jamk.fi

MQTT IoT-protokolla

Toiminta ja toteutus

Rami Ojala

Opinnäytetyö
Joulukuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), tieto- ja viestintätekniikka
Tietoverkkoturvallisuus ja teollinen internet

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Ojala, Rami	Opinnäytetyö, AMK	Joulukuu 2017
	Sivumäärä 109	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
MQTT IoT-Protokolla
Toiminta ja toteutus

Tutkinto-ohjelma
Insinööri (AMK), tieto- ja viestintätekniikka

Työn ohjaaja(t)
Väänänen, Olli

Toimeksiantaja(t)

Tiivistelmä

Opinnäytetyössä perehdyttiin IoT-laitteiden kommunikointiin, tehokkaan kommunikoinnin merkitykseen ja lähitulevaisuuden haasteisiin kommunikoinnissa. Toteuttavana osuutena opinnäytetyössä suunniteltiin ja toteutettiin MQTT-protokollan mukainen avoimen lähdekoodin asiakasohjelmisto.

IoT-laitteiden määrän ennustettu räjähdysmäinen kasvu asettaa erityisiä vaatimuksia kommunikointiin tarkoitetulle protokollalle. Osa vaatimuksista on esitelty ja kuvailtu yksityiskohtaisesti tässä opinnäytetyössä. Erityisesti protokollien vaatimien eri otsakkeiden suhde hyötykuormaan on tärkeässä roolissa IoT-laitteiden suhteellisen pienissä tietomäärissä. Opinnäytetyö esitteli protokollan otsakkeen tyypillisessä IoT-laitteen käyttötapauksessa.

IoT-laitteiden kommunikointiin yleisesti käytetyistä protokollista esitettiin opinnäytetyössä perustoiminnallisuuksien kuvaus. Tutkimusajankohtana yleisesti käytössä olevia protokollia olivat: AMQP, CoAP, HTTP, MQTT ja XMPP. Opinnäytetyössä kuvailtiin protokollien otsakkeita sekä sitä, millaisen alemman kerroksen protokollat vaativat. Lisäksi kerrottiin protokollan toiminnasta ja siitä, minkälaista hyötykuormaa sillä voi siirtää. Lisäksi opinnäytetyössä kuvailtiin IoT-laitteiden tietoturvaa TCP/IP-protokollapinon kuljetuskerroksen ja verkkokerroksen osalta.

Protokollista MQTT-protokollan viestirakenne, toiminta ja siihen liittyvät ohjelmistot esitettiin yksityiskohtaisesti. Esityksen perusteella määriteltiin MQTT-asiakasohjelmiston arkkitehtuuri ja testitapaukset. Näiden pohjalta toteutettiin MQTT versiota 3.1.1 noudattava asiakasohjelmisto. Ohjelmistoon on valittavissa joko dynaaminen tai staattinen muistinvaraus. IoT-laitteet suosivat staattista muistinvarausta, joka mahdollistaa ohjelmiston käytön kevyissä ympäristöissä.

Avainsanat (asiasanat)
AMQP, CoAP, esineiden internet, IoT, IoT-protokolla, MQTT, XMPP

Muut tiedot
Asiakasohjelmisto https://github.com/rojala/ROjal_MQTT_Client

jamk.fi

Description

Author(s)	Type of publication	Date
Ojala, Rami	Bachelor's thesis	December 2017
		Language of publication: Finnish
	Number of pages 109	Permission for web publication: x

Title of publication
MQTT IoT protocol
Functionality and implementation

Degree programme
Information and communication technology

Supervisor(s)
Väänänen, Olli

Assigned by

Abstract

The thesis focuses on the communication, significance of efficient communication and communicational challenges of IoT devices in the near future. An open source client application that is MQTT protocol compatible was designed and implemented as the applied part of the thesis.

The predicted massive increase of the number of IoT devices sets specific requirements on protocols used for communication. The thesis presents and describes some of the requirements in detail. In particular, the ratio between the payload and the different headers required by IoT protocols plays a significant role in the relatively small amounts of data being transferred by IoT devices. The thesis includes an example of an overhead of a protocol in a common IoT device use case.

The thesis presents an overview of the basic functionalities of the most commonly used IoT communication protocols. At the time of the study, the most commonly used IoT protocols were: AMQP, CoAP, HTTP, MQTT and XMPP. The overview focused on the headers and the kind of an underlaying level the protocols require. In addition, the protocol's functionality and the ability to carry payload is discussed in the thesis. Network layer and transport layer security solutions using TCP/IP protocol stack are also presented.

Of all the covered protocols, MQTT protocol's data format, functionality and related programs were presented in detail. MQTT (v. 3.1.1) client application's architecture, implementation and use cases were specified and implemented based on the presented details. The application's memory management can use either static or dynamic memory allocation. Usually simple IoT devices prefer static allocation, which enables the use of the application in lightweight environments.

Keywords/tags (subjects)
AMQP, CoAP, Internet of Things, IoT, IoT protocol, MQTT, XMPP

Miscellaneous
Client application https://github.com/rojala/ROjal_MQTT_Client

Sisältö

Kuviot

Taulukot

Lyhenteet

AES	Advanced Encryption Standard
AH	Authentication Header
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
DTD	Document Type Definition
DTLS	Datagram TLS
ESP	Encapsulation Security Payload
HMAC	Hashed Message Authentication Code
HTML	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IEFT	Internet Engineering Task Force
IIoT	Industrial Internet of Things
IKE	Internet Key Exchange
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
MAC	Lähiverkko: Media Access Control
MAC	Tietoturva: Message Authentication Code
MQTT	Message Queue Telemetry Transport
MQTT-SN	MQTT-Sensor Network
OASIS	Organization for the Advancement of Structured Information Standards
OSI	Open Systems Interconnection
REST	Representation State Transfer
RFID	Radio Frequency Identification
SASL	Simple Authentication and Security Layer
SSL	Secure Socket Layer
STOMP	The Simple Text Oriented Messaging Protocol
TCP	Transmission Control Protocol
TCP	Transmission Control Protocol

TLS	Transport Layer Security
UDP	User Datagram Protocol
VPN	Virtual Private Network
XML	Extensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

1 Opinnäytetyön aihe ja tavoitteet

Opinnäytetyö kuvaa työn hetkellä käytössä olevat keskeiset IoT-protokollat yleisellä tasolla. Työssä esitetään IoT-laitteiden välisen kommunikoinnin merkitys tulevaisuudessa. IoT-protokollien kuvauksesta ilmenevät protokollien tärkeimmät ominaisuudet sekä niiden väliset eroavaisuudet. MQTT-protokolla esitellään yksityiskohtaisesti ja opinnäytetyön osana toteutettiin MQTT-asiakasohjelmisto. Ohjelmiston tavoitteena oli mahdollistaa MQTT-protokollan käyttö eri ympäristöissä ja erityisesti IoT-laitteissa.

2 IoT ja kommunikointiprotokolla

2.1 IoT yleisesti

Vuonna 1999 Protect & Gamble-yhtiölle RFID-teknologian mahdollisuuksiin keskittyvässä esityksessä Kevin Ashton esitteli tiettävästi ensimmäisen kerran termin ”Internet Of Things” (IoT) (Ashton 2009). Nykyään IoT-termi tunnetaan myös termillä esineiden internet, mutta yksiselitteistä määritystä IoT-teknologialle ja termille ei kuitenkaan ole.

Vuonna 2015 IoT oli tunnetulla Gartnerin teknologiakäyrällä ylimmällä mahdollisella tasolla, mikä ennusti IoT:n siirtyvän kohti kaupallistumista lähivuosina (Gartner's 2015 hype -- 2015). Teknologiat yhdistyvät, ja todennäköisesti myös erilaisia IoT-teknologioita tuottavat yhtiöt tulevat fuusioitumaan tulevaisuudessa. Kehityksen seurauksena loppukäyttäjät saavat tulevaisuudessa valmiimpia tuotteita, jotka toimivat paremmin keskenään. Teknologiakäyrän perusteella voidaan olettaa myös, että laitteiden hinnat laskevat ja laitteiden käyttöikä pitenee. Todennäköisesti loppukäyttäjä ei jossain vaiheessa enää huomaa tai huomioi kaikkia itseään ympäröiviä IoT-laitteita.

Google-hakupalvelulla suoritettujen IoT-hakujen suhteellinen osuus aikaisempiin hakumääriin näytti, että vuodesta 2015 hakujen määrä on yli kolminkertaistunut, ja se on saavuttanut huippunsa 2017 (Google Trends 2017). Hakutulokset sekä teknologiakäyrä antavat samansuuntaisen viitteen, eli käännekohta kohti yleisesti tunnettua teknologiaa ja termiä olisi tapahtunut vuoden 2015 aikana.

Suuri ohjelmisto- ja analytiikkayhtiö SAS teetti analyysin IoT-teknologian kehityksestä. Analyysin perusteella IoT-teknologia olisi saavuttamassa kuluttajat vuonna 2020 (Internet of Things: Visualise -- 2016). Googlen tilastot, edellä mainittu teknologiakäyrä ja analyysi viittaavat kaikki IoT-laitteiden määrän merkittävään kasvuun. Laitteiden merkittävä lisääntyminen asettaa haasteita myös IoT-laitteiden kommunikoinnille. Kommunikointi on yksi IoT-laitteiden tärkeimmistä ominaisuuksista. Näistä lähtökodista opinnäytetyön aihe on ajankohtainen, koska se pureutuu juuri IoT-laitteiden kommunikaatioon.

2.2 IoT:n määritelmä

IEEE-järjestö aloitti IoT:n määrityksen kuvaamisen, mutta toistaiseksi vuoden 2015 määritystä ei ole päivitetty (Towards a definition -- 2015). SAS-instituutin tekemän analyysin perusteella määritys riippuu käsiteltävästä aihepiiristä (Internet of Things: Visualise – 2016, 5).

Yleisesti voidaan kuitenkin ajatella, että IoT-teknologia lisää älykkyyttä ja internet-yhteyden olemassa oleviin laitteisiin. Tällaisia laitteita ovat esimerkiksi televisiot, lämpötila-anturit, lämpötilatermostaatit, robotti-imurit, autot, polkupyörät. Uusia markkinoita IoT-sovelluksille on viime vuosina ilmestynyt, joista erityisesti hyvinvoinnin seurantaan erikoistuneet kellot ja rannekkeet ovat saavuttaneet suuren suosion. Hyvinvointia seuraavista laitteista tieto siirtyy pääsääntöisesti pilvipalveluun, missä omaa hyvinvointiaan pääsee analysoimaan. Toisaalta IoT-teknologian ominaisuuksia on yritetty liittää laitteisiin, missä niille ei välttämättä ole tarvetta. Tällaisia laitteita ovat esimerkiksi leivänpaahdin ja Twitter-viestejä lähettävä vessanpönttö. Nest Labs -yhtiön kehittämä lämpötilatermostaatti oli tiettävästi ensimmäinen kuluttajakäyttöön suunniteltu ja suuren suosion saavuttanut IoT-laite, joka tarjosi loppukäyttäjälle lisäarvoa säästämällä lämmityskuluissa. Laite mahdollisti myös lämmityksen etäohjauksen internetin välityksellä.

Teollisuuden IoT-terminä on käytetty ilmaisuja ”IIoT” (Industrial IoT) tai ”teollisuus 4.0” (Industry 4.0). Teollisuudessa on jo jonkin aikaa kerätty tietoja tuotantoprosessin eri vaiheista. Osaa tiedoista on käytetty hyväksi valmistamalla mahdollisimman vähän tavaraa varastoon, mutta kuitenkin riittävästi asiakkaiden tarpeisiin. Teollinen

internet vie automatisointia eteenpäin niin, että asiakkaiden tarpeet ennustetaan ja tuotantoa optimoidaan. Optimoinnissa on useita parametreja, kuten kustannukset, luonnonvarat ja aikataulu. Lopputuotteen osat saattavat valmistua eri tehtailta, ja järjestelmä pitää huolen siitä, että eri komponentit valmistuvat ja saapuvat oikeaan paikkaan oikeaan aikaan. Tuotantolaitteiden rikkoutumiset minimoidaan ennustamalla laitteen toimintatila ja tarvittavat huoltotoimenpiteet. Teollisen internetin voisi kiteyttää niin, että siinä analysoidaan tuotannon ja logistiikan toimivuutta, käyttäjän tarpeita ja kaupallisia olosuhteita ja näiden perusteella tehdään päätöksiä automaattisesti. (Gilchrist 2016.)

2.3 Tulevaisuuden näkymät

Edellisessä luvussa todettiin, että IoT mahdollistaa laitteen saavutettavuuden verkkoyhteyden avustuksella. Teollinen internet taas on isompi kokonaisuus, jossa tuotannon automatisointia viedään merkittävästi eteenpäin erilaisin automatiikan ja analytiikan keinoin.

Älykodit, jossa eri laitteet toimivat autonomisesti mahdollisimman vähäisellä loppukäyttäjän ohjauksella sekä mahdollisimman energiatehokkaasti ovat pienemmän mittakaavan teollisen internetin kuluttajasegmentin ilmentymä. Yhdistämällä älykodit, älykäs liikenne ja kaupungin tietojärjestelmät voidaan luoda älykäs kaupunki. IoT-ratkaisuihin erikoistunut Libelium-yhtiö käytti termiä älykäs maailma, mikä edellistä analogiaa noudattaen yhdistäisi älykaupungit, mutta myös teollisen internetin (Libelium Smart World 2013).

Älykodit ovat tehneet tuloaan jo muutaman vuoden, mutta varsinaista läpimurtoa ei ole vielä tapahtunut. Useilla kaupungeilla on yhteistyötä korkeakoulujen ja yliopistojen kanssa älykaupunkien määrittelemiseksi. Älyliikenne on myös erittäin tutkittu sekä rahoitettu tutkimuksen aihealue. IoT-teknologialla on keskeinen rooli kaikilla edellä mainituilla kehityksen aloilla, joten IoT-laitteiden määrän ennustetaankin kasvavan jopa 30 miljardiin laitteeseen vuoteen 2020 mennessä. Kasvun oletetaan jatkuvan eksponentiaalisesti ja vuonna 2025 on käytössä ennustettu olevan 75 miljardia IoT-laitetta (Columbus 2016). IoT-laitteiden erilaisten prosessien ja tiedon yhdistäminen tunnetaan myös lyhenteellä IoE (Internet of Everything) (Evans 2013). Analyysit

ja tulevaisuuden näkymät tukevat alussa esitettyä ennustetta IoT-laitteiden määrän suuresta kasvusta sekä asettavat lisää paineita IoT-laitteiden kommunikoinnille.

