

4.3.3 Mutexit

Mutexit ovat kuin binäärinen semafori, jotka käyttävät prioriteetin perintämekanismia. Mutexit ovat hyvä vaihtoehto keskinäisen poissulkemisen (Mutual Exclusion) implementoinnissa. Keskinäinen poissulkeminen on ohjelman objekti, joka estää jaettujen resurssien samanaikaisen käytön. Keskinäisen poissulkemisen käytössä oleva mutexi toimii polettina (token), jota käytetään resurssien suojaamiseen. Taskin halutessa käyttää näitä resursseja, sen on saatava poletti. Poletin saatua taski suoriutuu, jonka jälkeen taski palauttaa poletin, jotta muut taskit voivat käyttää suojattuja resursseja.

Kuten binäärisessä semaforissa, mutexia odottavat taskit menevät estotilaan. Mutex eroaa silti binäärisestä semaforista, sillä mutex käyttää prioriteetin perintämekanismia. Eli jos korkean prioriteetin omaava taski saapuu estotilaan mutta mutexin poletti on sillä hetkellä käytössä pienemmän prioriteetin omaavalla taskilla, taskin prioriteetti nostetaan saman arvoiseksi estotilassa olevan taskin kanssa. Tällä mekanismilla korkean prioriteetin omaavat taskit saavat kuitenkin mutexin käyttöön heti sen vapauduttua, ja näin ollen ovat estotilassa mahdollisimman lyhyen ajan. [7, 10, 12, 22]

5 AWS

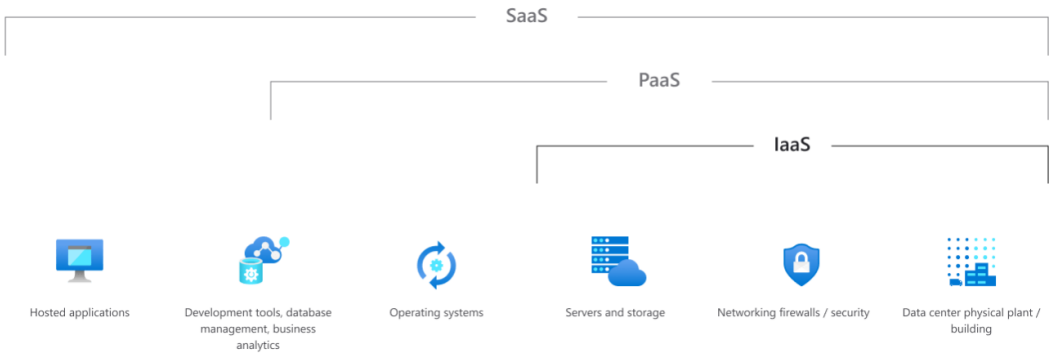
5.1 AWS-pilvipalvelusta yleistä

Amazon Web Service (AWS) on kattava ja käyttäjäystävällinen pilvipalvelualusta, jonka käyttö on hyvin yleistä yritysmaailmassa. AWS-pilvipalvelu alusta tarjoaa tietokantoja, tallennustilaa, sovellusten ja muiden IT-resurssien tiedon siirron pilvipalvelun kautta. Pilvipalvelualustan resurssien avulla käyttäjä on helppo rakentaa ja ylläpitää ohjelmistoja sekä tietokantoja.

Enne pilveä yritysten ja organisaatioiden täytyi itse ylläpitää datakeskuksiaan ja infrastruktuurin kokonaisuuden ylläpitäminen vaati huomattavan paljon resursseja ja työtä. Internetin yleistyminen johti tilan -ja verkkolaitteiden kysyntään, jolloin kaikkien yritysten ja organisaatioiden varat eivät riittäneet tämänkaltaisten palveluiden ylläpitämiseen. Pilvipalvelut syntyivät ratkaisemaan tämän ongelman.[1]

5.2 Pilvipalvelumallit

Pilvipalveluiden suosion kasvun johdosta on syntynyt useita erilaisia palvelumalleja, jotka auttavat vastaamaan käyttäjien eri tarpeisiin. Jokainen palvelumalli tarjoaa erilaisen abstraktion, hallinnan, joustavuuden ja hallinnan tasot.



Kuva 8. AWS tarjoamia palvelumalleja ja niiden tarjoamia ominaisuuksia.[27]

5.2.1 Infrastucture as a Service (IaaS)

IaaS-palvelumalli koostuu pilvi-IT:n perinteisistä ominaisuuksista. IaaS-palvelumalli tarjoaa pääsyn verkko-ominaisuuksiin, tietokoneisiin, virtuaalilaitteisiin sekä tallennustilaan. IaaS tarjoaa IT-resurssien joustavaa hallinnointia. IaaS-palvelumallin avulla pyritään välttämään ylimääräisiä kustannuskuluja ja monimutkaisten fyysisten palvelimien hallinnointia.[1]

5.2.2 Platform as a Service (PaaS)

PaaS-palvelumalli poistaa tarpeen hallinnoida taustalla olevaa infrastruktuuria, kuten käyttöjärjestelmää. PaaS-palvelumallia käyttäen resurssien hankinnasta, kapasiteetin suunnitellusta ohjelmistojen ajan tasalla pitämisestä ei tarvitse huolehtia. Tällöin sovellusten kehitys ja hallinnointi on huomattavasti tehokkaampaa.[1]

5.2.3 Software as a Service (SaaS)

SaaS-palvelumalli tarjoaa kokonaisen laitteiston, joka hallinnoi järjestelmää palvelimen avulla. SaaS-palvelumallissa tarkoituksena on ylläpitää ja hallinnoida palvelimen kokonaisuutta. Tällöin käyttäjän ei tarvitse kuin miettiä, mihin ohjelmistoa on käyttämässä. Esimerkkinä SaaS-järjestelmästä olisi verkkopohjaien sähköpostijärjestelmä, josta voidaan seurata lähetettyjä ja vastaanotettuja sähköposteja ilman minkäänlaista palvelimen tai käyttöjärjestelmän ylläpitoa.[1]

5.3 AWS IoT

AWS tarjoaa IoT-palveluita, joilla on mahdollista yhdistää ja hallinnoida miljardeja laitteita. AWS IoT:n avulla voidaan säilyttää, kerätä ja analysoida IoT-dataa. AWS IoT tarjoaa jokaiselle ohjelmiston tasolle sekä laitteelle turvan AWS IoT Device Defender-menetelmää käyttäen. AWS IoT Device Defender suojaa laitteiston dataa käyttäen erisalausmenetelmiä, valvomalla tiedon kulkua ja tarkastelemalla konfi-

guraatioita varmistaakseen, etteivät ne poikkea turvallisuuskäytännöistä. Konfiguraatiot ovat teknisiä ohjaimia, jotka auttavat pitämään datan turvassa, kun laitteet kommunikoivat keskenään tai pilven kanssa.[2]

5.3.1 AWS IoT-palveluita

AWS IoT tarjoaa palveluita laitteiden ohjelmistoille, datan analysointiin sekä laitteiden välistä kommunikointia ja ohjausta varten. Laitteiden ohjelmistoille suunnattuihin palveluihin kuuluu muun muassa FreeRTOS, jota käytettiin tässä projektityössä. Laitteiden ohjelmistoille suunnattuja palvelua käytetään yleisesti IoT-laitteistojen ja ohjelmistojen kehitykseen.

IoT-palveluita, joilla dataa analysoidaan ja kerätään, ovat muun muassa AWS IoT SiteWise, jota käytetään teollisuuslaitteiden datan keräämiseen, organisoimiseen sekä analysointiin. AWS IoT Events hallinnointipalvelu, jonka avulla tarkkaillaan laitteiden ja laitekantojen toiminnassa tapahtuvia muutoksia. AWS IoT Analytics, jolla voidaan operoida edistyksellistä analytiikkaa valtavissa IoT-datamäärissä. Ja DynamoDB, joka on täysin hallinnoitu, palvelimeton NoSQL-pohjainen datakeskus. DynamoDB:tä käytetään datavarastona, jonne AWS IoT-järjestelmässä kulkeva data on mahdollista säilöä.

Laitteiden väliseen kommunikointiin ja ohjaukseen suunnattuja IoT-palveluita, joita käytettiin myös opinnäytetyöprojektissa ovat muun muassa AWS IoT Device Defender ja AWS IoT Core. Kaikkien näiden palveluiden ytimenä toimii AWS IoT Core. AWS IoT Core:n avulla voidaan yhdistää miljardeja eri IoT-laitteita ja välittää biljoonia viestejä AWS-palveluihin ilman erillistä provisiointia tai hallinnointi-infrastruktuuria. AWS IoT Core mahdollistaa laitteiden välisen kommunikoinnin toimimalla MQTT-viestintäprotokollasta tuttuna välittäjänä (broker). AWS IoT Core välittää laitteilta tulleita julkaisuja muille IoT-laitteille, jotka ovat näitä tilanneet. AWS IoT Core tarjoaa myös Mirror Device State-ominaisuuden, jonka avulla voidaan luoda Device Shadow-tila. Mikäli IoT Core:en yhdistetty laite menettää yh-

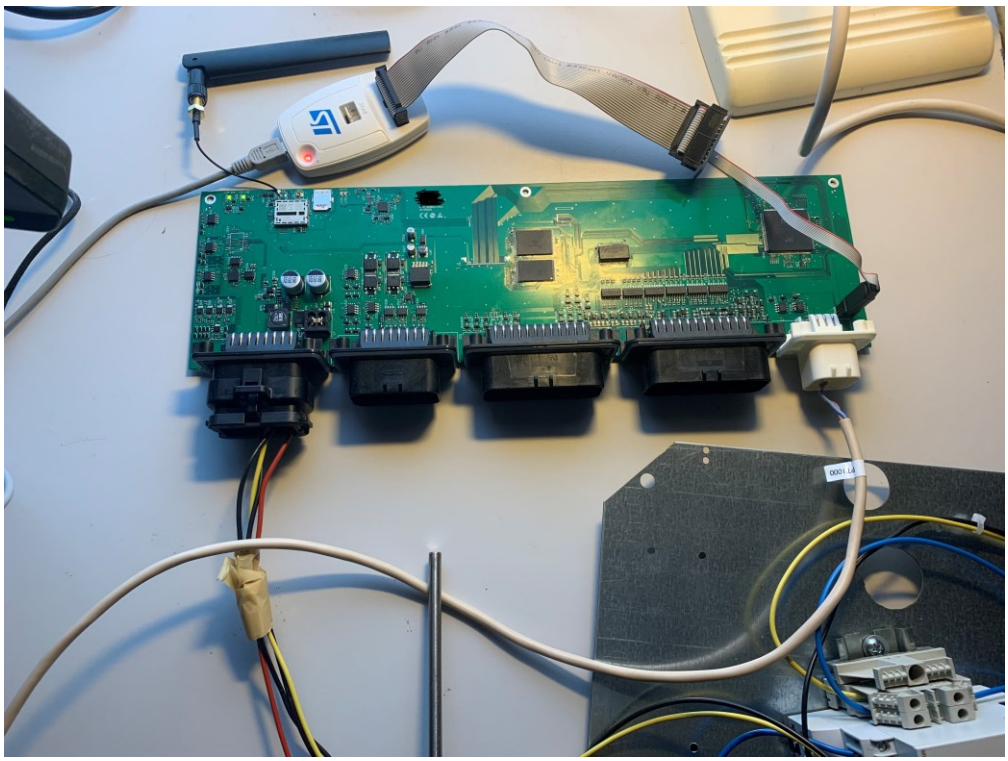
teyden Device Shadow-tila säilöö laiteen viimeisen tilan, milloin laite oli yhteydessä. Tällöin laitetta voidaan lukea ja täten esiintyisi muille laitteille yhdistettynä.[2]

6 PROJEKTIN TOTEUTUS

Tässä kappaleessa käydään läpi projektin tarkoitus ja sen tavoitteet. Kappaleessa käydään läpi myös projektin avainkomponentit ja niiden toiminnallisuus projektissa.