2.4 Kommunikointiprotokollat

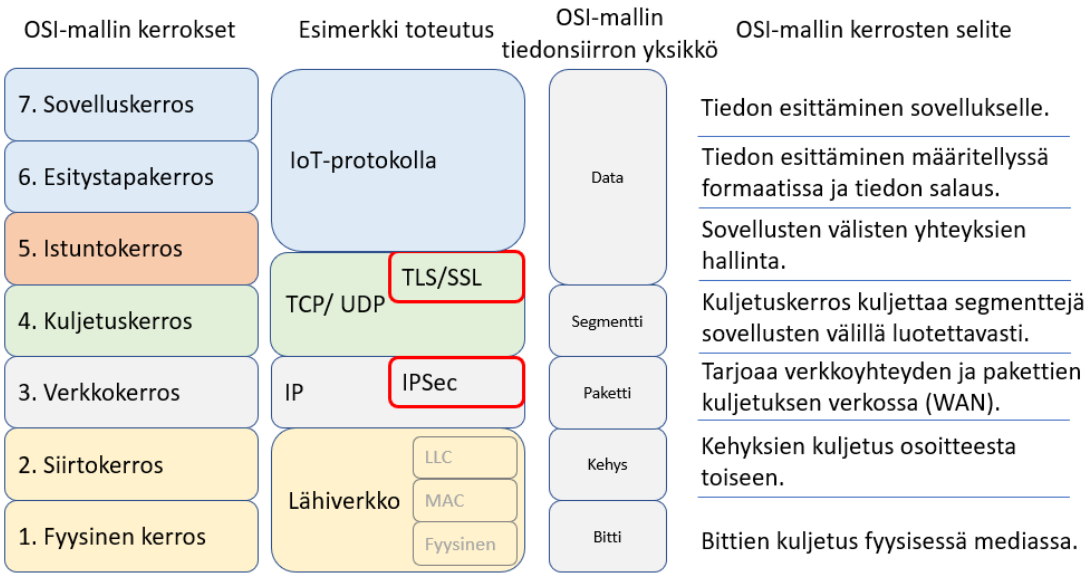
2.4.1 Yleisesti

Protokolla on määritelty kommunikoinnin tapa, joka mahdollistaa eri toimijoiden ja valmistajien laitteiden kommunikoinnin keskenään. IoT-protokolla on tapa, jolla IoT-laitteet kommunikoivat. IoT-laitteille ei toistaiseksi ole yhtenäistä kommunikointiprotokollaa määriteltynä, vaan niitä löytyy useita. Useiden vaihtoehtoisten kommunikointiprotokollien käyttöä selittänee osaltaan se, että IoT-teknologia ei ole vielä laajasti kaupallistettu. Opinnäytetyössä esitellään erilaisia kuviossa 2 esitetyn OSI-mallin sovelluskerroksen protokollia. Erityisen tutkinnan alla oli MQTT-protokolla, josta on muodostunut yksi käytetyimmistä IoT-protokollista.

Kuvio 1 määrittelee IoT-teknologian vaatimuksia Maslow'n tarvehierarkian mukaisesti. IoT-protokolla on määritelty 3. tasolle, mikä tässä tapauksessa tarkoittaa, ettei IoT-laitetta ole olemassa ilman fyysistä laitetta, tietoturvaa ja IoT-protokollaa. (Hunter 2015.)

Kuvio 1. IoT-teknologian tarvehierarkia (Hunter 2015, muokattu)

Kuvion 2 perinteisemmässä OSI-mallissa IoT-protokolla on kerroksella 7 ja lähteestä riippuen myös kerroksilla 6 ja 5. Tyypillisesti TCP toimii kuljettavana kerroksena ja IP verkkokerroksena. Siirtokerrokselle ja fyysiselle kerrokselle löytyy useita muitakin vaihtoehtoja kuin kuvassa esitetty lähiverkko (Ethernet/IEEE 802.3).



Kuvio 2. OSI-kerrokset

2.4.2 IoT-protokollan yleiset vaatimukset

IoT-laitteet ovat usein akkukäyttöisiä ja niissä on rajallinen laskentakapasiteetti. Käyttöajan maksimoimiseksi käytettävä protokolla ei saisi aiheuttaa ylimääräistä prosessointia, vaan ainoastaan tarpeellinen tieto lähetetään mahdollisimman nopeasti. Rajallinen laskentakapasiteetti saattaa aiheuttaa myös ongelmia tietoturvan toteuttamisessa.

TCP-yhteyttä käytettäessä laitteiden välille muodostetaan pysyvä yhteys ja protokolla voi tarkistaa yhteyden toimivuuden elossa-viesteillä (keep alive). UDP-yhteyttä käytettäessä pysyvää yhteyttä ei muodosteta, vaan haluttu tieto lähetetään verkkoon, eikä varmisteta lähetyksen onnistumista. (Dostálek & Kabelová 2006, 319-328, 342-345.)

IoT-laitteiden ennustettu suuri kappalemäärä asettaa haasteen internetprotokollan osoitteistukselle. Internetprotokollan 4. version osoitteet eivät enää riitä palvelemaan ennustettua kymmenien miljardien laitteiden osoittamista. IoT-laitteet tulevat todennäköisesti käyttämään internetprotokollasta versiota 6. Kyseisessä versiossa osoitekentän pituus on nelinkertainen, eli 16 tavua nykyisin yleisesti käytössä olevaan 4. version 4 tavuun verrattuna (Dostálek & Kabelová 2006, 280).

IoT-protokollan täytyy siis olla tehokas ja mahdollisimman vähän ylimääräistä liikennettä tuottava. IoT-protokollan pitää myös pystyä palvelemaan määrällisesti suuria määriä laitteita samanaikaisesti.

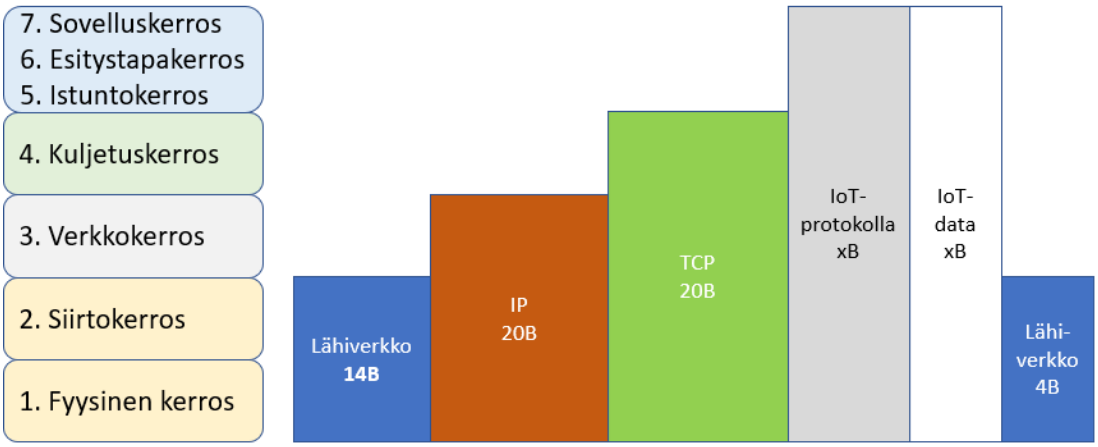
3 IoT-protokollat

3.1 Yleisesti

Suurin osa IoT-protokollista toimii OSI-mallin 5-7 kerroksilla (ks. kuviot 2 ja 3) tarjoten sovelluksille tavan siirtää tietoa. Osa IoT-protokollista toteuttaa myös alempia OSI-mallin kerroksia tai kaikkia OSI-mallin kerroksia. Luvuissa 3.3-3.9 esitetään yleisimmin käytettyjä IoT-protokollia, niiden toimintaa pääpiirteittäin, ominaisuuksia sekä niiden käyttötapoja, -kohteita ja -mahdollisuuksia.

3.2 OSI-mallin kerrokset

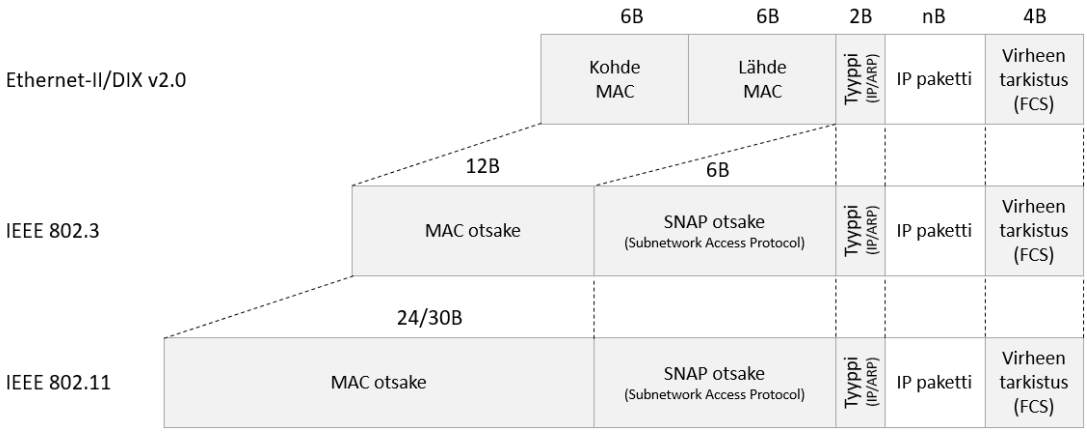
Kuvio 3 kuvaa, miten OSI-mallin eri kerrokset lisäävät oman otsakekentän lähiverkossa siirrettävään tietoon. Mikäli oletetaan sovellustason otsakkeen kooksi 16 tavua ja siirrettävän hyötykuorman kooksi 4 tavua (32-bittinen lukema sensorilta), tällöin lähiverkossa siirrettävä paketti on kooltaan vähintään 78 tavua (4 + 16 + 20 + 20 + 14 + 4). Protokollakerrosten lisäämillä otsaketavuilla on suuri merkitys, erityisesti pienten tietomäärien siirrossa, pienen laskentakapasiteetin omaavilla ja akkukäyttöisillä laitteilla. IoT-protokollia ennen luvuissa 3.2.1-3.2.4 esitellään lyhyesti OSI-mallin muut kerrokset.



Kuvio 3. OSI-kerrokset ja otsakkeet (Designing the Internet of Things n.d., muokattu)

3.2.1 Fyysinen kerros ja siirtokerros

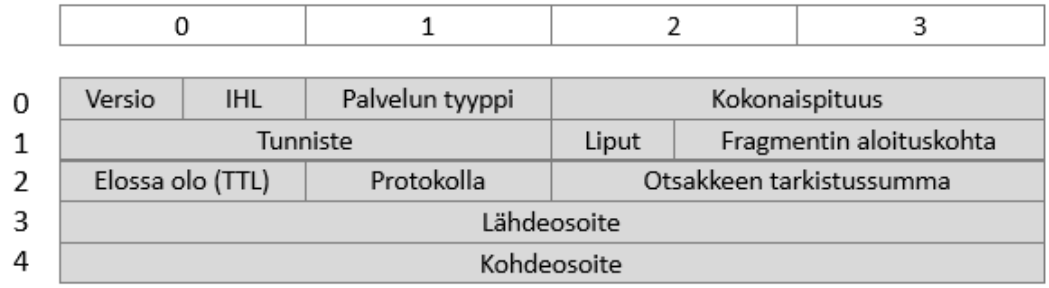
Kuviossa 3 esitetyn OSI-mallin 1-2 kerrokset kuvaavat fyysistä kerrosta ja siirtokerrosta. Kuviossa esitetyn lähiverkon otsakkeen pituus vaihtelee verkkotyypistä riippuen. Kuvio 4 esittää alkuperäisen langallisen lähiverkon (Ethernet-II/DIX v2.0) lisäksi IEEE 802.3 standardin mukaisen kehysrakenteen. IEEE 802.3 on nykyisissä langallisissa lähiverkoissa käytetty kehysrakenne, mikä lisää 6 tavua otsaketietoja alkuperäiseen rakenteeseen (IEEE 802.3 2015). IEEE 802.11 vastaavasti määrittelee langattoman lähiverkon kehysrakenteen (IEEE 802.11 2012).



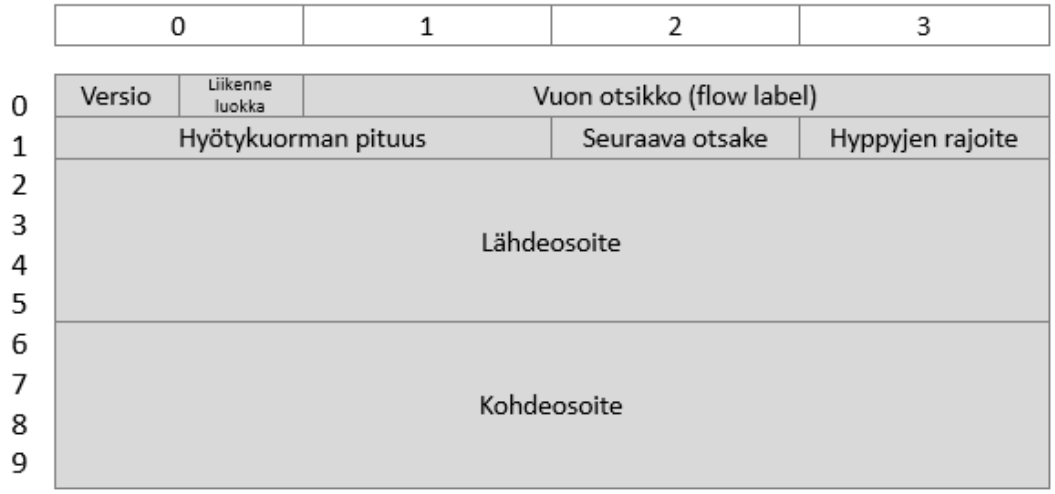
Kuvio 4. Lähiverkon otsakkeet (Gast 2005, muokattu)

3.2.2 Verkkokerros

OSI-mallin 3. kerros on verkkokerros. Verkkokerros sisältää IP-liikenteen tässä tapauksessa. Kuviossa 3 esitetty 20 tavun IPv4-otsake sisältää vain pakolliset kentät. IPv4-otsakkeen pakolliset kentät on esitetty kuviossa 5. IPv4 tarvitsee siis vähintään 20 tavua ja vastaavasti IPv6 40 tavua otsaketietoja (ks. kuvio 6).



Kuvio 5. IPv4 otsake (Huston 2003, muokattu)



Kuvio 6. IPv6 otsake (Huston 2003, muokattu)

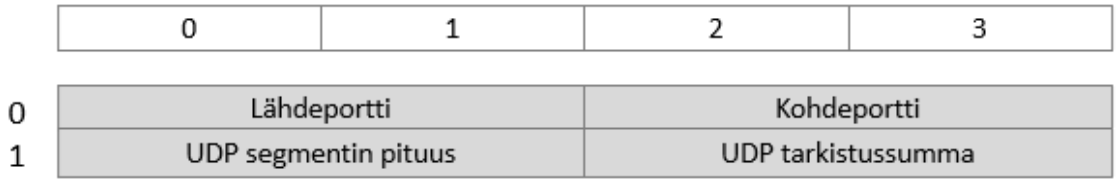
3.2.3 Kuljetuskerros ja siirtokerros

OSI-mallin 4. ja 5. kerros ovat kuljetus- ja siirtokerros. Lähteestä riippuen TCP on vain 4. kerroksessa, osaksi myös 5. kerroksessa tai kokonaan myös 5. kerroksessa. TCP muodostaa luotettavan siirtoyhteyden kahden sovelluksen välille, mikä soveltuisi istuntokerroksen määritelmään (5. kerros). Yhteys sovellusten välille muodostetaan

TCP-porttien avulla (0-65535). Luotettavan siirtoyhteyden takaavat tarkastussumma ja automaattinen uudelleenlähetys. Uudelleenlähetys vaatii sen, että jokainen lähetetty TCP-segmentti (ks. kuvio 7) myös kuitataan vastaanottajan toimesta. Kuvion 3 mukaan TCP-otsake on pituudeltaan 20 tavua. Kaksikymmentä tavua on otsakkeen minimipituus, minkä jälkeen voi olla lisäksi valinnaisia määrityksiä. (Dostálek & Kabelová 2006, 310-342.)

Kuvio 7. TCP-segmentti (Dostálek & Kabelová 2006, 312, muokattu)

Toisin kuin TCP, UDP ei muodosta luotettavaa yhteyttä ohjelmien välille. UDP-segmentit lähetetään vastaanottajalle varmistamatta läpimenoa. UDP-otsakkeen pituus on vain 8 tavua (ks. kuvio 8), eli 12 tavua vähemmän kuin on TCP-protokollan pituus. Otsakkeen lyhemmän pituuden lisäksi UDP-segmentin vastaanottaja ei kuittaa saatua segmenttiä. UDP ei myöskään muodosta pysyvää yhteyttä ohjelmien välille, joten yhteyden olemassaolon tarkistamista ja ylläpitoa ei tarvita. (Dostálek & Kabelová 2006, 343-347; RFC768 1980.)



Kuvio 8. UDP-otsake (Dostálek & Kabelová 2006, 315, muokattu)

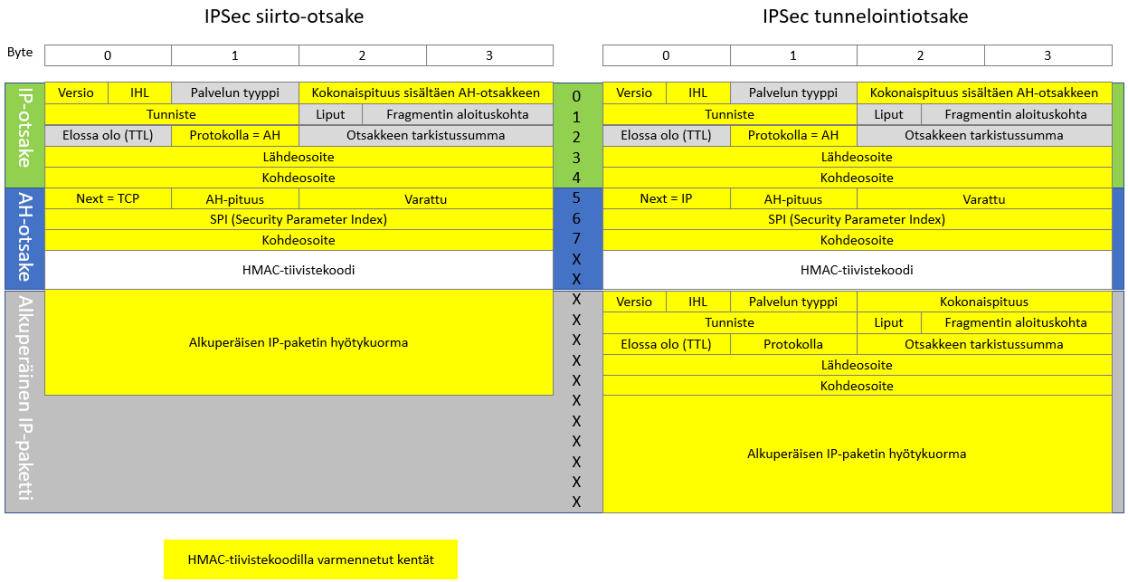
3.2.4 OSI-kerrosten tietoturva

Tietoturva on IoT-laitteille erittäin tärkeä ominaisuus. Tarvehierarkia kuviossa tietoturva oli heti fyysisen laitteen jälkeen seuraavaksi tärkein IoT-laitteen ominaisuus. Tämänhetkisissä IoT-laitteissa on valitettavan paljon tietoturvaongelmia. Euroopan Unioni on määrittelemässä uuteen teleliikennelakiin määrityksiä internettiin kytkettäville laitteelle niiden tietoturvan parantamiseksi (Stupp 2016). Seuraavaksi esitetään TCP/IP-protokollien kaksi yleisintä kommunikointia suojaavaa ratkaisua.

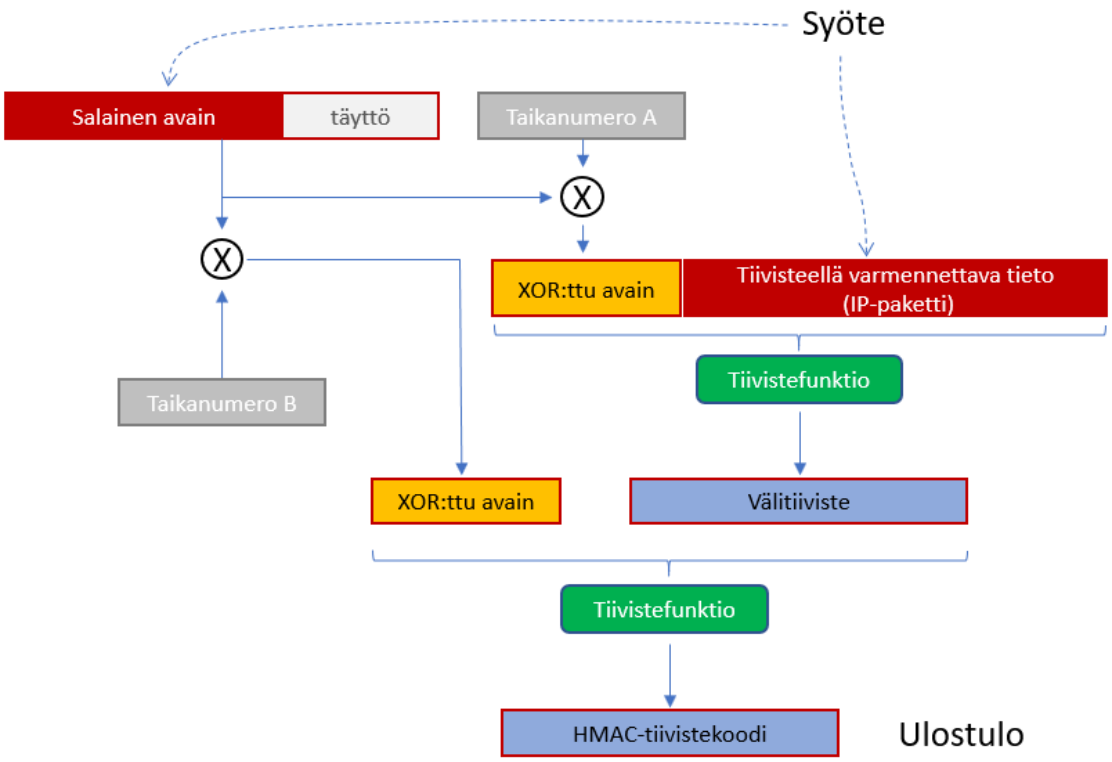
Verkkokerroksen tietoturva (IPSec)

IP-protokollaa käytettäessä voidaan käyttää IPSec-protokollaa suojaamaan yhteys. IPSec on vapaavalintainen ominaisuus IPv4-protokollassa ja pakollinen IPv6-protokollassa. Vaikka IPSec on pakollinen IPv6-protokollassa, sitä ei ole pakko käyttää. IPSec määrittelee kaksi toimintatapaa, joista ensimmäinen lisää tarvittavat otsakkeet tietoturvan mahdollistamiseksi ja toinen lisää uuden IP-otsakkeen tunnelointia varten. Tunnelointi tarkoittaa käytännössä VPN-yhteyttä. (Perez 2014, 77-108.)

AH-otsake (Authentication Header) määrittelee hyötykuorman ja IP-otsakkeen oikeellisuuden (todennus ja eheys). AH-otsake lisätään IP-otsakkeen perään, joko alkuperäisen tai tunneli-otsakkeen perään (ks. kuvio 9). AH-otsake sisältää 16 tavua, sekä HMAC-tiivistekoodin (Hashed Message Authentication Code). Tiivistekoodin pituus riippuu käytetystä algoritmista. Esimerkiksi mikäli algoritmina on SHA-1, se vaatii 20 tavua (ks. kuvio 10). IPv6-protokollassa on vastaavanlainen AH-otsake. Tunneloitaessa lisätään uusi IP-otsake ja alkuperäinen IP-otsake siirretään osaksi hyötykuormaa, jolloin lisäystä syntyy vähintään 40 tavua. (Perez 2014, 77-108; RFC4302 2005.)



Kuvio 9. IPv4 IPSec AH-otsake (Friedel 2005, muokattu)



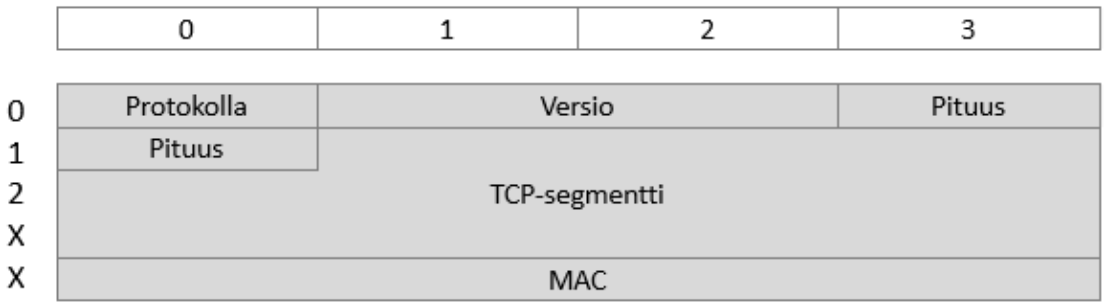
Kuvio 10. HMAC-tiivistekoodin laskenta (Friedel 2005, muokattu)

Salauksen lisäämiseksi tarvitaan ESP-menetelmä (Encapsulation Security Payload). Tunneloitaessa salatun hyötykuorman lisäksi salataan alkuperäinen IP-otsake ja siirto-menetelmässä vain alkuperäisen IP-paketin hyötykuorma, eli kuviossa 9 harmaalla alueella rajattu alue (”Alkuperäinen IP-paketti”). Avainten vaihtoa varten käytetään IKE-standardia. (Perez 2014, 77-108; RFC2409 1998.)

Kuljetuskerroksen tietoturva (TLS, DTLS)

Kuljetuskerroksen tietoturvan tarkoituksena on tarjota suojattu yhteys sovellusten välillä. Kuljetuskerroksen tietoturva TCP/IP-protokollapinossa toteutetaan TLS- ja DTLS-menetelmillä. TLS on jatkokehitetty versio Netscapen alun perin kehittämästä SSL-menetelmästä. SSL versio 3.0 vastaa TLS 1.0 määritystä. TLS-menetelmää käytetään TCP-yhteyksien suojaamiseen ja DTLS-menetelmää UDP-yhteyksille. (Perez 2014, 109-132; RFC5246 2008; RFC6101 2011; RFC6347 2012.)

TLS-menetelmässä sovelluksen tieto sijoitetaan protokollan tietueeseen (record). Tietueen alussa on muutama otsakekenttä, joilla kerrotaan tietueen tyyppi, versio ja viestin pituus. Tietueen lopussa on MAC-kenttä (Message Authentication Code), millä varmistetaan sovelluksen tiedon oikeellisuus (ks. kuvio 11). (Perez 2014, 109-132.)

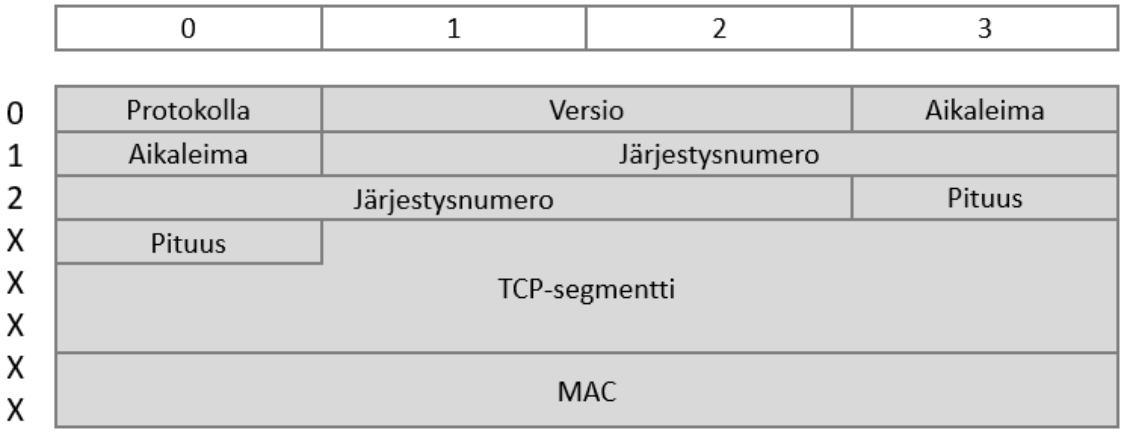


Kuvio 11. TLS tietue (Perez 2014, 153, muokattu)

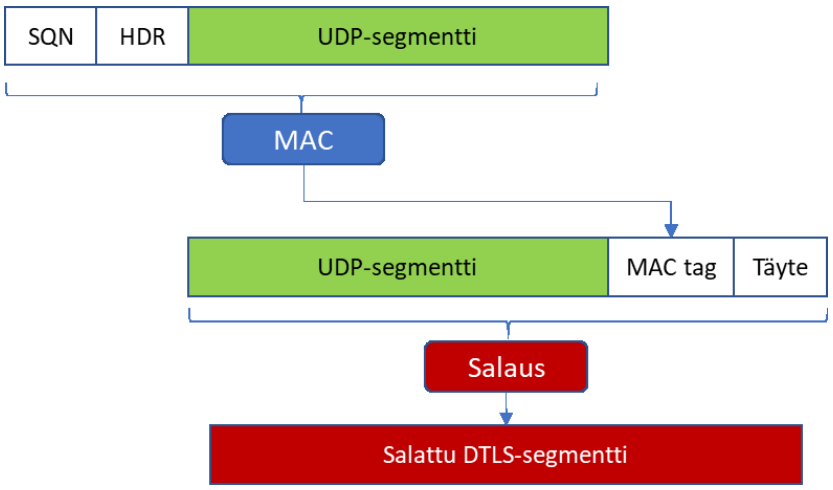
Tiedon oikeellisuuden varmistamiseksi ja salaamiseksi tarvitaan avaintenvaihto mekanismi, jossa käytetään Diffie-Hellman -avaintenvaihtoprotokollaa ja asymmetristä salausta. Avaintenvaihdon jälkeen käytetään esimerkiksi symmetristä AES-salausta. Symmetristä avainta vaihdetaan tietyin määräajoin salauksen parantamiseksi. IPSec

käyttää vastaavaa menetelmää. TLS 1.2 versiossa tiivisteen laskentaan käytetään SHA-256:ta tai pidempää algoritmia. (Perez 2014, 109-132.)

DTLS-menetelmä lisää järjestysnumeron (sequence number, SQN) TLS-tietueeseen, ennen sovelluksen salattua tietoa. Kuvio 12 esittää DTLS-otsakkeet ja kuvio 13 salausprosessin. Järjestysnumero kasvattaa tietueen kokoa kuudella tavulla. Järjestysnumerona käytetään varmistamaan tiedon läpimeno oikeassa järjestyksessä. Tämä on tarpeen, sillä UDP ei varmista tiedon läpimenoa. Sovelluksen lähettämä tieto on voitu jakaa myös useampaan DTLS-tietueeseen ja järjestysnumerolla pystytään kokoamaan alkuperäinen UDP-segmentti. DTLS-menetelmässä on määritelty myös erillinen kättely-protokolla, millä tiedon läpimeno ja uudelleenlähetys tunnistetaan. (Perez 2014, 109-132.)