6.1 Projektissa käytetyt laitteet ja ohjelmisto

Projektin tarkoituksena oli kehittää järjestelmä, joka lähettää PT1000-lämpötila-anturilla mitattuja lämpötila-arvoja GSM-yhteyttä ja MQTT-kirjastoa hyödyntäen AmazonWebService:en sekä tallentaa ne DynamoDB-tietokantaan. Järjestelmän kehitykseen käytettiin STM32CubeIDE-kehitysympäristöä, jolla ohjelmoitiin STM32F479IIT6-mikroprosessoria. Ohjelman testaamiseen ja lataamiseen käytettiin ST-LINK/V2-debuggeria. Järjestelmässä käytetään FreeRTOS-reaaliaikakäyttöjärjestelmää, mikä toimii järjestelmän kokonaisuuden ytimenä. Järjestelmä on kehitetty kustomoidulle prototyyppilevylle, missä on järjestelmälle kriittinen komponentti HL7802 cellular-moduuli.



Kuva 9. Projektissa käytetty prototyyppilevy.

6.2 Kehitysympäristö

STM32CubeIDE on Eclipse-pohjainen kehitysympäristö, joka julkaistiin vuonna 2019. Kehitysympäristö on suunnattu C ja C++ kielen ohjelmistokehittäjille. STM32CubeIDE tarjoaa ohjelmistokehitykseen oheislaitteiston konfiguraation, generoidun koodin, kääntäjän ja debug-ominaisuuden STM32-mikroprosessoreille.

Uutta projektia aloittaessa STM32CubeIDE tarjoaa listan STM32-mikroprosessoreista ja mikro-ohjaimista. Alustan valittua CubeIDE generoi projektin ja konfiguroi valitut oheislaitteet valmiiksi. Ohjelmiston kehittäjän on mahdollista muokata alkuperäisiä oheislaitteiden konfiguraatioita myöhemmin, kuten muokata kellotaajuuksia sekä nastoja. Tämän jälkeen CubeIDE generoi koodin uudelleen, jolloin muokatut kohdat tulevat osaksi ohjelmaa.

6.2.1 FreeRTOS-reaaliaikakäyttöjärjestelmän implementointi

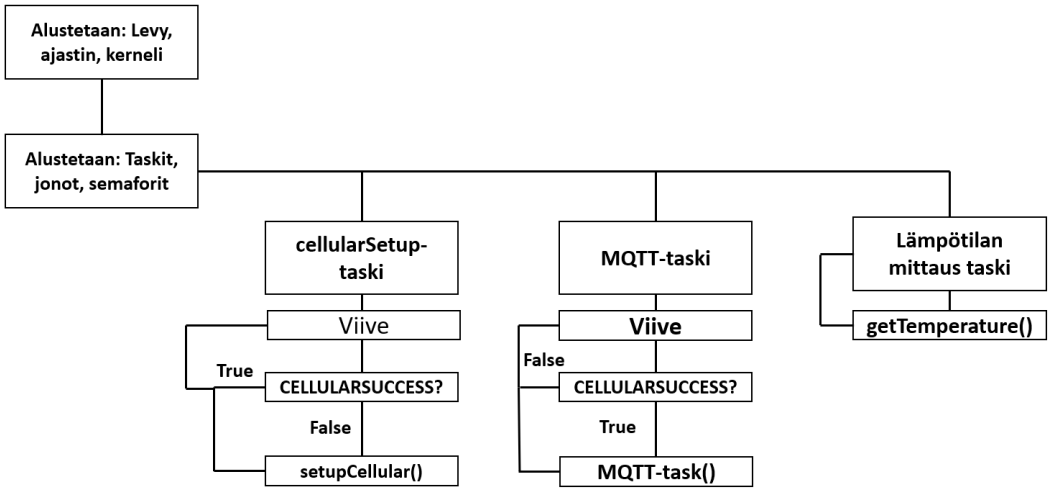
STM32CubeIDE tukee myös muutamia kolmannen osapuolen kirjastoja, jotka voidaan halutessa lisätä projektiin. Näitä ovat esimerkiksi FreeRTOS-reaaliaikakäyttöjärjestelmä ja MbedsTLS-salausalgoritmi. Näitä edellä mainittuja kolmannen osapuolen kirjastoja käytettiin opinnäytetyöprojektin toteutuksessa.

STM32CubeIDE:ssä oleva FreeRTOS-käyttöjärjestelmä on taustalla oleva ohjelmistokehys, jota ei suoranaisesti ohjata. FreeRTOS:sin ohjaamiseen käytetään CMSIS (Common Microcontroller Software Interface Standard) -RTOS kirjastoa CMSIS_V2. CMSIS_V2 avulla voidaan kommunikoida ohjelmistokehyksen kanssa, joka mahdollistaa FreeRTOS:sin ominaisuuksien käytön, kuten samanaikaisen suorittamisen ja skedulerin käytön.

Kun FreeRTOS on otettu käyttöön CubeIDE-konfiguraatio valikosta, voidaan sinne luoda ohjelmalle olennaisia taskeja. Tässä valikossa taskit ja niiden aloitusfunktiot voidaan nimetä, ja taskeille määritellään prioriteetit sekä se, paljonko kullekin taskille varataan muistia.

6.3 Projektin toteutus vaiheittain

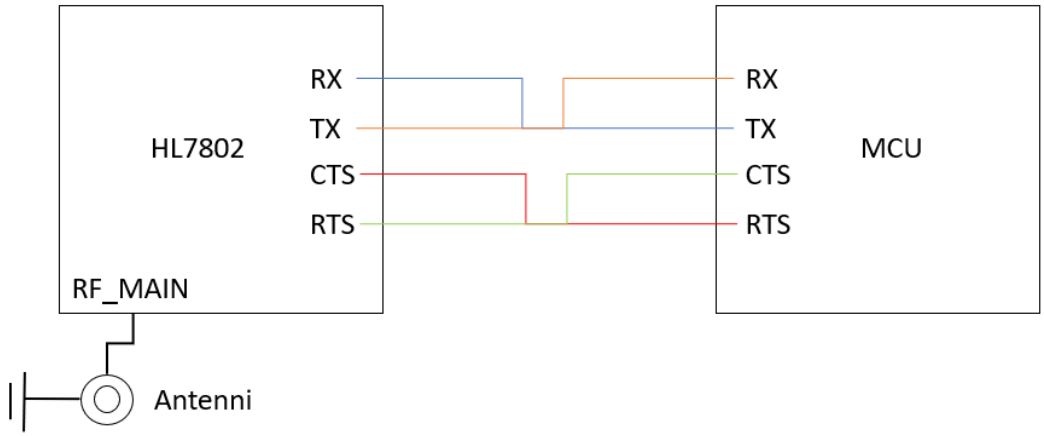
Seuraavissa alakappaleissa käydään läpi projektin kokonaisuuden suorittaminen. Kernelin käynnistyttyä skeduleri aloittaa FreeRTOS-taskien suorittamisen taskien prioriteettien ja funktioille määritettyjen ehtojen perusteilla.



Kuva 10. Vuokaavio, joka kuvaa järjestelmässä suoritettavia taskeja.

6.3.1 HL7802 Cellular-moduuli

Projektityössä matkapuhelinverkkoyhteyden luomiseksi käytettiin HL7802 cellular-moduulia. HL7802 on LPWA moduuli, joka tarjoaa tehokaan yhteyden globaaleihin LTE-M, NB-IoT, 2G ja GNSS verkkoihin. Projektityössä käytettävä verkkoyhteys oli 2G, joka määritellään setupCellular-funktiossa AT-komentoja käyttäen. HL7802 on Ready-to-Connect (R2C) moduuli, joka tukee sulautetun SIM-kortin (eSIM) käyttöä muodostaakseen globaalin datayhteyden radiotaajuuksiin. Radiotaajuuksien vastaanottamiseen ja lähettämiseen (Rx/Tx) käytettiin ANT-LTE-WS-SMA-dipoli antennia, joka kytkettiin moduulista lähtevään RF main-nastaan. **(Kuva 11.)** [9]



Kuva 11. Mikroprosessorin ja HL7802-moduulin välinen kytkentä.

Mikroprosessori ja HL7802-moduuli kommunikoivat USART-väylän välityksellä. Moduulia ohjataan lähettämällä mikroprosessorilta Tx-nastan kautta AT-komentoja, joihin moduuli vastaa Rx-nastan kautta. Ohjelmisto käyttää RS-232 ohjauslinjaa, jota kutsutaan laitteiston kulun ohjaukseksi (hardware flow control). Mikroprosessori ilmoittaa RTS-nastan kautta cellular-moduulin CTS-nastaan, että olisi valmis lähettämään komentoja moduulille. Moduuli ilmoittaa CTS-nastan kautta mikroprosessorille, olevansa valmis vastaanottamaan dataa.

Projektissa suoritettiin FreeRTOS-taski joka valmistelee matkapuhelinverkkoyhteyden, jotta dataa voidaan julkaista AWS:sään. Ohjelma asettaa cellular-lipun *false*-tilaan ja aloittaa funktion cellularSetup.

```
void cellularSetupTask(void *argument)
{
  /* USER CODE BEGIN 5 */
    cellularSetup.cellularFlag = false;

    bool retCellular = false;



  /* Infinite loop */
  for(;;)
  {
    if( retCellular == false )
    {
        retCellular = setupCellular();

        if( retCellular == true){
            cellularSetup.cellularFlag = true;
        }
    }

    osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

Kuva 12. FreeRTOS-taski, jolla muodostetaan yhteys matkapuhelinverkkoon.

Ensin cellularSetup-taski suorittaa alustavat funktiot matkapuhelinverkon käyttöliittymälle. Alustavissa funktioissa ohjelmalle määritetään mutexit ja semaforit, joidenka jälkeen cellular-moduuli otetaan käyttöön AT-komentoja käyttäen.