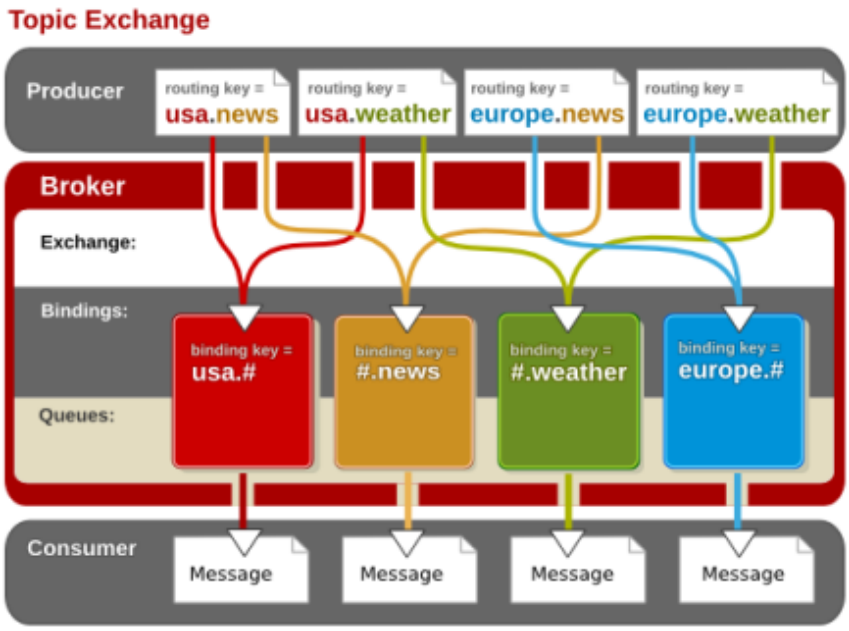
Kuvio 12. DTLS-otsakkeet (Perez 2014 127, muokattu)



Kuvio 13. DTLS-salausprosessi (Paterson & AlFardan 2013, 4, muokattu)

3.3 AMQP

AMQP:n (Advanced Message Queuing Protocol) versio 1.0 on OASIS järjestön hyväksymä ja ylläpitämä standardi. Protokollassa on tuottajia (producer), käyttäjiä (consumer) ja välittäjä (broker). Välittäjä jakaa viestit aihepiirien mukaan jonoihin, joista ne lähetetään käyttäjille (ks. kuvio 14). Käyttäjä tilaa tietyn aiheen yksityiskohtaisella aiheella tai korvaamalla osan aiheen määreestä *-merkillä. Usean määreen voi korvata #-merkillä. Protokolla mahdollistaa luotettavan ja tietoturvallisen viestien välityksen. Liikenteen salaukseen käytetään TLS- tai/ja SASL-menetelmää. Protokollan rakenne on määriteltynä XML DCD -tietorakennemäärityksen avulla. Hyötykuormalle löytyy useita tyyppimäärityksiä, jotka voivat olla myös vapaamuotoisia binaari- tai merkkijonoja. (ISO/IEC:19464 2014.)



Kuvio 14. AMQP viestien jonotuslogiikka (Chapter 2. Exchanges n.d.)

Avoimen lähdekoodin AMQP-välittäjiä ovat RabbitMQ, Apache Qpid ja joram. AMQP viesteille pilvipalveluna toteutettu AMQP-viestien välittäjiä ovat Microsoft Azure Service Bus ja CloudAMQP. IBM MQ Light on erillinen välittäjäpalvelu, mutta se on myös osana IBM:n Bluemix-pilvipalvelua. Microsoftin Azure-palvelusta löytyy AMQP-tuki ja maininta, että protokolla on erittäin luotettava, ja se on käytössä esimerkiksi isoilla

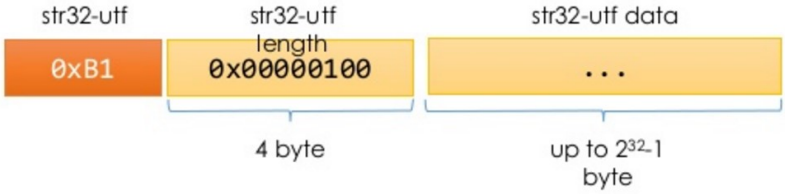
pankeilla (AMQP 1.0 support in Service Bus 2017). AMQP-protokollalle löytyy kirjastoja useille eri ohjelmointikielille, kuten Java, .NET, Ruby, Python, PHP, Objective-C, Node.js, C, C++, Go, Perl, GOBOL ja Scala (Clients & Developer Tools n.d). AMQP-protokollan eri komponenttien toiminnan ymmärtämiseen apuna voi käyttää RabbitMQ AMQP-simulaattoria (RabbitMQ Simulator n.d.).

Yhteyttä muodostettaessa määritellään suurin pakettikoko ja yhteyden tunniste. Yhteyden avaamisen yhteydessä suoritetaan tarvittaessa käyttäjän tunnistus SACL-menetelmällä ja/tai yhteyden salaus TLS-menetelmällä. Session avauksessa muodostetaan lähetys- ja vastaanottokanavat, jossa varsinaiset viestit liikkuvat. Salaamaton liikenne käyttää porttia 5672, TLS-porttia 5671 ja TLS websocket-portiksi on määritelty 433. (Clements 2015.)

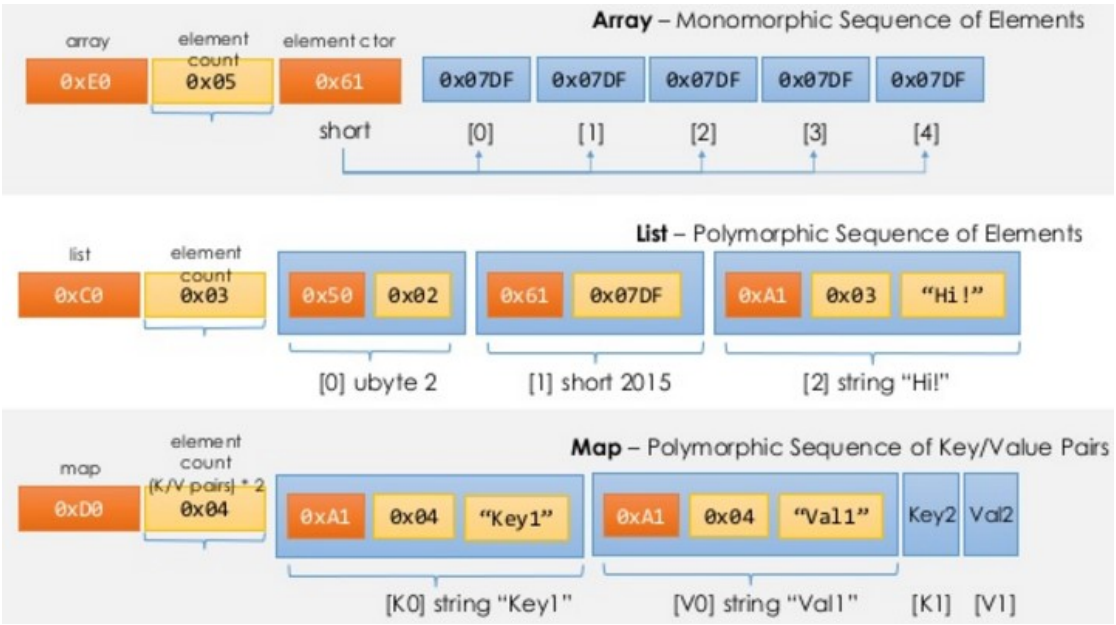
Viestin lähetyksessä voi valita kahdesta eri laatutasosta sopivamman. Ensimmäisellä laatutasolla viestin läpimenoa ei varmisteta (at most once/fire and forget). Toinen laatutaso varmistaa viestien läpimenon (at least once). Välittäjän jonoarkkitehtuurin vuoksi viestit pysyvät aikajärjestyksessä. Vuonohjauksella varmistetaan vastaanottajan kyvykkyys käsitellä viestejä. IoT-laitteissa vuonohjaus on tärkeä ominaisuus laitteiden rajoitetun kapasiteetin vuoksi. (Clements 2015.)

AMQP-protokollassa on tunnetuille tietotyypeille määritelty otsaketiedot, kuten esimerkiksi kuviossa 15. merkkijonon sovellustason otsakekenttä. AMQP-protokollan määrittelystä löytyy määritykset useille tietotyypeille, kuten kokonaisluvuille ja desimaaliluvuille. Tarvittaessa binaari-, merkkijono ja symbolitietoa voi siirtää 32-bittisen pituuskentän ilmoittaman määrän, eli teoreettisesti 4GB (ks. kuvio 16). AMQP mahdollistaa myös avain/arvo-, lista- ja taulukkomuotoisen viestirakenteen, joiden avulla voi lähettää erittäin monimuotoista tietoa (ks. kuvio 17). Siirrettävään hyötykuormaan lisätään aina hyötykuorman tyyppi, sen pituus, 8-tavuinen kehysotsake ja vähintään 11-tavuinen siirto-otsake hyötykuorman lisäksi. Esimerkiksi 4-tavuinen mittaustulos tarvitsisi vähintään 21 otsaketavua (kokonaisluku 0x54, pituus 0x01, 11 tavua siirto-otsaketta ja 8 tavua kehysotsaketta). Kuvio 18 esittää kuuden merkin siirtämiseen tarvittavat otsakkeet.

Kuvio 15. AMQP-merkkijonon otsake (Clements 2015, 26)



Kuvio 16. AMQP:n isot pakettikoot (Clements 2015, 28)

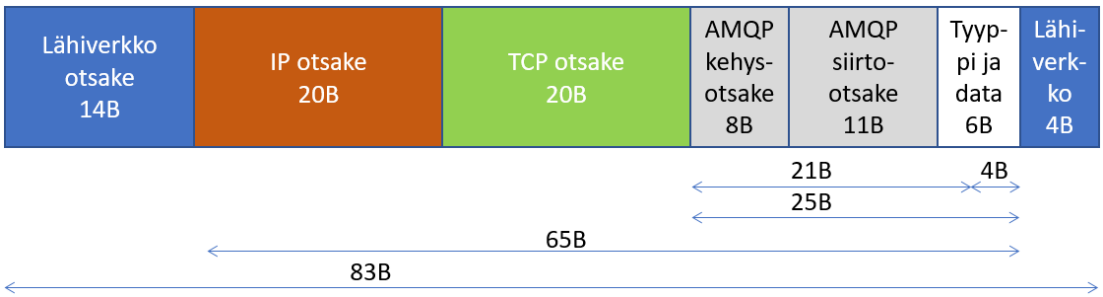


Kuvio 17. AMQP:n monimuotoiset viestityypit (Clements 2015, 29)

```
Frame Header                        ..... 8 bytes
Transfer Performative       5b
     link-handle     (1b) 1+4b
     delivery-id     (1b) 1+4b
     delivery-tag (2+1b) 2+4b
     message-format       1b
   (theoretical min 11 bytes)       .... 22 bytes
AMQP Message
     amqp-value           3b
         "Hello!"       2+6b        .... 11 bytes
                                    ==============
                                         41 bytes
```

Kuvio 18. AMQP-esimerkkikehyksen tavumäärä (Clements 2015, 37)

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden kokonaisluvun lähettämiseen AMQP-protokollalla 83 tavua eli 664 bittiä (ks. kuvio 19). IPv6-lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 109 tavua / 872 bittiä.



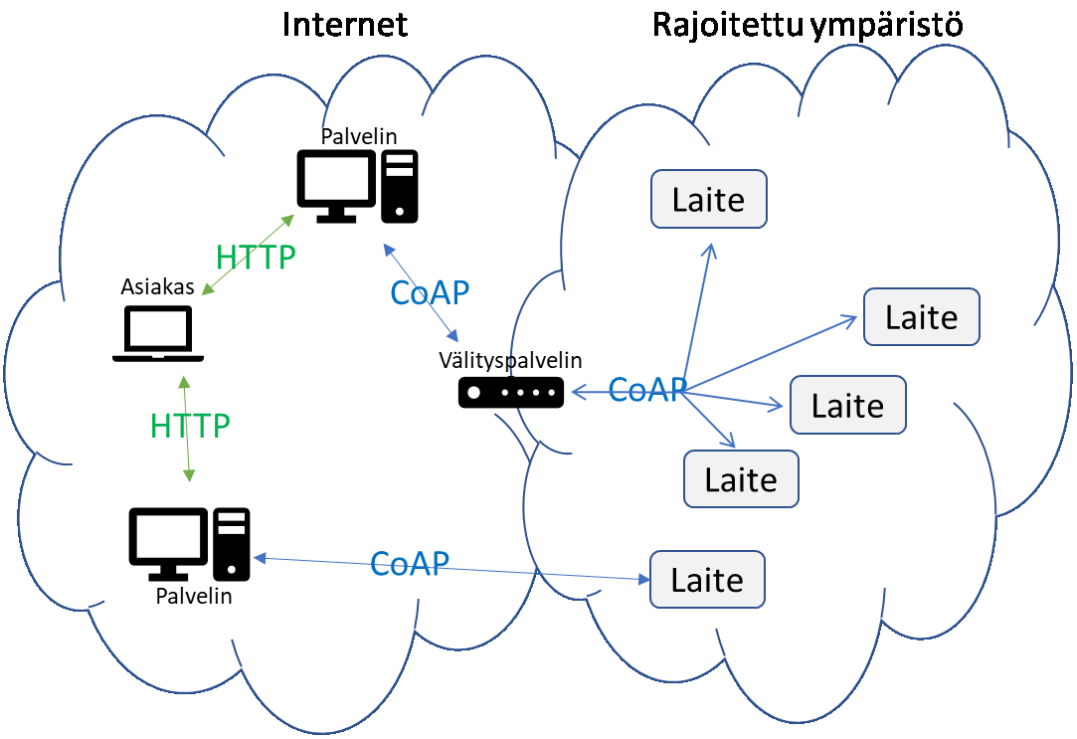
Kuvio 19. Kokonaisluvun lähetys lähiverkossa AMQP-protokollalla

3.4 CoAP

CoAP (Constrained Application Protocol) on IEFT:n standardoima protokolla vähävirtaisille ja pienen prosessointikapasiteetin omaaville laitteille. Tiedonsiirrossa CoAP käyttää UDP-segmenttejä ja turvalliseen liikenteeseen DTLS-segmenttejä. Toiminta-periaatteeltaan ja viestityypeiltään CoAP-protokolla pohjautuu HTTP-protokollaan. Viestin otsakkeet ovat kuitenkin kevyempiä ja binaari-formaatissa. Viestiliikenne protokollassa on asynkroninen pyyntö-vastaus (request-response) -tyyppinen. (RFC7252 2014, 1, 5.)

IoT-laitetta voidaan pyytää lähettämään viestejä jatkuvasti, eli on mahdollista esimerkiksi pyytää lämpötilasensoria lähettämään lämpötilatietoja 5 minuutin välein. Laitteiden osoitteena käytetään URI-osoitteita, jotka serveri prosessoi ja lähettää eteenpäin oikealle CoAP-laitteelle. Esimerkkiosoite, jossa tietyn lämpötilasensorin arvoa pyydetään raportoimaan 5 minuutin välein: ”coap://iotlaitteet.io/lampotila/alakerta?min=5”. Protokollassa on tuki myös palveluiden etsintään, eli palvelimelta voidaan hakea kaikki lämpötilasensorit.

Kuvio 20 kuvaa CoAP-verkon rakennetta ja samalla myös ideologiaa. Pienen prosessointikapasiteetin omaavat laitteet muodostavat oman CoAP-verkon, mikä kytkeytyy Internetiin erillisen välityspalvelimen (proxy) läpi. Eräänä mahdollisuutena onkin käyttää IoT-laitteille omaa 6LoWAN-verkkoa, jolla pystyy minimoimaan otsakkeet kymmenenteen osaan verrattuna TCP/IPv6-otsakkeeseen. (Shelby 2013, 5.)

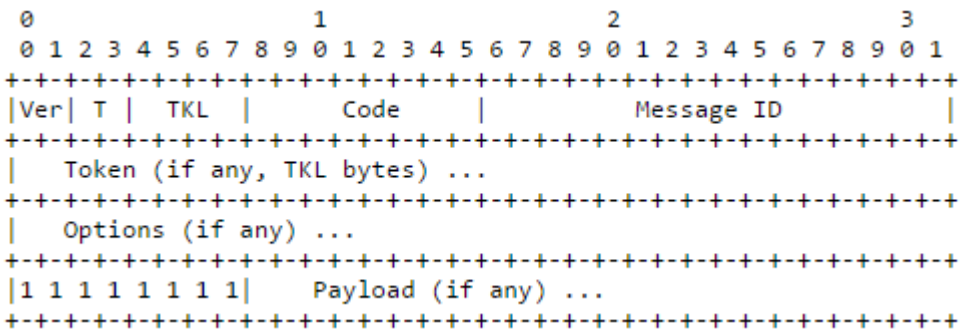


Kuvio 20. CoAP-verkkoarkkitehtuuri (Shelby 2013, 4, muokattu)

CoAP-protokollalle on kirjastot useampaan sulautettuun järjestelmään. Kirjastoja löytyy esimerkiksi: C-, C#-, Python-, Go-, Java-, JavaScript- ja Ruby-ohjelmointikielille. IoT-pilvipalveluista thethings.io, ARTIK tukee CoAP-protokollaa, mutta esimerkiksi AWS (AWS protocols 2017) ja Azure (Azure IoT protocol gateway. 2017) eivät. (Implementations n.d.)

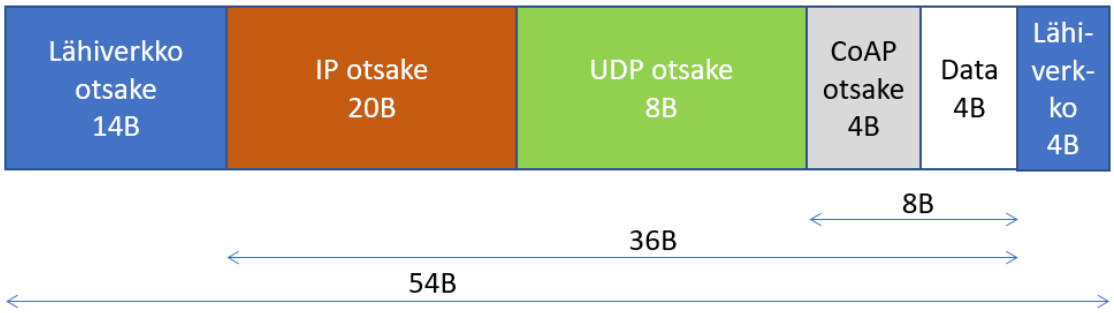
Internet-selaimiin on saatavilla Copper niminen laajennus, mikä mahdollistaa viestien muodostamisen ja lähettämisen selaimella (Kovatsch 2017). Protokollan yhtäläisyys HTTP-protokollaan on mahdollistanut tehokkaan työkalun rakentamisen selaimella.

Kuvio 21 esittää CoAP pyyntöviestin rakenteen. Viestin pakollinen osuus koostuu versiosta (Ver), viestin tyypistä (T), valtuutus kentän pituudesta (TKL), viestin koodista (Code), viestin numerosta (Message ID), sekä 0xFF-tavusta. CoAP-protokollan otsake on siis vähintään 5 tavua. IoT-laitteen vastauksessa jätetään 0xFF-kenttä huomioimatta, jolloin otsakkeen koko on 4 tavua. (RFC7252 2014, 15-20.)



Kuvio 21. CoAP-otsakkeet (RFC7252 2014, 16)

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden kokonaisluvun lähettämiseen CoAP-protokollalla 54 tavua eli 432 bittiä (Kuvio 22). IPv6-lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 80 tavua / 640 bittiä. Huomioitavaa viestin kokonaiskoossa on UDP-protokollan käyttö verrattuna useimpiin muihin IoT-protokolliin.

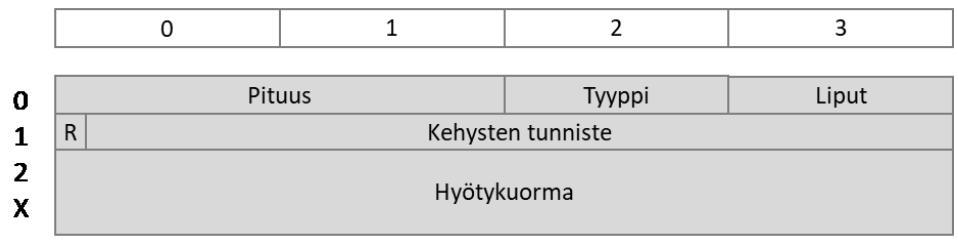


Kuvio 22. Kokonaisluvun lähetys lähiverkossa CoAP-protokolalla

3.5 HTTP/HTTPS

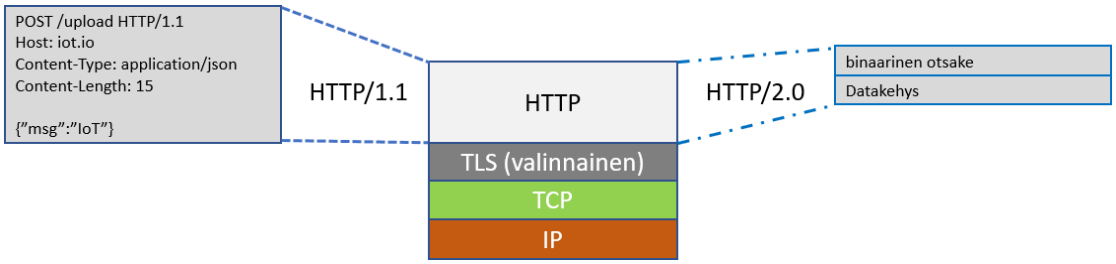
HTTP versio 1.1 ja HTTP versio 2 -protokollien käyttäminen tiedon välittämiseen IoT-laitteelta palvelimelle on todennäköisesti yksinkertaisin tapa. IoT-laitteet yleensä tukevat HTTP-viestien lähetystä. Tiedon välittämisessä käytetään URI-osoitetta ja monesti myös HTTP-protokollan otsaketta. URI-osoitteella muodostettu ohjelmointirajapinta (API) tunnetaan myös REST-arkkitehtuurina. Suurimmassa osassa IoT-pilvipalveluja, ellei kaikissa on REST-API mahdollisuus. Miten osoite muodostuu, millaisia parametrejä täytyy antaa ja millainen tietoturva palvelussa on, ovat täysin palveluntarjoajan määrittelemiä. IoT-laitteen ohjauksessa HTTP-REST -laitteen täytyy käytännössä käydä kysymässä palvelimelta ohjausarvoja halutuin määräajoin. Palvelin ei siis voi suoraan ilmoittaa IoT-laitteelle ohjaustarpeesta, mikä kasvattaa energiankulutusta ja verkkoliikennettä. (RFC2616 1999; RFC7540 2015.)

Kuvio 23 mukaisesti http-otsake muodostuu viestin pituudesta, tyypistä, lipuista, varatusta bitistä (R) ja yhteyteen liittyvien kehysten tunnisteesta (Stream Identifier). (Grigorik 2014; RFC2616 1999.)



Kuvio 23. HTTP 1.1 otsake (Grigorik 2014, 32, muokattu)

HTTP-protokollan versiot 1.1 ja 2.0 eroavat toisistaan esitystapakerroksen osalta. HTTP 1.1 käsittelee otsakkeen tiedot ASCII-formaatissa, kun taas HTTP 2.0 käsittelee ne binaari-formaatissa (ks. kuvio 24). Binaari-formaatti menee huomattavasti pienempään tilaa ja näin se tarvitsee huomattavasti vähemmän siirtokaistaa. (Grigorik 2014.)



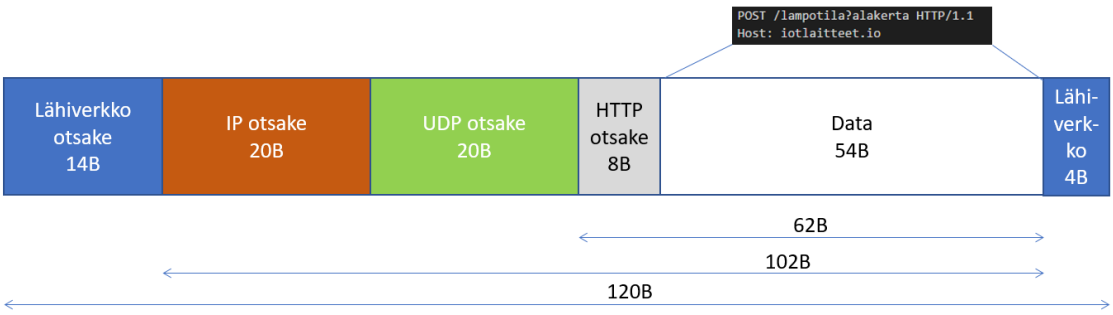
Kuvio 24. HTTP1.1:n ja HTTP2.0:n erot tietorakenteessa (Grigorik 2014, 30, muokattu)

HTTP POST -menetelmällä parametrit välitetään osana URL-osoitetta, esimerkiksi ”http://iotlaitteet.io/lampotila?alakerta”. GET-menetelmällä parametrit välitetään HTTP-otsakkeessa, esimerkiksi: ”http://iotlaitteet.io” ja otsakkeessa on esimerkiksi avain arvoparina ”lampotila: alakerta”. Kuvion 23 mukaisesti kummankin menetelmän tieto siirtyy kehyksen hyötykuormassa (Frame Payload) samaan tyyliin, kuten kuviossa 25.

```
POST /lampotila?alakerta HTTP/1.1
Host: iotlaitteet.io
```

Kuvio 25. HTTP 1.1 hyötykuorman esimerkki

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden mittaustuloksen lähettämiseen HTTP-protokollalla 120 tavua eli 960 bittiä (ks. kuvio 26). IPv6-lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 146 tavua / 1168 bittiä.



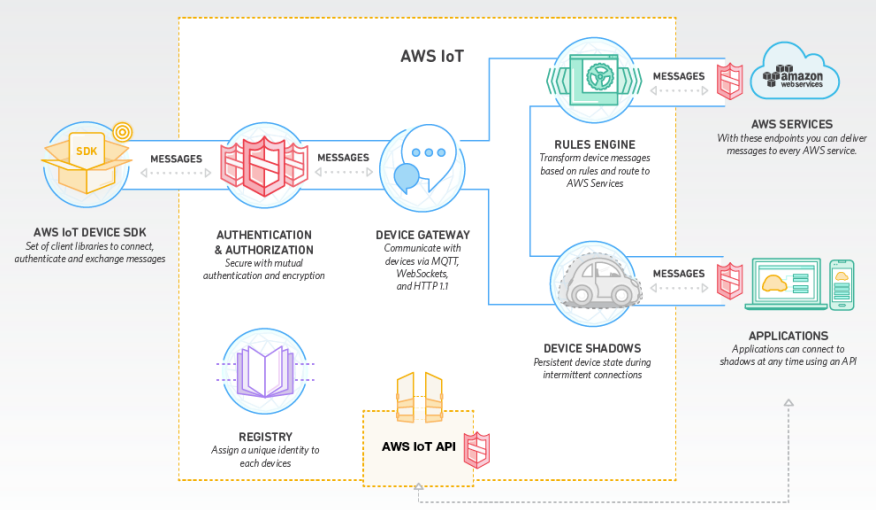
Kuvio 26. Mittaustiedon lähetys lähiverkossa HTTP1.1-protokollalla

3.6 MQTT

MQTT on lähtökohtaisesti määritelty yksinkertaiseksi ja kevyeksi tuottaja/tilaaja (publish/subscribe) protokollaksi. Protokolla on suunniteltu rajoitetun kapasiteetin laitteille ja toimimaan myös hitaalla ja epävakaalla internet-yhteydellä. (MQTT Essentials: Part 1 - - n.d.)

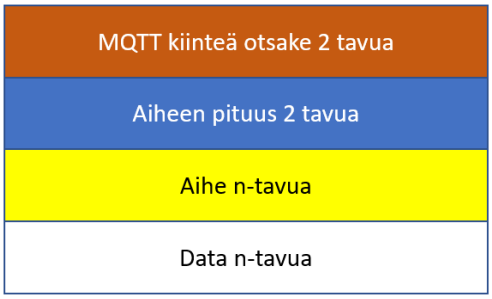
Protokolla kehitettiin vuonna 1999 öljyputkien seurantaan satelliittiyhteydellä. IBM jatkokehitti protokollaa ja julkaisi sen avoimena protokollana vuonna 2010, jolloin protokollasta oli määriteltynä versio 3.1. OASIS-järjestön viralliseksi standardiksi MQTT-protokolla otettiin vuonna 2014 ja samalla protokollasta julkaistiin versio 3.1 virallisesti. Virallista tietoa protokollan seuraavasta versiosta ja sen ominaisuuksista ei ole julkaistu. (MQTT Essentials: Part 1 - - n.d.)

MQTT-protokollalle löytyy tuki suurimmasta osasta IoT-pilvipalveluita. Kuvio 29 esittää tyypillisen IoT-pilvipalvelun arkkitehtuurin, missä MQTT-protokollaa käytetään kommunikointiin IoT-laitteiden kanssa. IBM on kehittänyt erittäin käyttäjäystävällisen käyttöliittymän erityisesti MQTT-protokollan viestien prosessointiin. Käyttöliittymä on selainpohjainen ja nimeltään Node-RED. Node-RED on merkittävänä osana IBM Bluemix, Samsung Artik, sekä FRED IoT-pilvipalveluissa. (Node-RED Flow-based - - n.d.)



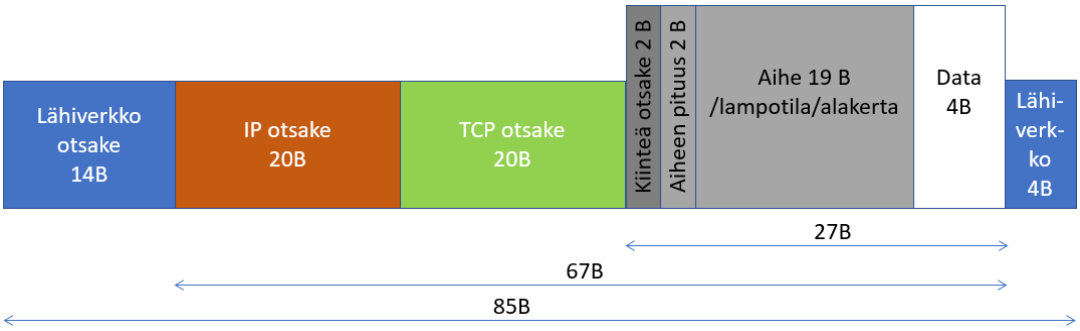
Kuvio 27. Amazon AWS IoT-pilvipalvelun arkkitehtuuri (How the AWS - - n.d.)

MQTT-viestit koostuvat kiinteämittaisesta otsakkeesta ja tarvittaessa muuttuvamittaisesta jatko-otsakkeesta (ks. kuvio 28). Kiinteän otsakkeen pituus voi myös vaihdella. Kiinteän otsakkeen pituus on 2 tavusta 5 tavuun, sillä kiinteän otsakkeen pituuskenttä voi olla 1-4 tavua pitkä. Pituuskentän ylin bitti ilmaisee sen, että jatkuuko pituuskenttä seuraavassakin tavussa. Viestin maksimipituus 2 tavun otsakkeella on 127 tavua ja 5 tavun otsakkeella 256 megatavua. (MQTT Version 3.1.1 2014, 16-22.)



Kuvio 28. MQTT-protokollan julkaisuviestin rakenne

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden mittaustuloksen lähettämiseen MQTT-protokollalla 85 tavua eli 680 bittiä (ks. kuvio 29). IPv6 lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 111 tavua / 888 bittiä.



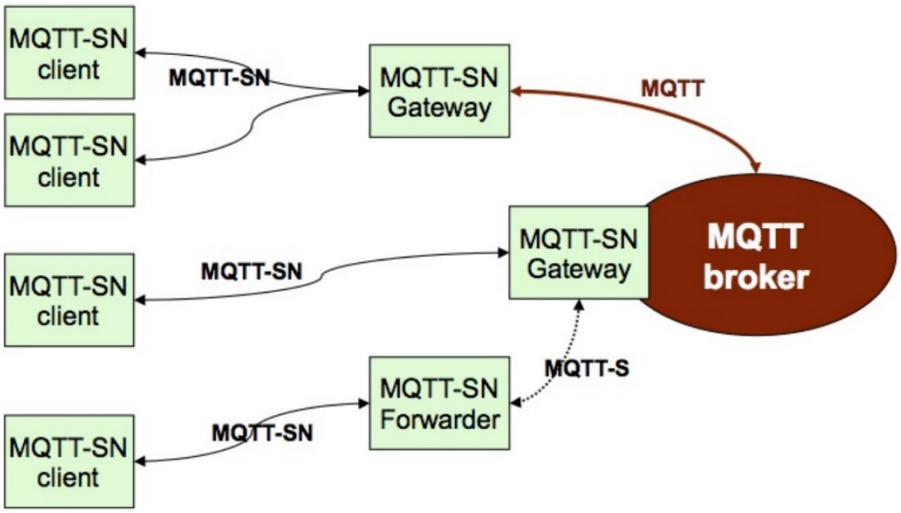
Kuvio 29. Mittaustiedon lähetys lähiverkossa MQTT-protokollalla

3.7 MQTT-SN/MQTT-S

MQTT-protokollassa on muutamia ongelmia, jotka heikentävät sen käyttöä pienen energialähteen laitteissa. Ongelmia ovat TCP/IP-protokollaan sitoutuminen, jatkuva olemassaolon viestitys ja viestin aiheen lähetys tekstiformaatissa.