```
122 /*-----------------------------------------------------------*/
123
124 CellularError_t Cellular_CommonInit( CellularHandle_t * pCellularHandle,
125                                      const CellularCommInterface_t * pCommInterface )
126 {
127     CellularError_t cellularStatus = CELLULAR_SUCCESS;
128     CellularContext_t * pContext = NULL;
129
130     CellularTokenTable_t tokenTable =
131     {
132         .pCellularUrcHandlerTable             = CellularUrcHandlerTable,
133         .cellularPrefixToParserMapSize        = CellularUrcHandlerTableSize,
134         .pCellularSrcTokenErrorTable          = CellularSrcTokenErrorTable,
135         .cellularSrcTokenErrorTableSize       = CellularSrcTokenErrorTableSize,
136         .pCellularSrcTokenSuccessTable        = CellularSrcTokenSuccessTable,
137         .cellularSrcTokenSuccessTableSize     = CellularSrcTokenSuccessTableSize,
138         .pCellularUrcTokenWoPrefixTable       = CellularUrcTokenWoPrefixTable,
139         .cellularUrcTokenWoPrefixTableSize    = CellularUrcTokenWoPrefixTableSize,
140         .pCellularSrcExtraTokenSuccessTable   = NULL,
141         .cellularSrcExtraTokenSuccessTableSize = 0
142     };
143
144     /* Init the common library. */
145     cellularStatus = _Cellular_LibInit( pCellularHandle, pCommInterface, &tokenTable );
146
147     /* Init the module. */
148     if( cellularStatus == CELLULAR_SUCCESS )
149     {
150         pContext = *pCellularHandle;
151         cellularStatus = Cellular_ModuleInit( pContext, &pContext->pModueContext );
152     }
153
```

Kuva 13. CellularSetup-taskin alustusfunktiot.

Kun mutexit ja semaforit on määritetty, moduulin UE (User Equipment) ja Urc (Unsolicited Result Code) otetaan käyttöön. Liitteestä 1 nähdään, kuinka moduuli otetaan käyttöön AT-komentoja käyttäen. AT-komento ”AT+KSRAT=2” määrittää moduulin muodostamaan GMS-verkon. Kun AT-komennot on annettu, viimeistellään käyttöönotto AT-komennolla ”AT+CFUN=1,1”. Tämä komento pyytää moduulia käynnistymään uudelleen, jotta annetut komennot tulevat voimaan.[20]

Kun moduulin toiminnallisuudelle tarpeelliset komennot on annettu, otetaan käyttöön moduulin Urc. Näiden komentojen tarkoituksena on ilmoittaa tulevista tapahtumista GSM-verkossa.

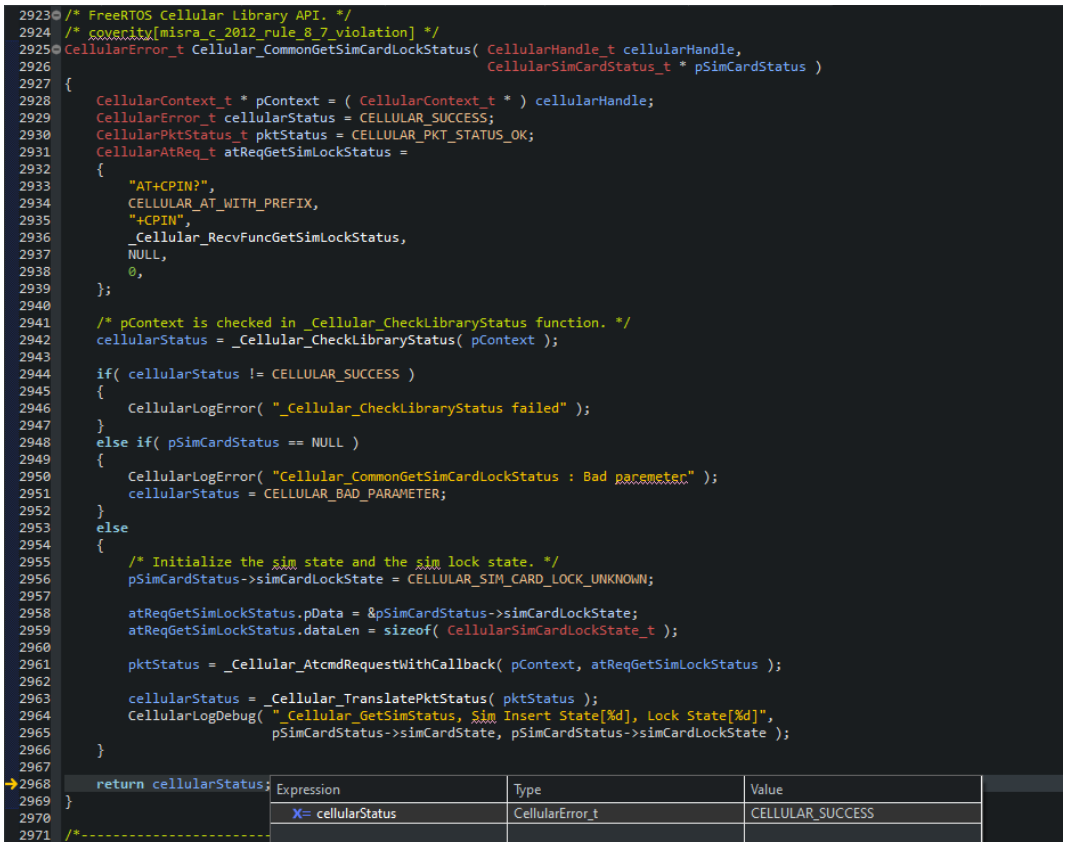
```
247 /* FreeRTOS Cellular Common Library porting interface. */
248 /* coverity[misra_c_2012_rule_8_7_violation] */
249 CellularError_t Cellular_ModuleEnableUrc( CellularContext_t * pContext )
250 {
251     CellularError_t cellularStatus = CELLULAR_SUCCESS;
252     CellularAtReq_t atReqGetNoResult =
253     {
254         NULL,
255         CELLULAR_AT_NO_RESULT,
256         NULL,
257         NULL,
258         NULL,
259         0
260     };
261
262     atReqGetNoResult.pAtCmd = "AT+COPS=3,2";
263     ( void ) _Cellular_AtcmdRequestWithCallback( pContext, atReqGetNoResult );
264
265     atReqGetNoResult.pAtCmd = "AT+CREG=2";
266     ( void ) _Cellular_AtcmdRequestWithCallback( pContext, atReqGetNoResult );
267
268     atReqGetNoResult.pAtCmd = "AT+CEREG=2";
269     ( void ) _Cellular_AtcmdRequestWithCallback( pContext, atReqGetNoResult );
270
271     atReqGetNoResult.pAtCmd = "AT+CTZR=1";
272     ( void ) _Cellular_AtcmdRequestWithCallback( pContext, atReqGetNoResult );
273
274     return cellularStatus;
```

Kuva 14. Moduulille annetut AT-komennot Urc:n varalta.

Kuvassa 14 moduulille annettuja komentoja. ”AT+COPS=3,2” valitsee operaattorin. ”AT+CREG=2” ottaa verkkorekisteröinnin- ja sijaintintiedot käyttöön. ”AT+CEREG=2” määrittää EPS-verkon rekisteröinnin tila. Ja ”AT+CTZR=1” raportoi aikavyöhykkeen.[20]

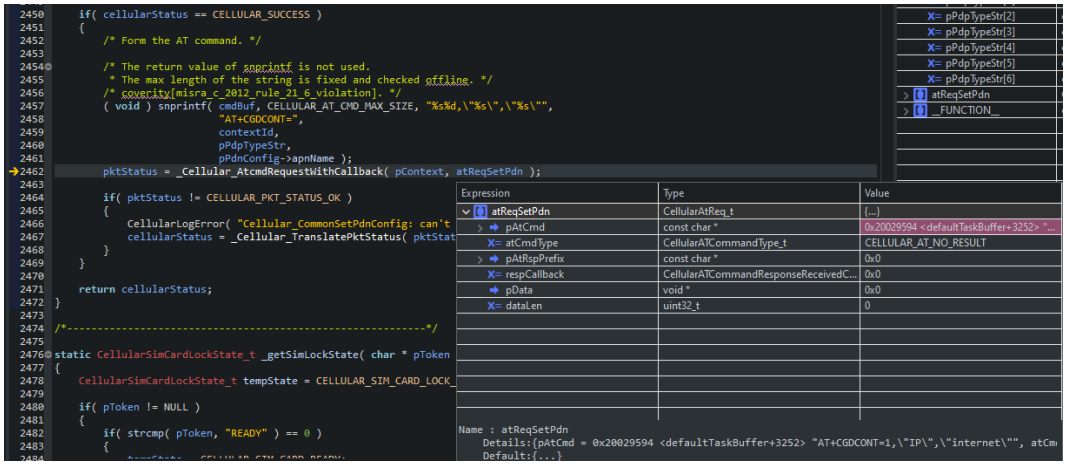
Kun moduuli on otettu käyttöön cellularsetup-taski tarkistaa SIM-kortin tilan ja ottaa sen käyttöön. (**Kuva 15.)** Funktio tarkistaa SIM-kortin lukitustilan. Funktiolle

annettu AT-komento ”AT+CPIN?” kuuluisi pitää sisällään SIM-kortin PIN-koodin. Tässä tapauksessa SIM-kortilla ei käytetty PIN-koodia.[20]



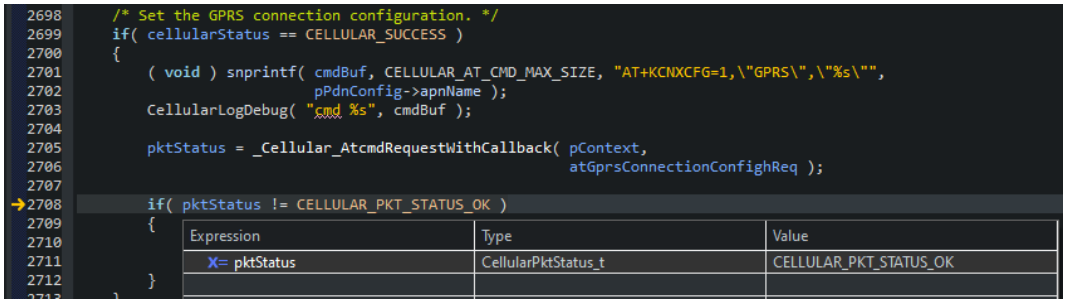
Kuva 15. SIM-kortin tilan tarkistusfunktio, joka palauttaa onnistuneentilan.

Kun SIM-kortin tila on tarkastettu ja funktio palauttaa ”CELLULAR_SUCCESS” tilan, PDN-konfiguraatiot otetaan käyttöön, joissa määritellään verkkoprotokollaksi IPv4. Tämä tapahtuu AT-komennon ”AT+CGDCOUNT=” avulla.[20]



Kuva 16. Moduulille annettu AT-komentopaketti.

AT-komento saa ”AT+CGDCOUNT=” parametrit ”1”, ”IP” ja ”internet”. Parametri ”1” on paikallinen kontekstitunniste, joka identifioi PDP-kontekstin, jotta siihen voidaan jatkossa viitata. Parametri ”IP” määrittää IPv4-verkkoprotokollan käytön. Parametri ” internet” on määritetty cellular-konfiguraatiotiedostossa ja se määrittää APN:än. Kun PDN-konfiguraatiot on asetettu, ohjelma konfiguroi GPRS-tiedonsiirtoyhteyden. **(Kuva 17.)** AT-komennolle ”AT+KCNXCFG=” annetut parametrit ovat ”1”,”GPRS” ja ”internet”. Parametri ”1” viittaa edellä mainittuun paikalliseen PDP-kontekstitunnisteeseen. ”GPRS” määrittää tiedonsiirtopalvelun. Parametri ”internet” taas viittaa samaan APN:ään.[20]



Kuva 17. GPRS-tiedonsiirtoyhteyden konfigurointi.

Kun PDN ja GPRS on konfiguroitu ohjelma aktivoi PDN:n erillisessä funktiossa. Funktio tarkistaa vielä, onko konfiguroitu PDN käyvä. Tämän suoritettua aktivoidaan pakettikytkentä (Packet Switching) tilaan *liitetty* AT-komennolla ”AT+CGATT=1”. Kun pakettikytkentä on liitetty ohjelma tarkistaa liitteen tilan. Jos