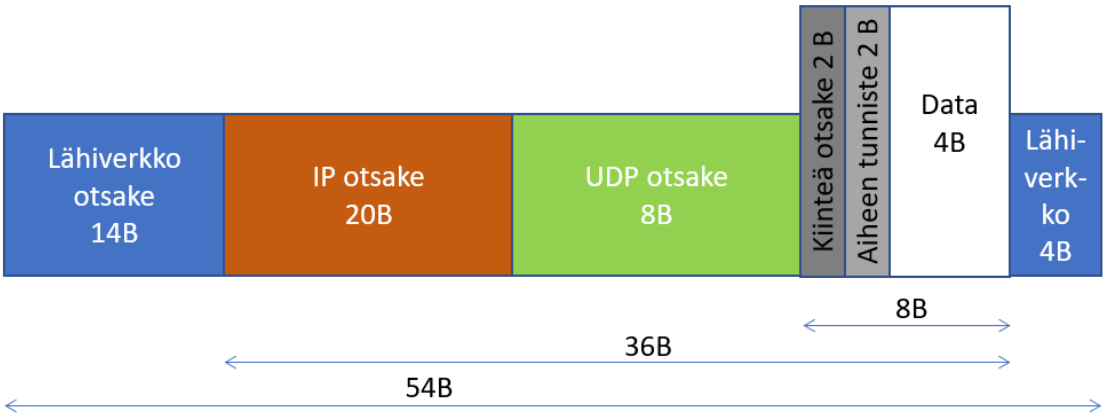
MQTT-SN korjaa edellä mainittuja ongelmia. MQTT-S merkintää käytetään myös kuvaamaan MQTT-SN-protokollaa. Protokollassa viestit lähetetään käyttäen aiheen tunnistetta (2B). Laite voi ilmoittaa nukkuvansa pitkän ajan, jolloin välittäjä tallettaa kaikki laitteen tilaamat viestit ja välittää ne laitteelle sen kommunikoidessa välittäjälle. TCP/IP-protokollan sijaan siirtotienä on alun perin ollut ZigBee- ja UDP/IP-protokolla. (MQTT For Sensor - - 2013, 3-5.)

MQTT-SN arkkitehtuurissa määritellään erillinen MQTT-SN -yhdyskäytävä (Gateway) ja siirtäjä (Forwarder), joihin MQTT-SN -laitteet ovat yhteydessä (ks. kuvio 30). Yhdyskäytävä ja siirtäjä toimivat mediamuuntimena mahdollistaen MQTT-SN -laitteiden yhdistymisen MQTT-protokollan välittäjään. Mosquitto-välittäjä osaa toimia sekä MQTT-SN yhdyskäytävänä, että MQTT-välittäjänä. (MQTT For Sensor - - 2013, 3-5.)



Kuvio 30. MQTT-SN arkkitehtuuri (MQTT For Sensor - - 2013, 5)

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden kokonaisluvun lähettämiseen MQTT-SN-protokollalla 54 tavua eli 432 bittiä (ks. kuvio 31). IPv6-lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 80 tavua / 640 bittiä. Huomioitavaa viestin kokonaiskoossa on UDP-protokollan käyttömahdollisuus ja viestin pituus, mikä vastaa CoAP-protokollaa.



Kuvio 31. Kokonaisluvun lähetys lähiverkossa MQTT-SN-protokolalla

3.8 STOMP

STOMP on HTTP-tyyppinen tekstipohjainen protokolla, missä asiakkaat (clients) muodostavat yhteyden välittäjään (broker). Tiedon kuluttajat tilaavat (subscribe) halutun lähteen. Tekstipohjaisuuden etuna on se, että välittäjään voi muodostaa yhteyden ja lähettää viestejä esimerkiksi Telnet-ohjelmistolla. (STOMP Protocol - - 2012.)

Viestityyppejä Stomp-protokollassa on vain noin kymmenen kappaletta (ks. kuvio 32). Yhteyden muodostamisen (ks. kuvio 33) ja katkaisemisen lisäksi IoT-laitteen täytyisi toteuttaa ainakin lähetysviesti (ks. kuvio 34). Kaksisuuntaisessa toteutuksessa IoT-laitteen täytyisi toteuttaa myös tilaus-viesti. (STOMP Protocol - - 2012.)

```
NULL                = <US-ASCII null (octet 0)>
LF                  = <US-ASCII line feed (aka newline) (octet 10)>
CR                  = <US-ASCII carriage return (octet 13)>
EOL                 = [CR] LF
OCTET               = <any 8-bit sequence of data>

frame-stream        = 1*frame

frame               = command EOL
                      *( header EOL )
                      EOL
                      *OCTET
                      NULL
                      *( EOL )

command             = client-command | server-command

client-command      = "SEND"
                      | "SUBSCRIBE"
                      | "UNSUBSCRIBE"
                      | "BEGIN"
                      | "COMMIT"
                      | "ABORT"
                      | "ACK"
                      | "NACK"
                      | "DISCONNECT"
                      | "CONNECT"
                      | "STOMP"

server-command      = "CONNECTED"
                      | "MESSAGE"
                      | "RECEIPT"
                      | "ERROR"

header              = header-name ":" header-value
header-name         = 1*<any OCTET except CR or LF or ":">
header-value        = *<any OCTET except CR or LF or ":">
```

Kuvio 32. Stomp-protokollan BNF (Backus-Naur Form) -rakenne (STOMP Protocol - - 2012)

```
CONNECT
accept-version:1.0,1.1,2.0
host:iotlaitteet.io
^@
```

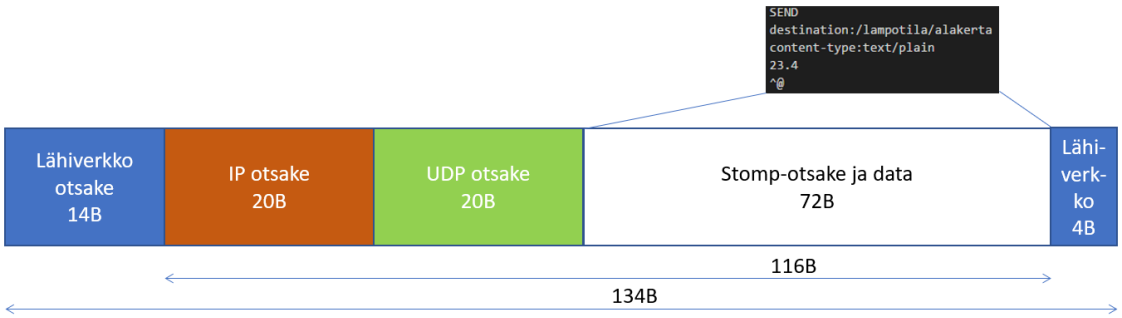
Kuvio 33. Stomp-yhteyden muodostus

```
SEND
destination:/lampotila/alakerta
content-type:text/plain
23.4
^@
```

Kuvio 34. Stomp-viestin lähetys

Stomp-protokolla toimii yleisesti portissa 61613 ja SSL salattuna 61614-portissa. Protokollassa on mahdollista tunnistautua käyttäjätunnuksella ja salasanalla. Välittäjiä protokollalle ovat ainakin Apachen ActiveMQ ja RabbitMQ. Pilvipalvelut eivät suoraan mainosta Stomp-protokollan tukea, mutta erillisillä lisäosilla tuki protokollalle löytyy joistakin palveluista.

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden mittaustuloksen lähettämiseen Stomp-protokollalla 134 tavua eli 1072 bittiä (ks. kuvio 35). IPv6 lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 160 tavua / 1280 bittiä.



Kuvio 35. Mittaustiedon lähetys lähiverkossa Stomp-protokollalla

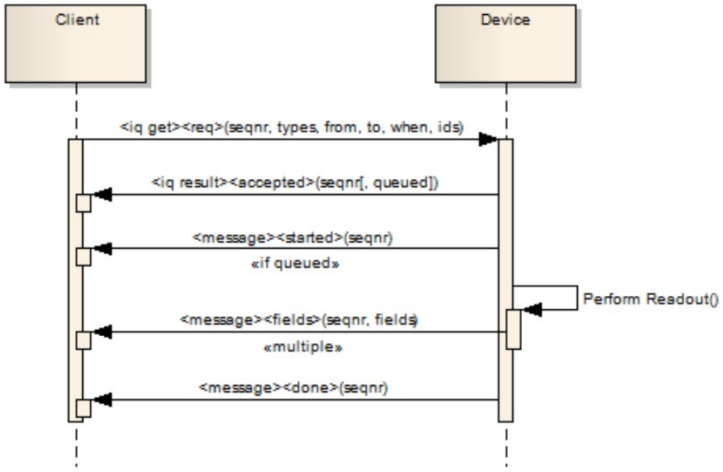
3.9 XMPP

XMPP suunniteltiin erilaisten keskustelu-ohjelmien tarpeeseen. XMPP mahdollistaa kaksisuuntaisen asynkronisen viestinnän asiakkaan ja palvelimen välillä. XMPP-protokollassa viestin rakenne kuvataan XML-kielellä. Protokollaa on käytetty ja käytetään verkkosivujen reaaliaikaiseen päivittämiseen, kuten keskusteluihin, työtiloihin, peleihin ja videopuheluihin. Usean käyttäjän keskustelu, palveluiden haku (service discovery), tuottaja/tilaaja (publish/subscribe) ja etäproseduurikutsu (remote procedure call, RPC) ovat myös osana protokollan määritystä. IEFT standardoi XMPP-protokollan vuonna 2004. (Moffitt 2010, 21, 32.)

XMPP-protokolla tukee useita eri viestimuotoja. Viestimuodot määritellään XMPP-laajennuksina, joita on jo lähes 400 erilaista. XEP-0114 laajennos määrittelee, miten laajennoksia tehdään palvelin riippumattomasti. IoT-laitteiden kommunikoinnin formaatin määritelevä laajennos on toistaiseksi työn alla, eikä siitä ole vielä määritystä saatavilla. IoT-laitteille löytyy laajennuksia anturitiedon siirtoon, käyttöönottoon, ohjaukseen ja hallintaan. Tietoturva XMPP-protokollaan on toteutettu TLS- ja SASL-protokollien mukaisesti (Moffitt 2010, 30, 46; XMPP Specifications n.d.).

XMPP-osoite, eli JID (jabber identifier) muistuttaa sähköpostiosoitetta (esimerkiksi ”koti@iotlaitteet.lampotila”). Toimialue (domain) -osa on aina pakollinen (esimerkissä ”iotlaitteet”). Toimialue vastaa DNS (Domain Name Server) palvelimelta löytyvää nimeä, jolle löytyy IP-osoite. IP-osoitteen mukaan viesti löytää perille oikeaan kohteeseen. Osoitteessa kaksi muuta osaa ovat paikallinen- (local) ja resurssiosa (resource). Paikallinen osa viittaa yleensä käyttäjään ja resurssipalveluun. Resurssia voi tarkentaa lisäämällä kenoviivan ja lisämääritteen osoitteeseen (esimerkiksi ”koti@iotlaitteet.lampotila/alakerta”). (Moffitt 2010, 34.)

XEP-0323 mukaan IoT-laitteelta pyydetään (iq-tyyppinen viesti) haluttua tietoa (ks. kuvio 37). Laite lähettää pyynnölle hyväksynnän tai hylkäyksen (ks. kuvio 38). Varsinainen mittaustieto lähetetään erillisellä viestillä (message-tyyppi) (ks. kuvio 39). Sekvenssi on kuvattuna alla olevassa kuviossa. (XEP-0323 2015.)



Kuvio 36. XMPP- ja anturitiedon lukeminen (XEP-0323 2015)

```
<iq type='get'
    from='client@example.org/mobile'
    to='koti@iotlaitteet.lampotila'
    id='S0001'>
    <req xmlns='urn:xmpp:iot:sensordata' seqnr='1' momentary='true'/>
</iq >
```

Kuvio 37. XMPP IoT-tiedon pyyntö

```
<iq type='result'
    from='koti@iotlaitteet.lampotila'
    to='client@example.org/mobile'
    id='S0001'>
    <accepted xmlns='urn:xmpp:iot:sensordata' seqnr='1'/ >
</iq >
```

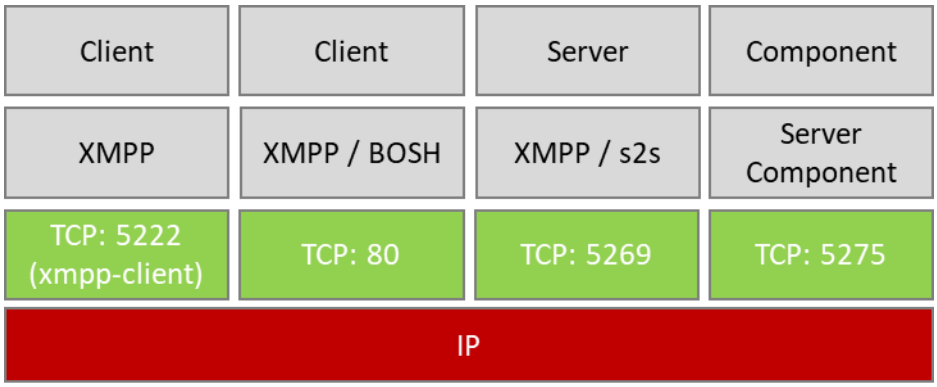
Kuvio 38. XMPP IoT-hakupyynnön hyväksyntä

```
<message
    from='koti@iotlaitteet.lampotila'
    to='client@example.org/mobile'>
    <fields xmlns='urn:xmpp:iot:sensordata' seqnr='1' done='true'>
        <node nodeId ='Device01'>
            <timestamp value ='2017-04-01 T14:54:00'>
                <numeric name='Temperature' momentary='true '
                    automaticReadout='true' value='23.4' unit='°C'/>
            </timestamp>
        </node>
    </fields>
</message>
```

Kuvio 39. XMPP IoT-mittaustulos

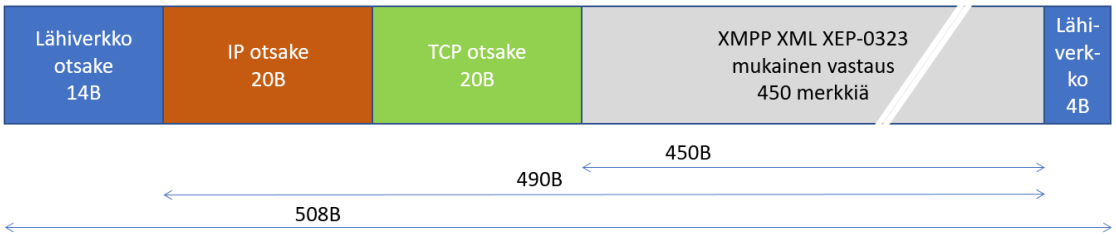
XMPP-protokollassa on myös läsnäolo (presence) -määritys, jolla on viestiohjelmissa pääsääntöisesti esitetty, onko henkilö tavoitettavissa vai ei (XEP-0163 2010). IoT-laitteissa ominaisuutta voitaisiin käyttää laitteen tilan kertomiseen. Laitteen tilasta voisi kertoa, että onko laite yhdistettynä verkkoon ja onko se toimintakunnossa.