liitteen tila ei ole aktiivinen, ohjelma aktivoi PDN sisällön nostamalla PDP (Packet Data Protocol) tilan ylös AT-komennolla ”AT+KCNXUP”.[20]

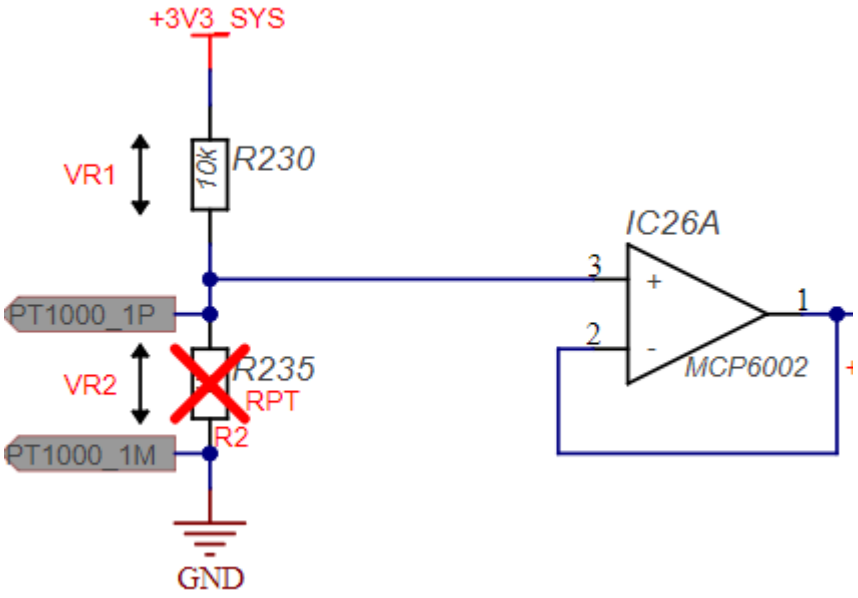
Cellular_ActivePdn-funktion palautettuaan tilan ”CELLULAR_SUCCES” ohjelma suorittaa funktion Cellular_GetIPAddress. AT-komennolla ”AT+CGPADDR” annetaan paikallinen kontekstitunnus, jolla saadaan esille PDP-osoitteen, joka viittaa IPv4 osoitteeseen. IP-osoitteen saatuaan, funktio palauttaa tilan ”CELLULAR_SUCCESS”.[20]

Lopuksi funktio suorittaa PND-tilan tarkastuksen enne kuin palauuttaa cellularSetup-funktion tilan *True*. Saatuaan tilan *True* matkapuhelinverkkoyhteys on onnistuneesti muodostettu ja näin ollen ohjelman on mahdollista lähettää dataa pilveen. Liitteessä 2 nähdään setupCellular-funktio kokonaisuudessaan.

6.3.2 PT-1000

PT-1000 on vastuslämpötila-anturi, joka mittaa lämpötilaa lämpötilariippuvaisen sähkövastuksen avulla. Lämpötilan ollessa nolla astetta vastuslämpötila-anturi resistanssi on 1000Ω. Lämpötilan noustessa reisistanssi kasvaa lähes lineaarisesti.

Opinnäytetyössä mitattavan ympäristön lämpötila ei mene miinusasteisiin. Lämpötilan laskemiseen tarvitaan vastuslämpötila-anturin resistanssi nollassa asteessa, anturin sen hetkinen resistanssi, sekä lineaarinen -ja neliöllinen lämpötilakerroin. PT-1000 ja käyttöjännitteen väliin on kytketty operaatiovahvistin jännitteen puskurointia varten. **(Kuva 18.)** Kuvassa 18 yli rastittua vastusta käytettiin protolevyn testaus vaiheessa. AD-muuntaja mittaa kymmenen näytettä nastalta, joista laskee anturin nastaan menevän jännitteen keskiarvon. Näin ollen jännitteenjako kaavalla 1 mitataan sen hetkinen resistanssi. Kun resistanssi on saatu selville kaavoja 2 ja 3 käyttäen saadaan selville ympäristön sen hetkinen lämpötila.



Kuva 18. PT-1000 kytketty operaatiovahvistin.[13]

$$R_\Delta = R_{230} * \frac{1}{\left(\frac{V_{in}}{V_{out}} - 1\right)}, \text{jossa} \quad (1)$$

$R_\Delta =$ Näytteistä laskettu resistanssi

$R_{230} = 10k\Omega$

$V_{in} = 3.3V$

$V_{out} =$ AD − muuntiemelta laskettu jännite

$$R_t = (R_\Delta - R_0), \text{jossa} \quad (2)$$

$R_t =$ resistanssi suhteessa lämpötilaan

Vastuslämpötilamittarin vakio resistanssi $R_0 = 1000\Omega$

$$t = \frac{R_t}{R_0 * (A + 2 * B)}, \text{jossa} \quad (3)$$

$t = lämpötila$

Lineaarinen lämpötilakerroin $A = \ 3.9083 * 10^{-3}$

Neliöllinen lämpötilakerroin $B = \ -5.7750 * 10^{-7}$

Lämpötilan mittaus suoritetaan FreeRTOS-taskissa. Taski suorittaa funktion, joka mittaa lämpötilan ja lisää lämpötila-arvon taulukkoon, joka myöhemmin julkaistaan AWS-pilvipalveluun MQTT-taskissa.

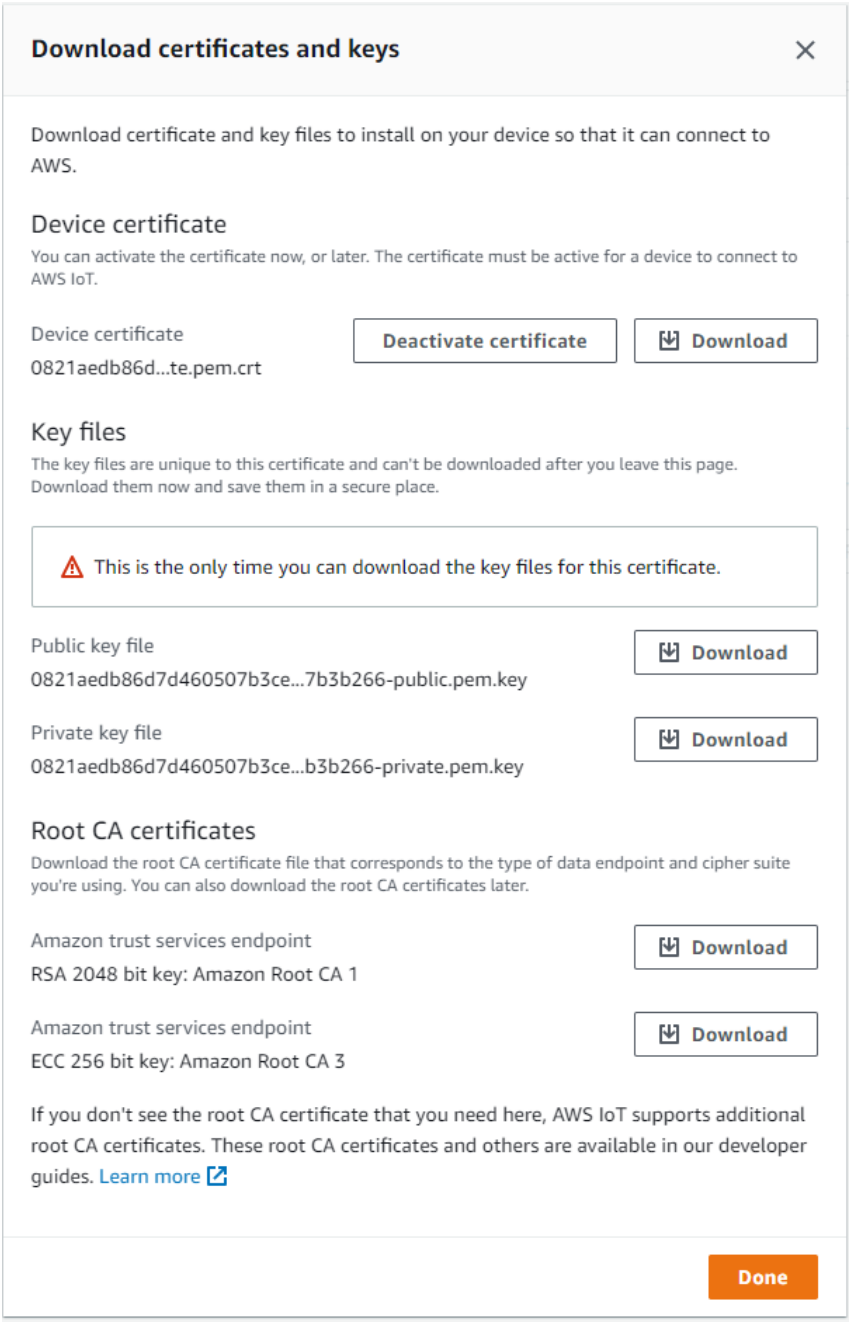
6.3.3 AWS-käyttäjän ja Thing-objektin määrittely

Jotta mitatut lämpötila arvot saataisiin lähetettyä matkapuhelinverkkoyhteydellä ja MQTT:n avulla AWS:sään, on luotava AWS-käyttäjätili ja konfiguroitava IAM (Identity and Access Management)-käyttäjä. IAM-käyttäjän avulla voidaan määritellä, millä laitteilla on oikeudet käyttää AWS-palvelemin resursseja. IAM-käyttäjän luominen suoritetaan root-käyttäjänä. Hyviin käytäntöihin kuuluu, ettei root-käyttäjää käytettäisi AWS-palvelimelle kulkeutuvan datan hallinnointiin tai seuraamiseen, vaan luotaisiin erillisiä käyttäjiä tällaiseen käyttötarkoitukseen.

Luodessaan IAM-käyttäjää, käyttäjälle tulee antaa nimi. Opinnäytetyöprojektin tapauksessa käyttäjän nimi on samuli_xedi. Käyttäjälle samuli_xedi myönnetään oikeudet: AmazonFreeRTOSFullAccess ja AWSIoTFullAccess. Kun IAM-käyttäjä on luotu ja sille kuuluvat oikeudet on myönnetty, saamme AWS-palvelimen endpoint-osoitteen. Tätä osoitetta käytetään myöhemmin MQTT-kirjasto funktioissa.

Seuraavaksi määritellään AWS IoT-käytännöt (policy). Nämä käytännöt liitetään myöhemmin laitteelle tarkoitettuihin sertifikaatteihin, joita laite näyttää muodostaessa yhteyden ja lähettäessä viestejä AWS IoT Core:lle. Käytännöt määrittävät mitä AWS IoT-toimintoja laitteisto on oikeutettu suorittamaan, kuten tilaamaan tai julkaisemaan MQTT-aiheita. Projektissa käytettävät käytännöt, ovat ominaisuus yhdistää (iot:connect), julkaista (iot:publish), tilata (iot:subscribe) ja vastaanottaa (iot:recieve) dataa.

Kun käytännöt ovat määritelty luodaan thing-objekti, joka kuvaa AWS IoT:hen yhteydessä olevaa laitetta. Thing-objektia luodessa, objektille annetaan juuri luodut käytännöt. Kun thing on yhdistetty käytäntöihin, AWS generoi ladattavat sertifikaatit ja avaimet. **(Kuva 19.)** Näiden avulla järjestelmä mahdollistaa MQTT-viestinnän AWS IoT:hen.



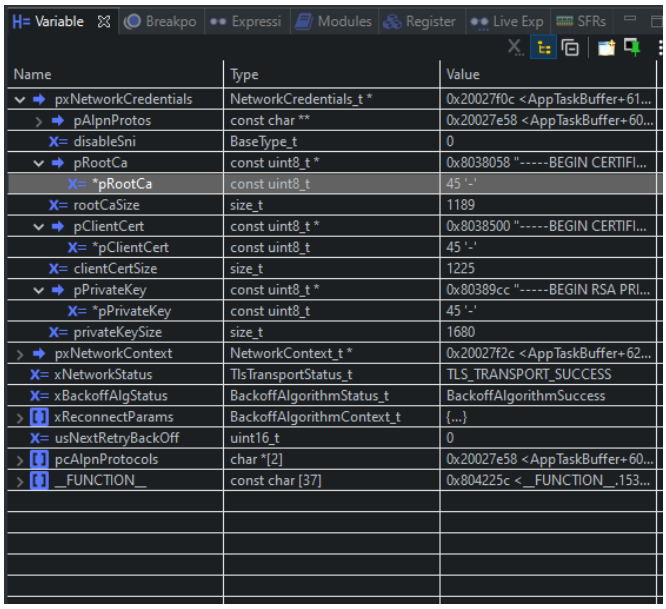
Kuva 19. AWS:sän generoimat sertifikaatit ja avaimet thing-objektille.

6.3.4 CoreMQTT

AWS:sän ja ohjelmoitavan laitteen väliseen MQTT-viestinnän muodostamiseen projektissa käytettiin MQTT-kirjastoa, nimeltä coreMQTT. Jotta yhteys olisi mahdollista luoda, coreMQTT-kirjasto vaatii AWS-käyttäjältä saatuja sertifikaatteja ja avaimia tunnistautuakseen palvelimelle. Ladatut sertifikaatit ja avaimet määritetään coreMQTT-konfiguraatiotiedostoon. Konfiguraatiotiedostoon määritellään myös laitteelle annettu thing-nimi, palvelimen portti ja endpoint, sekä aihe johon laite julkaisee.

Liitteessä 3 nähdään MQTT-taski funktio kokonaisuudessaan. Funktioon tehty ehto estää MQTT-taskin suoriutumista ennen kuin matkapuhelinverkkoyhteys on luotu, eli ohjelma tarkistaa onko cellularSetup-funktio palauttanut cellular-lipun tilaan *true*. Kun matkapuhelinverkkoyhteys on luotu, ohjelma voi aloittaa suorittamaan MQTT-taskia.