Protokollan asiakkaalle on määritelty portiksi 5222 ja serverille 5269 (ks. kuvio 40). Tämän lisäksi yhteyksien hallintaan on olemassa BOSH (Bidirectional-streams Over Synchronous HTTP) -protokolla, mikä toimii HTTP/HTTPS- tai omassa 5280-portissa. HTTP/HTTPS-porttien hyvänä puolena on se, että yleisesti palomuurit sallivat liikenteen näiden porttien läpi. (Moffitt 2010, 56-58, 403-406.)



Kuvio 40. XMPP-protokollapino (Waher 2015, 137, muokattu)

Kuvion 3 (s. 16) mukaista lähiverkon kehysrakennetta noudattaen tarvitaan yhden mittaustuloksen lähettämiseen XMPP-protokollalla 508 tavua eli 4064 bittiä (ks. kuvio 41). IPv6-lähiverkossa lisäystä tulee vähintään 26 tavua, eli yhteensä 534 tavua / 4272 bittiä.



Kuvio 41. Mittaustiedon lähetys lähiverkossa XMPP-protokollalla

Protokollassa on mahdollisuus hakea tuettuja ominaisuuksia haku-tyypin (query) komennolla. IoT-laitteissa tämä voisi mahdollistaa esimerkiksi laitteiden ja niiden ominaisuuksien kyselyn. Tilaaja-tuottaja (publish-subscribe) -ominaisuus löytyy myös laajennoksena (XEP-0060, 2016). Edellä esitetyssä XMPP IoT-laajennuksessa tilaaja-tuottaja ominaisuutta ei kuitenkaan hyödynnetty. (Moffitt 2010, 171-174, 229-244.)

XMPP-protokollalle löytyy useita asiakassovelluksia, sekä kirjastoja eri ohjelmointikielille. Tuettuja ohjelmointikieliä ovat esimerkiksi: C, Objective-C, C++, C#, Java, JavaScript, PHP, Python, Qt ja Ruby. Arduino-sovelluskehitys ympäristöön löytyy myös XMPP-protokolla. IoT-pilvipalveluissa ei suoranaisesti mainostettu XMPP-protokollan tukea, muuhun kuin keskusteluviestien välittämiseen. (XMPP software n.d.)

3.10 Yhteenveto ja vertailu

Tietoturvan näkökulmasta kaikki protokollat luottavat alemman kerroksen tai kerroksien tarjoamaan tietoturvaan (esim. TLS ja IPSec). Lähes kaikista protokollista löytyy käyttäjän tai laitteen tunnistus (username ja password). Varsinaista laitehallintaa ei kuitenkaan yhdessäkään protokollassa ole määritelty.

Esimerkki-mittaustuloksen lähettäminen vaatii selvästi eniten lähetyskapasiteettia XMPP-protokollalta (508B). HTTP ja Stomp vaativat merkittävästi vähemmän (120B – 134B). AMQP ja MQTT ovat seuraavassa sarjassa (83B – 85B). Vähiten kapasiteettia vaatii CoAP ja MQTT-SN (54B). Tämä on toisaalta vain yksi näkökulma protokollien soveltuvuuteen IoT-laitteissa. Tärkeitä huomioitavia näkökulmia on myös ylläpitoon vaaditut viestit sekä erilaiset kuittausviestit. CoAP ja MQTT-SN -protokollat vaativat enemmän viestien kuittausviestejä, sillä ne toimivat käyttäen epäluotettavia yhteyksiä (esim. UDP/IP). Toisaalta esimerkiksi TCP aiheuttaa ylimääräistä viestitystä varmistaakseen viestien läpimenon ja yhteyden olemassaolon. Kuittaus ja ylläpitoviestien vaikutus protokollaan on rajattu ulos tästä opinnäytetyöstä.

Eräs vastaan tullut jaottelunäkökulma oli IoT-protokollien käyttökohteet. MQTT ja XMPP oli merkitty laitteen ja palvelimen väliseen kommunikointiin (D2S device to server). AMQP-protokolla oli käytössä palvelimien välisessä liikenteessä (S2S server to server). (Schneider 2013.)

Eniten IoT-protokollan valintaan vaikuttaa se, millaisista komponenteista järjestelmä koostuu. Toisaalta protokolla voi olla järjestelmän arkkitehtuurin perusta. Käyttökohteeseen sopivan protokollan valintaan liikenteen kuormittavuutta enemmän vaikuttaa se, tarvitseeko liikenne olla kaksisuuntaista. Joissain tapauksissa on myös tärkeää tietää, onko laite saavutettavissa ja toiminnassa. Valmiit ohjelmistot ja kohteeseen sopivat lisenssit määrittävät myös protokollan sopivuuden käyttökohteeseen. Erilaiset työkalut helpottavat tuotekehitystä, joten työkaluille kannattaa antaa myös painoarvoa protokollaa valittaessa. Ei siis ole suoraa vastausta siihen, mikä olisi paras IoT-protokolla. MQTT-protokolla on kuitenkin saavuttanut vankan aseman IoT-protokollana, vaikka siinä heikkouksia onkin. Käytännössä kaikki pilvipalvelut, palvelimet ja välittäjät pystyvät käsittelemään MQTT-protokollaa. Todennäköisesti MQTT-protokollan yleisyyteen on syynä myös toteutuksesta saatavilla olevat avoimet lähdekoodit niin laitteelle, kuin välittäjällekin. MQTT-protokollan aiheen sisällyttäminen jokaiseen julkaistavaan viestiin kasvattaa viestin kokoa merkittävästi, tämä on korjattu MQTT-SN-protokollaan. MQTT-SN- ja CoAP-protokollat olisivat keveytensä ansiosta paremmat vaihtoehdot IoT-protokollalle, mutta aika näyttää voittavan protokollan. Lyhyt yhteenveto kappaleessa 3 esitettyjen protokollien ominaisuuksista löytyy liitteestä 1.

4 MQTT-protokolla

4.1 Yleistä

MQTT-protokollan yleiset ominaisuudet esitettiin kappaleessa 3.6 (s. 31). IBM:n Redbook kokoelmasta löytyy muutamia hyviä teoksia, joissa käydään läpi MQTT-protokollan yleisiä ominaisuuksia ja erilaisia asiakasohjelmistoja. Kirjoista ”Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry” syventyy protokollan toimintaan, viestirakenteisiin ja asiakasohjelmiston toteutukseen C ja Java -ohjelmointikielillä (Lampkin, Leong, Olivera, Rawat, Subrahmanyam 2012). ”Building Real-time Mobile Solutions with MQTT and IBM MessageSight” käsittelee protokollan yleisen toiminnan lisäksi muutamaa käyttötapausta, sekä iOS, Android ja HTML5 toteutusta (Boyd, Gauci, Robertson, Duy, Gupta, Gucer, Kislicins 2014).

MQTT-protokollan yleistymiseen syy lienee maksuttomuus, sekä avoimen lähdekoodin toteutukset. Windows, Linux ja iOS alustoille löytyy lukuisia eri välittäjiä, joista Mosquitto lienee tunnetuin (Gupta 2016). Arduinolle ja esimerkiksi ESP8266-alustalle löytyy useita asiakasohjelmistoja (Lewis n.d.). Yleisimmille mobiilikäyttöjärjestelmille löytyy myös omat asiakasohjelmistot (Boyd ym. 2014, 189-216). Testaamiseen on olemassa myös työkaluja, joista mainittakoon WireShark ja MQTT.fx. WireShark pystyy seuraamaan verkkoliikennettä ja avaamaan MQTT-protokollan viestikehykset reaaliaikaisesti. MQTT.fx on graafinen viestien julkaisu- ja tilaustyökalu.

Välittäjien nopeudessa IBM:n MessageSight pystyy välittämään 13 miljoonaa viestiä sekunnissa ja palvelemaan samaan aikaan miljoonaa yhtäaikaista yhteyttä. Valitettavasti mittausjärjestelyä ei ole kerrottu ja toisaalta tulokset ovat vähintään 3 vuotta vanhoja (Boyd ym. 2014, 49). IBM:n avoimen lähdekoodin Mosquitto välittäjän vertailussa muutamaan muuhun kevyempään välittäjään osoittaa, että näillä välittäjillä pystytään palvelemaan tuhansista muutamiin kymmeniin tuhansiin asiakasta (Benchmark of MQTT servers 2015). Merkittävänä erona MessageSight ja muiden välittäjien välillä on se, että MessageSight on pelkästään MQTT-protokollaan optimoitu käyttöjärjestelmä laitteistokiihdytyksellä, kun taas muut ovat käyttöjärjestelmään asennettavia ohjelmistoja.

4.1.1 Julkaiseminen (publish)

Asiakkaan lähettämää tietoa kutsutaan MQTT-protokollassa julkaisemiseksi. Julkaiseminen tapahtuu aina johonkin aiheeseen. Aiheen muodostus ja käyttö on esitetty kappaleessa 4.1.3. Lähetyksen sisältö on sovelluskohtainen yksityiskohta, johon protokolla ei ota kantaa (ks. kappale 4.1.4). Viestin julkaisu on esitetty yksityiskohtaisesti kappaleessa 4.4 (s. 61). Julkaistessa tiedon, asiakas ei tiedä, kuinka monelle tilaajalle tieto päätyy.

Sovelluskohtaisesti voi päättää, luodaanko yhteys ja pidetäänkö se voimassa, mikä olisi yleisin tapaus (ks. liite 7). Joissain tilanteissa saattaa olla järkevää muodostaa yhteys tietojen julkaisemiseksi ja tämän jälkeen sulkea yhteys. Yhteyden katkaiseminen voi olla mielekästä, jos julkaistaan erittäin harvoin ja julkaiseva laite voi siirtyä tehokkaaseen virransäästötilaan (esim. paristolla toimiva lämpömittarit tai kytkin).

4.1.2 Tilaaminen (subscribe)

Asiakas saa tiedon välittäjältä tilaamalla tietyn aiheen. Tilattaessa aihe voi muodostua jokerimerkeistä, mikä mahdollistaa usean aiheen tilaamisen yhdellä tilauksella (ks. liite 7). Tarkemmin aiheesta kerrotaan kappaleessa 4.1.3. Tilatessa on mahdollista tilata useita aiheita yhdellä MQTT-viestillä. Tilauksen viestirakenne ja toiminta ovat kuvattuna tarkemmin kappaleessa 4.5 (s. 66).

Tilaus on aina tilaajan ja välittäjän välinen sopimus, eikä tiedon julkaisija tiedä tilaajista mitään. Välittäjän ja tilaajan välillä on myös oma laatutaso, joka on kuvattu tarkemmin kappaleessa 4.1.5.

4.1.3 Viestin aihe (topic)

MQTT-protokollassa kaikki viestit lähetetään ja vastaanotetaan aiheen perusteella. Aiheessa voi olla useita tasoja, jotka erotellaan ”/”-merkillä. Tilatessa on mahdollista käyttää jokerimerkkiä (wildcard). Jokerimerkkejä on kaksi, joista ”+”-merkillä voi tilata kaikki tietyn tason viestit. Kuvitellaan, että Ilmatieteenlaitos keräisisi säätietoja kaupungeista ja kunnista MQTT-protokollaa käyttäen. Mittauslukemat julkaistaisiin aiheella: ”/maa/kaupunki/mittari”. Pirkkalan kaikki mittarit saataisiin tilaamalla aihe ”/Suomi/Pirkkala/+”. Toinen jokerimerkki on ”#”, millä voi korvata lopun aiheesta. Edellistä esimerkkiä hyväksikäyttäen Suomen kaikki säämittaukset saataisiin tilaamalla aihe: ”/Suomi/#”. Aiheen tasojen määrittelemisessä kannattaa olla huolellinen, sillä järkevästi määriteltynä jokerimerkkien käyttö tekee viestien tilaamisen helpoksi. Kannattaa myös pitää mielessä, että mikään ei estä julkaisemasta samaa tietoa useammalle eri aiheelle. Tarvittaessa mikään ei estä luomasta asiakasta, mikä uudelleen julkaisee viestejä uusille aiheille. (MQTT Essentials Part 5 - - n.d.; MQTT Version 3.1.1 2014, 57-58.)

Välittäjät saattavat julkaista sisäisiä tietoja, kuten yhdistettyjen asiakkaiden määrän ja muita tilastotietoja. MQTT-määrityksessä ”$SYS/” -alkuiset aiheet on varattu välittäjän tilastotietojen julkaisemiseksi. Kaikki välittäjät eivät tätä määritystä noudata ja aiheen eri tasotkin ovat välittäjäkohtaisia. Usein välittäjästä täytyy erikseen aktivoida tilastotietojen julkaisu. (MQTT Essentials Part 5 - - n.d.; MQTT Version 3.1.1 2014, 58.)

Mosquitto-välittäjä mahdollistaa välittäjien siltauksen. Siltauksessa on mahdollista asettaa lähteviin julkaisuihin etuliite aiheelle ja vastaavasti poistaa etuliite sisään tulevista viesteistä. Järjestely mahdollistaa viestien suodatuksen, sekä hierarkkisen rakenteen. Kotiverkossa olevan välittäjän aiheet liittyisivät omaan kotiin ja siltauksessa toiseen välittäjään aiheeseen lisättäisiin esimerkiksi asiakastunnus aiheen alkuun. Siltaus ei ole protokollan määrittelemä ominaisuus. (Cope 2017.)

4.1.4 Viestin hyötykuorma

Protokolla ei määritä formaattia hyötykuormalle, vaan jättää sen määrittelemisen sovelluskohtaiseksi (MQTT Version 3.1.1 2014, 21, 36). Teoreettinen viestin pituus on 268 435 455 tavua, mutta käytännössä välittäjä rajoittaa viestin maksimipituutta. Viestin aihe vähentää hyötykuormalle varattua tilaa. (MQTT Version 3.1.1 2014, 18-19.)

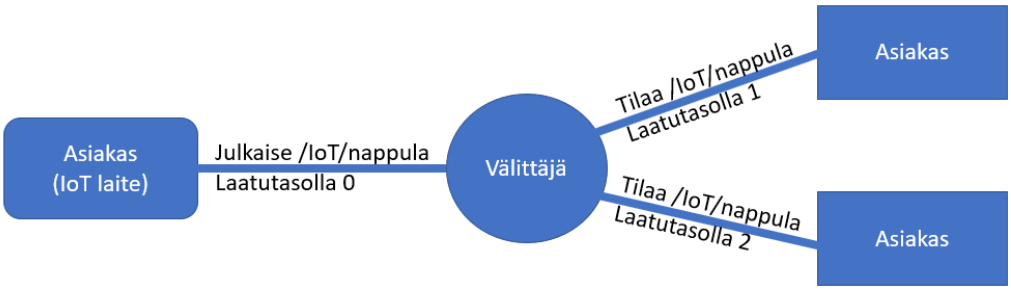
Viestin aiheiden avulla voidaan kategorisoida viesti hyvinkin tarkasti ja lähettää viestin sisältönä vain binäärinen arvo. Toisaalta aihe voidaan jättää lyhyemmäksi ja kategorisoida lähetettävä tieto hyötykuorman formaatilla. Verkkosivuilla monesti käytetään JSON-formaattia, minkä avulla tieto jäsennellään avaimien ja arvojen avulla. JSON-formaatissa oleva tieto on myös erittäin helppo tallettaa esimerkiksi Mongo-tietokantaan (esimerkki liitteessä 8).

4.1.5 Viestin laatutasoa (QoS)

MQTT määrittää kolme eri laatutasoa (QoS) asiakkaan ja välittäjän välille. Laatutasot määritellään tilaukselle ja julkaisulle erikseen, mistä esimerkki kuviossa 42. Laatutaso on aina välittäjän ja asiakkaan välinen sopimus (MQTT Version 3.1.1 2014, 36-47, 52-55). Laatutason toiminta on tarkemmin kuvattuna kappaleessa 4.4, jossa käsitellään julkaisua.