MQTT-taski muodostaa TLS-istunnon AWS IoT:n tarjoaman välittäjän kanssa. Ohjelma siirtää konfiguraatiotiedostossa määritetyt avaimet ja sertifikaatit taulukkoon, jota käytetään yhteyden muodostamisessa. **(Kuva 20.)**

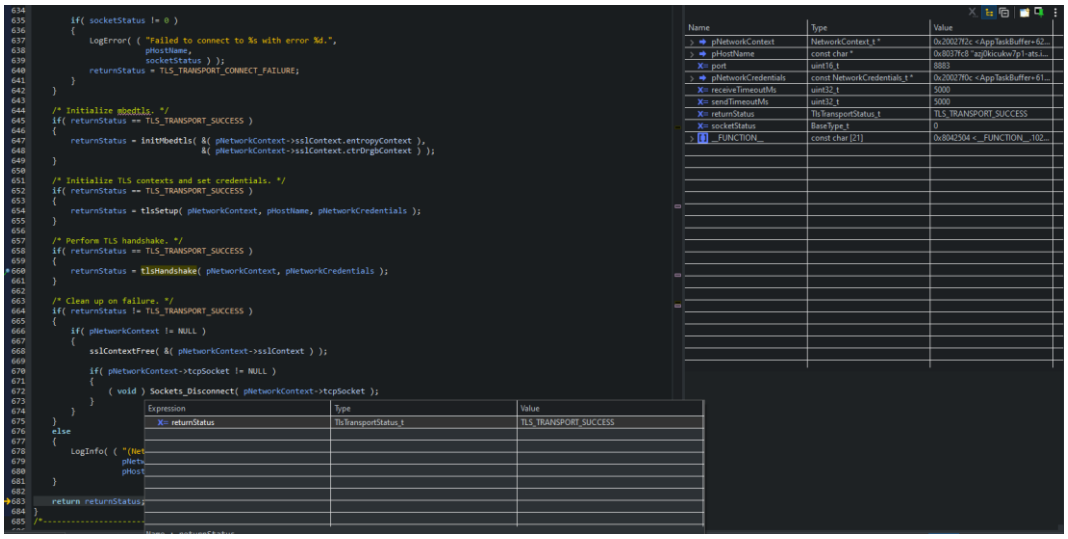


Kuva 20. Sertifikaatit ja avaimet siirrettynä pxNetworkCrenetials-taulukkoon.

Liitteessä 4 funktiossa: TLS_FreeRTOS_Connect tarkastetaan onko pxNetworkCrenetials-taulukon sisältö käyvä ja palauttaa tilan *TLS_TRANSPORT_SUCCES*. Seuraavaksi taski suorittaa Sockets_Connect-funktion, joka luo TCP-yhteyden palvelimelle. Jotta TCP-yhteys voidaan muodostaa, ohjelma luo uuden TCP-verkkosoketin, jolla laite tunnistautuu isäntäverkolle, eli tässä tapauksessa juuri muodostetulle matkapuhelinverkolle. TCP-verkkosoketin luotua, funktio muodostaa TCP-yhteyden.

Kun TCP-yhteys on onnistuneesti muodostettu, ohjelma antaa mutexin ja alustaa MbedTLS-salausalgoritmin, joka lisää järjestelmässä käytettävien yhteyksien salaussapysymistä. Salaukseen käytetään myös STM32CubeIDE:en tarjoamaa RNG-ominaisuutta, joka antaa MbedTLS:lle satunnaisesti generoidun numerosarjan, jota ominaisuus käyttää sisältöjen salaamiseen. Kun salaukset on luotu, TLS otetaan käyttöön. TLS käyttöön ottovaiheessa sertifikaateille ja avaimille asetetaan mbedTLS:sän luomat salaukset.

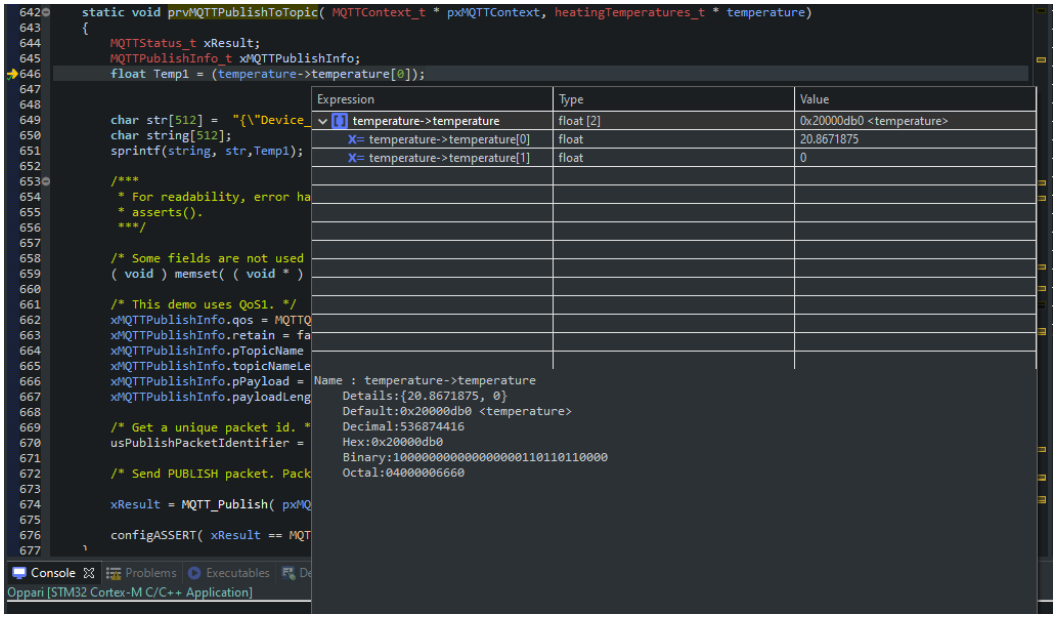
TLS_FreeRTOS_Connect-funktio viimeistellään suorittamalla TLS-kättely. TLS-kättelyn avulla laite ja palvelin määrittävät salaiset avaimet, joiden avulla ne kommunikoivat. Kun TLS-kättely on suoritettu TLS_FreeRTOS_Connect-funktio palauttaa tilan *TLS_TRANSPORT_SUCCESS.* **(Kuva 21.)**



Kuva 21. TLS_FreeRTOS_Connect-funktio palauttaa onnistuneen tilan.

TLS-yhteyden muodostettua, ohjelma lähettää MQTT-välittäjälle connect-viestipaketin funktiossa prvCreateMQTTConnectionWithBroker/LIITE 3/, /LIITE 5/. Connect-viestipakettia lähettäessä MQTT-välittäjälle, funktio määrittää viestipaketin sisällön. Cleansession-lippu asetetaan *true* tilaan, jolloin MQTT-viestintäprotokollan katkettua välittäjä pyyhkii istunnon datan. Sama pätee myös, jos edellisen yhdistys kerran tiedot ovat jääneet välittäjälle, välittäjä pyyhkii ne. Viestipaketin ClientID on laitteelle määritetty Thing-objekti, jolla laite tunnistautuu välittäjälle. Lopuksi connect-viestipaketti ilmoittaa keepAlive- ajan, joka määrittää kauanko yhteys välittäjän kanssa pysyy, ilman välittäjän ja laitteen välistä kommunikaatiota.

Kun connect-viestipaketti on lähetetty välittäjälle, luodaan julkaistava lämpötila viestipaketti funktiossa: prvMQTTPublishToTopic /LIITE 6/. Lämpötila viestipaketti pitää sisällään aiheen, jota tilataan AWS IoT-tilaajalla. Lisäksi viestipaketti sisältää payload-merkkijonon, joka kuvaa viestin sisältöä ja sisältää lämpötila-arvon, jonka PT-1000 taski on tallentanut taulukkoon. Funktio MQTT_GetPacketId määrittää viestipaketille tunnuksen, jotta välittäjä välittää oikean paketin oikealle tilaajalle. Kun julkaistava viestipaketti on valmis, viesti paketti lähetetään välittäjälle, joka välittää sen viestipaketin aihetta tilanneelle tilaajalle.



Kuva 22. PT-1000 taskin mittaama lämpötila julkaistaan AWS:sään.

Kun viestipaketti on julkaistu välittäjälle, MQTT-viestintä katkaistaan, jotta ohjelman muita toimintoja voidaan suorittaa. MQTT-viestintä katkaistaan lähettämällä välittäjälle disconnect-viestipaketti. Disconnect-viestipaketin lähetettyä, MQTT-taski sulkee verkkoyhteyden ja resetoi tilatun aiheen SUBACK-tilan.

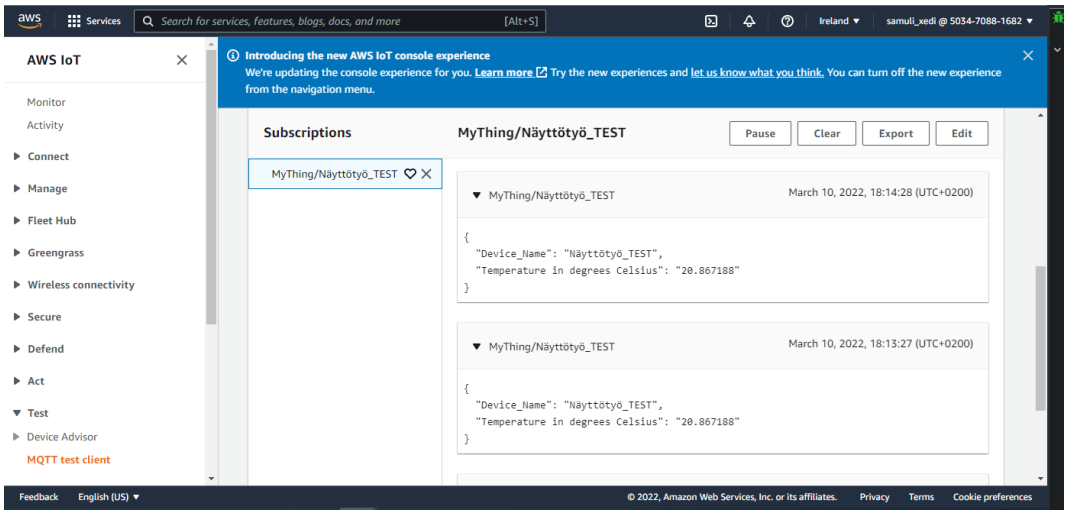
7 JÄRJESTELMÄN TESTAUS

7.1 IoT-järjestelmän kokonaisuuden suorittaminen

Järjestelmän kokonaisuuden suorittamainen aloitetaan oheislaitteiden resetoinnilla, jonka yhteydessä alustetaan Flash-käyttöliittymä ja Systick-ajastin (timer). Oheislaitteille asetetaan STM32CubeIDE:ssä määritetty konfiguraatiot, jonka jälkeen kerneli alustetaan. Kernelin alustuksen jälkeen luodaan järjestelmässä suoritettavat FreeRTOS-taskit ja skeduleri käynnistetään.

Skeduler suorittaa taskeja niille määriteltyjen prioriteettien perustella, mutta ohjelmalle ensisijainen prioriteetti on luoda matkapuhelinverkkoyhteys. MQTT-taskia järjestelmä ei voi suorittaa, ennekuin matkapuhelinverkkoyhteys on luotu. Siispä skeduleri suorittaa lämpötilan mittaus taskia matkapuhelinverkko taskin kanssa saman aikaisesti.

Kun matkapuhelinverkkoyhteys on muodostettu ja taski palauttaa cellular-lipun tilaan *true* skeduleri pääsee suorittamaan MQTT-taskia. MQTT-taski julkaisee lämpötila mittaus-taskin mittaamia lämpötila-arvoja MQTT-viestipakettina. **(Kuva 22**.) AWS IoT-välittäjälle. AWS IoT-käyttäjä tilaa aihetta: MyThing/Näyttötyö_TEST ja vastaanottaa järjestelmän julkaiseman viestipaketin ja sen sisällön. **(Kuva 23.)**

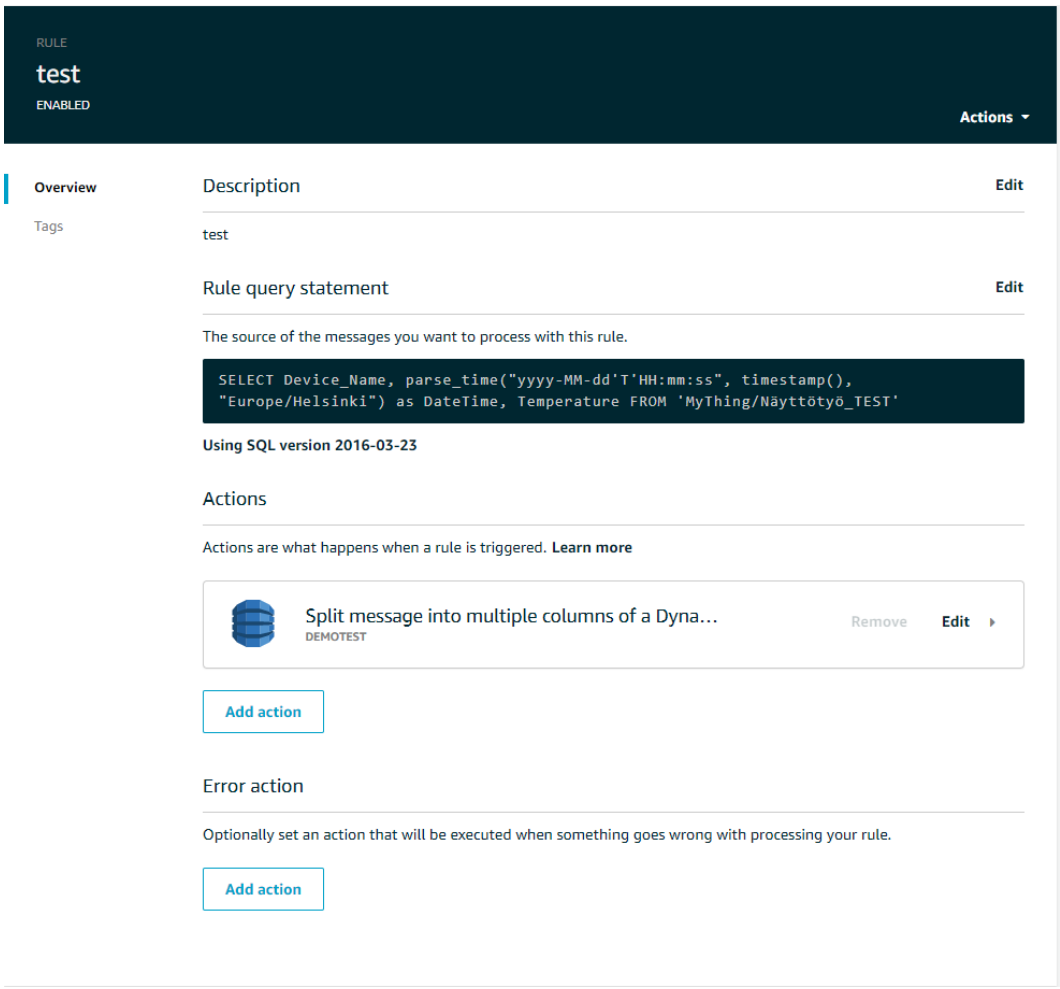


Kuva 23. Tilaaja on onnistuneesti vastaanottanut viestipaketin.

7.2 DynamoDB-tietokanta

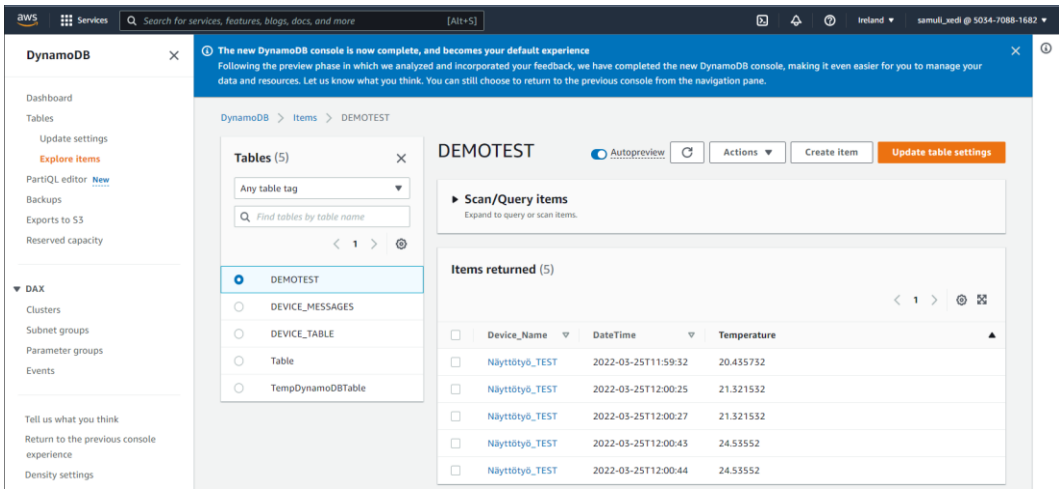
Opinnäytetyön raportin kirjoitusvaiheessa ilmeni jatkokehitysidea järjestelmälle. Tarkoituksena oli tallentaa AWS-pilveen lähetetyt arvot DynamoDB-tietokantaan. Prototyyppilevy oli kuitenkin jo palautettu asiakkaalle, joten osuus oli demonstroitava julkaisemalla manuaalisesti lämpötila viestipaketteja käyttämällä AWS demokäyttäjää.

Jotta arvot saataisiin DynamoDB-tietokantaan, oli luotava AWS IoT-sääntö, jonka avulla DynamoDB kuuntelee AWS-IoT:n dataliikennettä. Sääntö sisältää aiheen, mitä DynamoDB kuuntelee. DynamoDB luo aiheelle tietokantataulukon, jonne tallentaa aiheeseen liittyvän payload-sisällön ja ajan, milloinka viestipaketti on saapunut.



Kuva 24. AWS IoT-sääntö, joka määrittää palvelulle kuunneltavan aiheen.

Demonstraatio suoritti julkaise-tilaa menetelmän samaan tapaan, kuin alkuperäinen järjestelmäkin. Demokäyttäjällä julkaistiin saman kaltainen viestipaketti, kuin projektissa luotu järjestelmäkin julkaisi ja aihetta tilattiin käyttäjällä. Julkaistun viestipaketin kulkeutuessa tilaajalle, dynamoDB kuuntelee AWS-palvelun liikennettä ja kopioi viestipaketin tietokantaan ja tallentaa sen.



Kuva 25. DynamoBD-tietokantaan tallentuneet viestipaketit.

8 LOPPUPÄÄTELMÄT

Opinnäytetyön lopputuloksena kustomoidulle prototyyppilevylle kehitettiin toimiva lämpötilaa mittaava IoT-järjestelmä, joka julkaisee dataa AWS-pilvipalveluun. Opinnäytetyöprojekti ja siihen liittyvä tutkimuksen tekeminen opetti minulle paljon FreeRTOS-reaaliaikakäyttöjärjestelmästä, matkapuhelinverkkosysteemeistä, sekä MQTT-viestintäprotokollasta. IoT-järjestelmän kehittäminen kustomoidulle prototyyppilevylle oli huomattavasti haastavampaa, kuin esimerkiksi kehitysalustalle, jonka virheilmoituksiin ja ongelmiin olisi löytynyt netistä huomattavasti enemmän tietoa.

Opinnäytetyön kirjoitusvaiheessa mieleen tuli muutamia jatkokehitysideoita tietosuojan parantamiseen, sekä lisäyksiä MQTT-viestintäprotokollaan. Avaimet ja sertifikaatit, jota järjestelmä käyttää muodostaessa yhteyden AWS-pilvipalvelun kanssa olisi säilöä erillisiin tiedostoihin, joista ohjelma tarvittaessa hakisi ne. Avaimet ja sertifikaatit voitaisiin esimerkiksi lukea SD-kortilta. Tämä olisi huomattavasti tietosuoja ystävällisempi keino, kuin ”kovakoodata” avaimet ja sertifikaatit MQTT:n konfiguraatiotiedostoon. MQTT-viestintäprotokollaan voisi lisätä vielä LWT-viestipaketin, sekä Device_Shadow-tilan.

LÄHTEET

AWS Cloud Essentials Viitattu 15.4.2022 https://aws.amazon.com/getting-started/cloud-essentials/?pg=gs

AWS IoT Viitattu 18.4.2022 https://aws.amazon.com/iot/

Basics of Cellular System. Walchand Institute of Technology Solapur. Viitattu 15.3.2022 https://www.youtube.com/watch?v=GwKyhTl4Wxw

Basic Cellular Communications System Concepts. Viitattu 21.3.2022 https://www.electronics-notes.com/articles/connectivity/cellular-mobile-phone/basic-cellular-system-concept.php

Cellular Telephone Basics. Luettu 14.3.2022 https://web.archive.org/web/20120511005739/http://www.privateline.com/mt_cellbasics/i_introduction/

Client-Server architecture. Viitattu 28.3.2022 https://www.britannica.com/technology/client-server-architecture

FreeRTOS kernel fundamentals Viitattu 8.4.2022 https://docs.aws.amazon.com/freertos/latest/userguide/dev-guide-freertos-kernel.html

Generations of Mobile Networks: Explained. Viittattu 23.3.2022 https://justtaskthales.com/en/generations-mobile-networks-explained/

HL7802 LPWA Module Viitattu 20.4.2022 https://www.sierrawireless.com/iot-solutions/products/hl7802/

Inter-task Communication Viitattu 11.4.2022 http://www.openrtos.net/Inter-Task-Communication.html

Introducing the MQTT Protocol - MQTT Essentials: Part 1. Viitattu 28.3.2022 https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/

Introduction to RTOS - Solution to Part 7 (FreeRTOS Semaphore Example) Viitattu 11.4.2022 https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-7-freertos-semaphore-example/51aa8660524c4daba38cba7c2f5baba7

Mäki, J Elektroniikkainsinööri Lämpötila-anturin kytkentäkaavio 2019.

Multitasking Basics Viitattu 9.4.2022 https://www.freertos.org/implementation/a00004.html

MQTT Basics. Viitattu 25.3.2022 https://www.mathworks.com/help/thingspeak/mqtt-basics.html

MQTT Client and Broker and MQTT Server and Connection Establishment Explained - MQTT Essentials: Part 3 Viitattu 1.4.2022 https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/

MQTT Publish, Subscribe & Unsubscribe - MQTT Essentials: Part 4 Viitattu 4.4.2022 https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/

Publish & Subscribe - MQTT Essentials: Part 2. Viitattu 29.3.2022 https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/

Scheduling Basics Viitattu 9.4.2022 https://www.freertos.org/implementation/a00005.html

Sierra Wireless™ HL78xx AT Commands Interface Guide Viitattu 21.4.2022 Ladattavissa https://source.sierrawireless.com/resources/airprime/software/hl78xx_at_commands_interface_guide/

STM32CubeIDE Luettu 6.4.2022 Ladattavissa: https://www.st.com/en/development-tools/stm32cubeide.html

Task Communication Viitattu 11.4.2022 https://onlinedocs.microchip.com/pr/GUID-F3CEAE3B-C3C1-4B92-B031-4E07B8ACCD81-en-US-3/index.html?GUID-1CA16B63-3810-403A-8225-289F3234DF39

TCP IP -mikä se on, mihin sitä tarvitaan ja mitä se tekee? Viitattu 31.03.2022 https://nordvpn.com/fi/blog/tcp-ip-protokolla/

Understanding TCP Socket With Examples. Viitattu 25.3.2022 https://www.howtouselinux.com/post/tcp-socket

What is An RTOS? Viitattu 7.4.2022 https://www.freertos.org/about-RTOS.html

What is Cellular Communications: Mobile Technology. Viitattu 23.3.2022 https://www.electronics-notes.com/articles/connectivity/cellular-mobile-phone/what-is-cellular-communications.php

What is IaaS? Viitattu 16.4.2022 https://azure.microsoft.com/en-us/overview/what-is-iaas/#overview

What is Multitasking? Viitattu 9.4.2022 http://fastbitlab.com/what-is-multitasking/

LIITTEET

LIITE 1 Cellular-moduulin käyttöönotto funktio

```
/* FreeRTOS Cellular Common Library porting interface. */
/* coverity[misra_c_2012_rule_8_7_violation] */
CellularError_t Cellular_ModuleEnableUE( CellularContext_t * pContext )
{
    CellularHandle_t * cellularHandle = ( CellularContext_t * ) pContext;
    CellularError_t cellularStatus = CELLULAR_SUCCESS;
    bool radioBandStatus = false;
    CellularAtReq_t atReqGetNoResult =
    {
        NULL,
        CELLULAR_AT_NO_RESULT,
        NULL,
        NULL,
        NULL,
        0
    };
    CellularAtReq_t atReqGetWithResult =
    {
        NULL,
        CELLULAR_AT_MULTI_WO_PREFIX,
        NULL,
        NULL,
        NULL,
        0
    };

    if( pContext != NULL )
    {
        /* Disable echo. */
        atReqGetWithResult.pAtCmd = "ATE0";
        cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                        &atReqGetWithResult );

        if( cellularStatus == CELLULAR_SUCCESS )
        {
           /* Disable DTR function. */
            atReqGetNoResult.pAtCmd = "AT&D0";
            cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                            &atReqGetNoResult );
        }

        if( cellularStatus == CELLULAR_SUCCESS )
        {
            /* Enable RTS/CTS hardware flow control. */
            atReqGetNoResult.pAtCmd = "AT+IFC=2,2";
            cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                            &atReqGetNoResult );
        }

        /* Disable standalone sleep mode. */
        if( cellularStatus == CELLULAR_SUCCESS )
        {
           atReqGetNoResult.pAtCmd = "AT+KSLEEP=2";
           cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                           &atReqGetNoResult );
        }
```

```
        /* Force initialization of radio to consider new configured bands. */
        if( cellularStatus == CELLULAR_SUCCESS )
        {

            cellularStatus = _Cellular_GetRadioBandStatus( cellularHandle,
                                                           &radioBandStatus );
            if( ( cellularStatus == CELLULAR_SUCCESS ) &&
                ( radioBandStatus == false ) )
            {
                atReqGetNoResult.pAtCmd = "AT+KSRAT=2";
                cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                                &atReqGetNoResult );
            }


        }
        if( cellularStatus == CELLULAR_SUCCESS )
        {
         atReqGetNoResult.pAtCmd = "AT+CFUN=1,1";
         cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                         &atReqGetNoResult );
        }
        Platform_Delay( CELLULAR_HL7802_RESET_DELAY_MS );
        /* Disable echo after reboot device. */
        if( cellularStatus == CELLULAR_SUCCESS )
        {
          atReqGetWithResult.pAtCmd = "ATE0";
          cellularStatus = sendAtCommandWithRetryTimeout( pContext,
                                                          &atReqGetWithResult );
        }
    }

    return cellularStatus;
}
```

LIITE 2 Matkapuhelinverkon pystytys funktio.

```
bool setupCellular( void )
{
    bool cellularRet = true;
    CellularError_t cellularStatus = CELLULAR_SUCCESS;
    CellularSimCardStatus_t simStatus = { 0 };
    CellularCommInterface_t * pCommIntf = &CellularCommInterface;
    uint8_t tries = 0;
    CellularPdnConfig_t pdnConfig = { CELLULAR_PDN_CONTEXT_IPV4,
                                      CELLULAR_PDN_AUTH_NONE, CELLULAR_APN, "", "" };
    CellularPdnStatus_t PdnStatusBuffers = { 0 };
    char localIP[ CELLULAR_IP_ADDRESS_MAX_SIZE ] = { '\0' };
    uint8_t NumStatus = 1;


    /* Initialize Cellular Comm Interface. */
    cellularStatus = Cellular_Init( &CellularHandle, pCommIntf );



    if( cellularStatus == CELLULAR_SUCCESS )
    {
        /* wait until SIM is ready */
        for( tries = 0; tries < CELLULAR_MAX_SIM_RETRY; tries++ )
        {
            cellularStatus = Cellular_GetSimCardStatus( CellularHandle,
                                                        &simStatus );

            if( ( cellularStatus == CELLULAR_SUCCESS ) &&
                ( ( simStatus.simCardState == CELLULAR_SIM_CARD_INSERTED ) &&
                  ( simStatus.simCardLockState == CELLULAR_SIM_CARD_READY ) ) )
            {
                configPRINTF( ( ">>>  Cellular SIM okay  <<<\r\n" ) );
                break;
            }
            else
            {
            configPRINTF((">>> Cellular SIM card state %d, Lock State %d <<<\r\n",
                          simStatus.simCardState,
                          simStatus.simCardLockState ) );
            }

            vTaskDelay( pdMS_TO_TICKS( CELLULAR_SIM_CARD_WAIT_INTERVAL_MS ) );
        }
    }

    /* Setup the PDN config. */
    if( cellularStatus == CELLULAR_SUCCESS )
    {
        cellularStatus = Cellular_SetPdnConfig( CellularHandle,
                         CellularSocketPdnContextId, &pdnConfig );
    }
    else
    {
        configPRINTF( ( ">>>  Cellular SIM failure  <<<\r\n" ) );
    }
```

```
    if( cellularStatus == CELLULAR_SUCCESS )
    {
        cellularStatus = Cellular_ActivatePdn( CellularHandle,
                                               CellularSocketPdnContextId );
    }

    if( cellularStatus == CELLULAR_SUCCESS )
    {
        cellularStatus = Cellular_GetIPAddress( CellularHandle,
                         CellularSocketPdnContextId, localIP, sizeof( localIP ) );
    }

    if( cellularStatus == CELLULAR_SUCCESS )
    {
        cellularStatus = Cellular_GetPdnStatus( CellularHandle, &PdnStatusBuffers,
                                                CellularSocketPdnContextId, &NumStatus );
    }


    if( ( cellularStatus == CELLULAR_SUCCESS ) && ( PdnStatusBuffers.state == 1 ))
    {
        cellularRet = true;
    }
    else
    {
        cellularRet = false;
    }

    return cellularRet;
}
```

LIITE 3 MQTT-taski funktio

```
void RunMQTTAppTask( void * pvParameters ){


        uint32_t  ulTopicCount = 0U, ulPublishCount = 0U;
        const uint32_t ulMaxPublishCount = 5UL;
        NetworkContext_t xNetworkContext = { 0 };
        NetworkCredentials_t xNetworkCredentials = { 0 };
        MQTTContext_t xMQTTContext = { 0 };
        MQTTStatus_t xMQTTStatus;
        TlsTransportStatus_t xNetworkStatus;

        /* Remove compiler warnings about unused parameters. */
        ( void ) pvParameters;

        /* Set the entry time of the demo application. This entry time will be
         * used to calculate relative time elapsed in the execution of the demo
         * application, by the timer utility function that is provided to the MQTT
         * library.
         */
        ulGlobalEntryTimeMs = prvGetTimeMs();

        #ifdef USE_1NCE_ZERO_TOUCH_PROVISIONING
            uint8_t status = nce_onboard( &pThingName,
                                          &pEndpoint,
                                          &pExampleTopic,
                                          &pRootCA,
                                          &pClientCert,
                                          &pPrvKey );
            configASSERT( status == EXIT_SUCCESS );
        #else
            pThingName = democonfigCLIENT_IDENTIFIER;
            pEndpoint = democonfigMQTT_BROKER_ENDPOINT;
            pExampleTopic = mqttexampleTOPIC;
            pAlarmTopic   = mqttalarmTOPIC;
        #endif /* ifdef USE_1NCE_ZERO_TOUCH_PROVISIONING */

        for( ulTopicCount = 0; ulTopicCount < mqttexampleTOPIC_COUNT;
             ulTopicCount++ )
        {
            xTopicFilterContext[ ulTopicCount ].pcTopicFilter = pExampleTopic;
        }

        for( ulTopicCount = 0; ulTopicCount < mqttalarmTOPIC_COUNT;
             ulTopicCount++)
        {
            aTopicFilterContext[ ulTopicCount ].pcTopicFilter = pAlarmTopic;
        }


        for( ; ; )
        {
            if(cellularSetup.cellularFlag == true)
            {
            ADC_Stop();


          /****************************** Connect. ******************************/

            /* Attempt to establish TLS session with MQTT broker. If connection
```

```
     * fails,retry after a timeout. Timeout value will be exponentially
     * increased until  the maximum number of attempts are reached or the
     * maximum timeout value is reached. The function returns a failure
     * status if the TCP connection cannot be established to the broker
     * after the configured number of attempts. */

    xNetworkStatus = prvConnectToServerWithBackoffRetries
                 ( &xNetworkCredentials, &xNetworkContext );

    configASSERT( xNetworkStatus == TLS_TRANSPORT_SUCCESS );
   /* Sends an MQTT Connect packet over the already established TLS
    * connection, and waits for connection acknowledgment (CONNACK)
    * packet. */

    prvCreateMQTTConnectionWithBroker( &xMQTTContext, &xNetworkContext );

/****************** Publish and Keep Alive Loop. *********************/
/* Publish messages with QoS1, send and process Keep alive messages. */

        for(ulPublishCount = 0; ulPublishCount < ulMaxPublishCount;
            ulPublishCount++){

            prvMQTTPublishToTopic( &xMQTTContext, &temperature);
            xMQTTStatus = MQTT_ProcessLoop( &xMQTTContext,
                          mqttexamplePROCESS_LOOP_TIMEOUT_MS );

            /* Leave Connection Idle for some time */
            vTaskDelay( mqttexampleDELAY_BETWEEN_PUBLISHES_TICKS );

        }

/*************************** Disconnect. ***************************/

        /* Send an MQTT Disconnect packet over the already connected TLS
         * overTCP connection. There is no corresponding response for the
         * disconnect packet. After sending disconnect, client must close
         * the network connection. */

         xMQTTStatus = MQTT_Disconnect( &xMQTTContext );


         configASSERT( xMQTTStatus == MQTTSuccess );

         /* Close the network connection.  */
         TLS_FreeRTOS_Disconnect( &xNetworkContext );
         /* Reset SUBACK status for each topic filter after completion of
          * subscription request cycle. */

         for( ulTopicCount = 0; ulTopicCount < mqttexampleTOPIC_COUNT;
              ulTopicCount++ )
         {
            xTopicFilterContext[ ulTopicCount ].xSubAckStatus =
            MQTTSubAckFailure;
         }
         for( ulTopicCount = 0; ulTopicCount < mqttalarmTOPIC_COUNT;
              ulTopicCount++ )
         {
             aTopicFilterContext[ ulTopicCount ].xSubAckStatus =
             MQTTSubAckFailure;
         }
```

```
                ADC_Run();

        /* Wait for some time between two iterations to ensure that we do not
         * bombard the broker. */
         vTaskDelay( mqttexampleDELAY_BETWEEN_DEMO_ITERATIONS_TICKS );
       }
         vTaskDelay( mqttexampleDELAY_BETWEEN_DEMO_ITERATIONS_TICKS );
       }


    }
```

LIITE 4 TLS_FreeRTOS_Connect funktio

```
TlsTransportStatus_t TLS_FreeRTOS_Connect( NetworkContext_t * pNetworkContext,
                                           const char * pHostName,
                                           uint16_t port,
                                           const NetworkCredentials_t
                                                *pNetworkCredentials,
                                           uint32_t receiveTimeoutMs,
                                           uint32_t sendTimeoutMs )
{
    TlsTransportStatus_t returnStatus = TLS_TRANSPORT_SUCCESS;
    BaseType_t socketStatus = 0;

    if( ( pNetworkContext == NULL ) ||
        ( pHostName == NULL ) ||
        ( pNetworkCredentials == NULL ) )
    {
        LogError( ( "Invalid input parameter(s): Arguments cannot be NULL.
                    pNetworkContext=%p, "
                    "pHostName=%p, pNetworkCredentials=%p.",
                    pNetworkContext,
                    pHostName,
                    pNetworkCredentials ) );
        returnStatus = TLS_TRANSPORT_INVALID_PARAMETER;
    }
    else if( ( pNetworkCredentials->pRootCa == NULL ) )
    {
        LogError( ( "pRootCa cannot be NULL." ) );
        returnStatus = TLS_TRANSPORT_INVALID_PARAMETER;
    }
    else
    {
        /* Empty else for MISRA 15.7 compliance. */
    }

    /* Establish a TCP connection with the server. */
    if( returnStatus == TLS_TRANSPORT_SUCCESS )
    {
        socketStatus = Sockets_Connect( &( pNetworkContext->tcpSocket ),
                                        pHostName,
                                        port,
                                        receiveTimeoutMs,
                                        sendTimeoutMs );

        if( socketStatus != 0 )
        {
```

```
            LogError( ( "Failed to connect to %s with error %d.",
                        pHostName,
                        socketStatus ) );
            returnStatus = TLS_TRANSPORT_CONNECT_FAILURE;
        }
    }

    /* Initialize mbedtls. */
    if( returnStatus == TLS_TRANSPORT_SUCCESS )
    {
        returnStatus = initMbedtls( &( pNetworkContext
                                       ->sslContext.entropyContext ),
                                    &( pNetworkContext
                                       ->sslContext.ctrDrgbContext ) );
    }

    /* Initialize TLS contexts and set credentials. */
    if( returnStatus == TLS_TRANSPORT_SUCCESS )
    {
        returnStatus = tlsSetup( pNetworkContext, pHostName,
                                 pNetworkCredentials );
    }

    /* Perform TLS handshake. */
    if( returnStatus == TLS_TRANSPORT_SUCCESS )
    {
        returnStatus = tlsHandshake( pNetworkContext, pNetworkCredentials);
    }

    /* Clean up on failure. */
    if( returnStatus != TLS_TRANSPORT_SUCCESS )
    {
        if( pNetworkContext != NULL )
        {
            sslContextFree( &( pNetworkContext->sslContext ) );

            if( pNetworkContext->tcpSocket != NULL )
            {
                ( void ) Sockets_Disconnect( pNetworkContext->tcpSocket );
            }
        }
    }
    else
    {
        LogInfo( ( "(Network connection %p) Connection to %s established.",
                   pNetworkContext,
                   pHostName ) );
    }

    return returnStatus;
}
```

LIITE 5 Funktio, jolla muodostetaan MQTT-viestintäyhteyden välittäjän kanssa.

```
static void prvCreateMQTTConnectionWithBroker( MQTTContext_t * pxMQTTContext,
                                               NetworkContext_t
                                             * pxNetworkContext )
{
    MQTTStatus_t xResult;
    MQTTConnectInfo_t xConnectInfo;
    bool xSessionPresent;

    /***
     * For readability, error handling in this function is restricted to the
     * use of asserts().
     ***/


    /* Some fields are not used in this demo so start with everything at 0. */
    ( void ) memset( ( void * ) &xConnectInfo, 0x00, sizeof( xConnectInfo ) );
    ( void ) memset( ( void * ) &xMQTTPublishInfo, 0x00,
                                                   sizeof( xMQTTPublishInfo ) );

    /* Start with a clean session i.e. direct the MQTT broker to discard any
     * previous session data. Also, establishing a connection with clean
     * session will ensure that the broker does not store any data when this
     * client gets disconnected. */
    xConnectInfo.cleanSession = true;

    /* The client identifier is used to uniquely identify this MQTT client to
     * the MQTT broker. In a production device the identifier can be something
     * unique, such as a device serial number. */
    xConnectInfo.pClientIdentifier = pThingName;
    xConnectInfo.clientIdentifierLength = ( uint16_t ) strlen( pThingName );

    /* Set MQTT keep-alive period. If the application does not send packets at
     * an interval less than the keep-alive period, the MQTT library will send
     * PINGREQ packets. */
    xConnectInfo.keepAliveSeconds = mqttexampleKEEP_ALIVE_TIMEOUT_SECONDS;

    /* Append metrics when connecting to the AWS IoT Core broker. */
    #ifdef democonfigUSE_AWS_IOT_CORE_BROKER
        #ifdef democonfigCLIENT_USERNAME
            xConnectInfo.pUserName = CLIENT_USERNAME_WITH_METRICS;
            xConnectInfo.userNameLength = ( uint16_t ) strlen
                                    ( CLIENT_USERNAME_WITH_METRICS );
            xConnectInfo.pPassword = democonfigCLIENT_PASSWORD;
            xConnectInfo.passwordLength = ( uint16_t )
                                strlen( democonfigCLIENT_PASSWORD );
        #else
            xConnectInfo.pUserName = AWS_IOT_METRICS_STRING;
            xConnectInfo.userNameLength = AWS_IOT_METRICS_STRING_LENGTH;
            /* Password for authentication is not used. */
            xConnectInfo.pPassword = NULL;
            xConnectInfo.passwordLength = 0U;
        #endif
    #else /* ifdef democonfigUSE_AWS_IOT_CORE_BROKER */
        #ifdef democonfigCLIENT_USERNAME
            xConnectInfo.pUserName = democonfigCLIENT_USERNAME;
            xConnectInfo.userNameLength = ( uint16_t )
                                strlen( democonfigCLIENT_USERNAME );
            xConnectInfo.pPassword = democonfigCLIENT_PASSWORD;
            xConnectInfo.passwordLength = ( uint16_t )
```

```
                                strlen( democonfigCLIENT_PASSWORD );
            #endif /* ifdef democonfigCLIENT_USERNAME */
        #endif /* ifdef democonfigUSE_AWS_IOT_CORE_BROKER */

        /* Send MQTT CONNECT packet to broker. LWT is not used in this demo, so it
         * is passed as NULL. */


             xResult = MQTT_Connect( pxMQTTContext,
                                     &xConnectInfo,
                                     &xMQTTPublishInfo,
                                     mqttexampleCONNACK_RECV_TIMEOUT_MS,
                                     &xSessionPresent );


        configASSERT( xResult == MQTTSuccess );

        /* Successfully established and MQTT connection with the broker. */
        LogInfo( ( "An MQTT connection is established with %s.", pEndpoint ) );
    }
    /*-----------------------------------------------------------*
```