Laatutaso 0 lähettää tiedon, eikä jää odottamaan vastausta (at most once delivery). Laatutasolla 1 varmistetaan, että viesti saavuttaa vastaanottajan vähintään kertaalleen (at least once delivery). Laatutasolla 2 varmistetaan, että vastaanottaja saa viestin ja saa sen vain kerran (exactly once delivery). (MQTT Version 3.1.1 2014, 36-47, 52-55.)

Otsakkeessa on laatutason lisäksi DUP-bitti, joka liittyy laatutasoon. Tiedon julkaisija asettaa tämän bitin arvoon 0, silloin kun tieto julkaistaan ensimmäisen kerran. Uudelleen julkaistaessa julkaisija asettaa bitin arvoon 1, minkä perusteella vastaanottaja pystyy käsittelemään mahdollisesti useaan kertaan lähetetyt identtiset viestit. (MQTT Version 3.1.1 2014, 33-34.)



Kuvio 42. MQTT-laatutasot

4.1.6 Protokollan tietoturva

Protokollan määrityksissä on kaksi tietoturvaan liittyvää määritystä. Ensimmäinen on käyttäjätunnus ja salasana, jonka toiminta on tarkemmin esitettynä kappaleessa 4.3. Toinen on suojatun yhteyden porttisuositus (MQTT Version 3.1.1 2014, 27, 30, 52, 61).

MQTT-protokolla ei käytännössä tarjoa tietoturvaominaisuuksia vaan käyttäjän tunnistamiseen täytyy välittäjän toteuttaa oma menetelmä. Tiedon salaamisessa ja eheydessä protokolla luottaa alemman kerroksen tietoturvaan. Alemmalla kerroksella käytössä on joko TLS- tai IPSec-menetelmä. Mikään ei toisaalta estä kummankin menetelmän yhtäaikaista käyttöä. TLS- ja IPSec-menetelmät ovat tarkemmin esiteltynä kappaleessa 3.2.4.

Mosquitto-välittäjää käytettäessä sallitun käyttäjätunnuksia ja salasanoja voi hallita mosquitto_passwd-komennolla (Documentation | Mosquitto n.d., mosquitto_passwd). Mosquito mahdollistaa eri oikeuksien asettamisen eri aiheisiin (Documentation | Mosquitto n.d., mosquitto-conf/acl_file).

Varmenteiden ja avaiminen muodostaminen Mosquitto-välittäjän ja MQTT.FX-asiakasohjelmiston käyttöön on esitettynä liitteessä 4 (Welcome to the home of MQTT.fx n.d.). Mosquitto-välittäjän tarvitsemat asetukset on esitetty liitteessä 5. Vastaavasti MQTT.FX-ohjelmiston määritykset löytyvät liitteestä 6.

IPSec-tunnelointi menetelmän, eli VPN-menetelmän käyttöönotto Linux-ympäristössä on yksityiskohtaisesti esitettynä liitteessä 9. VPN-menetelmä on helppo ja luotettava tapa, mutta käyttökohteena soveltuu parhaiten välittäjien välisen yhteyden turvaamiseen. Raspberry Pi ja vastaavan tasoiset laitteistot kykenevät ongelmitta muodostamaan VPN yhteyden ja toimimaan välittäjinä.

4.1.7 Viimeinen tahto ja testamentti (Last Will/Testament)

Viimeinen tahto (last will) on mekanismi, millä asiakas voi määritellä välittäjälle julkaistavan viestin yhteyden katketessa. Viimeisestä tahdosta käytetään myös testamentti (testament) -nimitystä tai nimitystä viimeinen tahto ja testamentti (LWT). Viimeinen tahto määritellään yhteyden luomisen yhteydessä (ks. kappale 4.3). Viimeinen tahto tarvitsee aiheen, joka on vapaasti valittavissa. Aihe voi olla sama kuin mihin asiakas yleisesti julkaisee tietoja, mutta aihe voi olla myös eri. Mielekäs viimeisen tahdon aiheen käyttö on tapauskohtaista.

Välittäjän täytyy lähettää viimeinen tahto välittömästi, kun yhteys asiakkaaseen on katkennut. Välittäjä ei lähetä viimeistä tahtoa, jos yhteyden päättäminen on pyydetty asiakkaan toimesta. (MQTT Version 3.1.1 2014, 26-27).

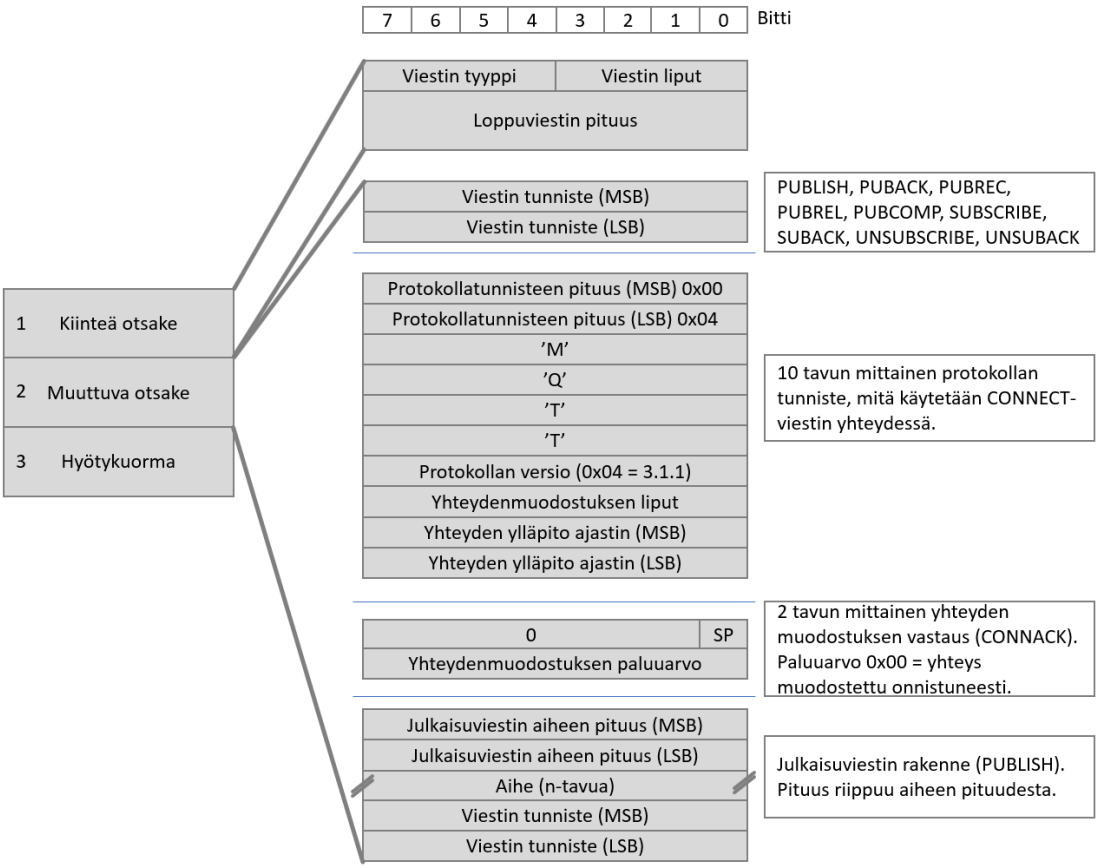
Viimeinen tahto -menetelmän avulla voi helposti luoda IoT-laitteille olemassaolo-aiheen. Asiakkaan kytkeytyessä julkaistaan esimerkiksi aiheeseen ”/iot/tila/laitex” arvo ”1”. Viimeinen tahto voi julkaista arvon ”0” samaan aiheeseen. Tällä yksinkertaisella mekanismilla voidaan kaikkien asiakkaiden yhteyden tilaa seurata tilaamalla yksi aihe ”/iot/tila/+”. Viimeinen tahdon lisäksi asiakkaan täytyy julkaista viimeinen tahto aiheeseen ”0” silloin, kun yhteys päätetään suunnitellusti. Yksittäisen sensorin tilanteessa voi olla mielekästä, että viimeinen tahto julkaistaan samaan aiheeseen kuin mittaustuloksetkin. Esimerkkilaite voisi olla lämpötilasensori, joka julkaisisi viimeisenä tahtona ”NA” tai mitta-alueen ulkopuolelta olevan arvon.

4.2 Protokollan viestirakenne yleisesti

4.2.1 Yleinen viestikehys

MQTT-protokollan viestirakenne koostuu kolmesta osasta (ks. kuvio 43):

1. Ensimmäinen osa on kaikille viesteille pakollinen niin sanottu kiinteä otsake. Kiinteä otsake ei kuitenkaan ole kiinteän mittainen, vaan pituuskentän pituus vaihtelee MQTT-viestin pituuden mukaan. Pituuskenttä vie lyhimmillään 1 tavun ja maksimissaan 3 tavua. Pituuskentän vaatima tavumäärä on esitettynä taulukossa 1.
2. Toisena osana on muuttuva otsake, jolle löytyy 5 erilaista vaihtoehtoa. Kuvio 43 on esitettynä 4 vaihtoehtoa ja 5. vaihtoehto on, ettei muuttuvaa otsaketta ole viestissä mukana. Muuttuva otsake puuttuu kuviossa esitetyistä viesteistä silloin kun laatutaso on 0.
3. Hyötykuorman sisältö riippuu viestistä, esimerkiksi kuittausviesteillä ei ole hyötykuormaa.



Kuvio 43. MQTT-viestirakenne (MQTT Version 3.1.1 2014, 17-50)

Taulukko 1. MQTT-viestin pituuskenttä (MQTT Version 3.1.1 2014, 29-30)

Pituus tavua	Viestin pituus alkaen tavusta	Viestin pituus loppuen tavuun
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 382 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0xF7)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

4.2.2 Viestien tyypit

MQTT-protokolla sisältää yhteensä 14 viestiä, jotka ovat listattuna taulukossa 2. Taulukossa Arvo-sarake viittaa kehyksessä olevan lipun arvoon ja Kehys-sarake viittaa kehyksen tyyppiin (M=muuttuva tai K=kiinteä). Viesteistä PUBREC, PUBREL ja PUBCOMP liittyvät nollaa isompiin laatutasoihin. Yksinkertaisimmillaan laatutason 0 täyttävän asiakasohjelmiston on pakko toteuttaa CONNECT- ja CONNACK-viestit. PINGREQ ja PINRESP-viestit riippuvat yhteyden luomisessa asetetuista parametreista. Yksinkertaisimmillaankin asiakas/laite haluaa joko lähettää tai vastaanottaa tietoja, jolloin PUB* tai SUB*-viestit täytyy toteuttaa. CONN*, PING* ja DISCONNECT-viestit esitetään tarkemmin kappaleessa 4.3. PUB*-viestit käydään läpi julkaisuja käsittelevässä kappaleessa 4.4. Kappaleessa 4.5 käsitellään tiedon tilaus, eli SUB* ja UNSUB*-viestit.

Taulukko 2. MQTT-viestien tyypit (MQTT Version 3.1.1 2014, 17-18)

Viesti	Arvo	Suunta	Selite	Kappale
CONNECT	1	Asiakas -> välittäjä	Asiakas ottaa yhteyttä serveriin	4.3
CONNACK	2	Välittäjä -> asiakas	Välittäjä vastaa yhteydenotto-pyyntöön	4.3
PUBLISH	3	Asiakas <-> välittäjä	Asiakas julkaisee viestin tai välittäjä lähettää asiakkaan viestin eteenpäin (julkaisee)	4.4 ja 4.5
PUBACK	4	Asiakas <-> välittäjä	Asiakas/välittäjä kuittaa viestin julkaisun	4.4
PUBREC	5	Asiakas <-> välittäjä	Julkaisu vastaanotettu	4.4
PUBREL	6	Asiakas <-> välittäjä	Julkaisu vapautettu	4.4
PUBCOMP	7	Asiakas <-> välittäjä	Julkaisu suoritettu	4.4
SUBSCRIBE	8	Asiakas -> välittäjä	Asiakas tilaa aiheen palvelimelta	4.5
SUBACK	9	Välittäjä -> asiakas	Välittäjä kuittaa tilauksen	4.5
UNSUBSCRIBE	10	Asiakas -> välittäjä	Aiheen tilauksen peruuttaminen	4.5
UNSUBACK	11	Välittäjä -> asiakas	Välittäjä kuittaa aiheen peruutuksen	4.5
PINGREQ	12	Asiakas -> välittäjä	Asiakas lähettää palvelimelle PING-pyynnön	4.3
PINGRESP	13	Välittäjä -> asiakas	Välittäjä kuittaa PING-pyynnön	4.3
DISCONNECT	14	Asiakas -> välittäjä	Asiakas katkaisee yhteyden	4.3

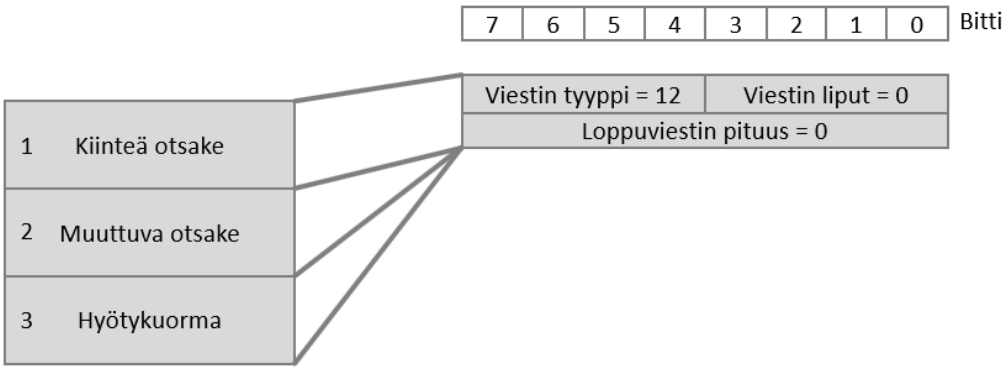
4.3 Yhdistäminen ja yhteyden päättäminen

MQTT-yhteyden muodostamisessa voi asettaa seuraavia määrityksiä:

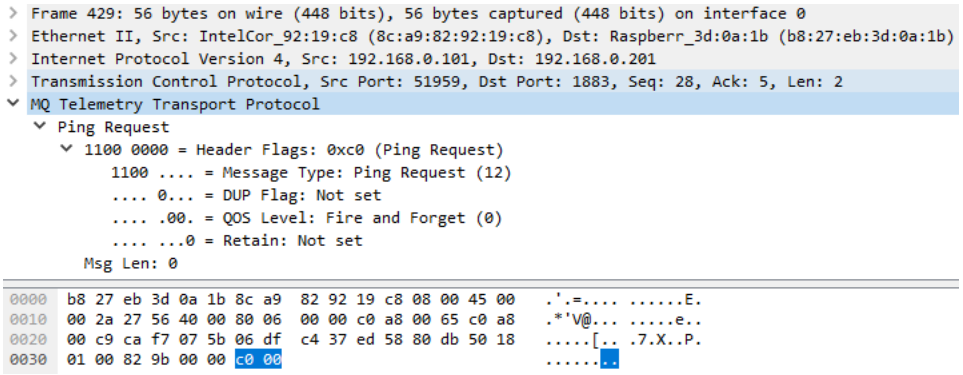
- Asiakastunnus (Client ID)
- Asetus yhteyden tarkistusaikaväli (keep alive interval)
- Puhdas yhteys (clean session)
- Käyttäjätunnus ja salasana
- Yhteyden salaus
- Viimeinen tahto (last will and testament)

Asiakastunnus yksilöi yhdistettävän laitteen. Välittäjän täytyy sallia vähintään 23 merkkiä pitkä UTF-8 formaatissa oleva asiakastunnus. Välittäjä voi sallia myös pidemmät tunnukset (65535 tavua), sekä 0-pituisen tunnuksen. Tunnuksen ollessa 0-pituinen, täytyy välittäjän itse määrittää laitteelle asiakastunnus. Puhdas yhteyslippu täytyy asettaa, jos pituus on 0. (MQTT Version 3.1.1 2014, 10).

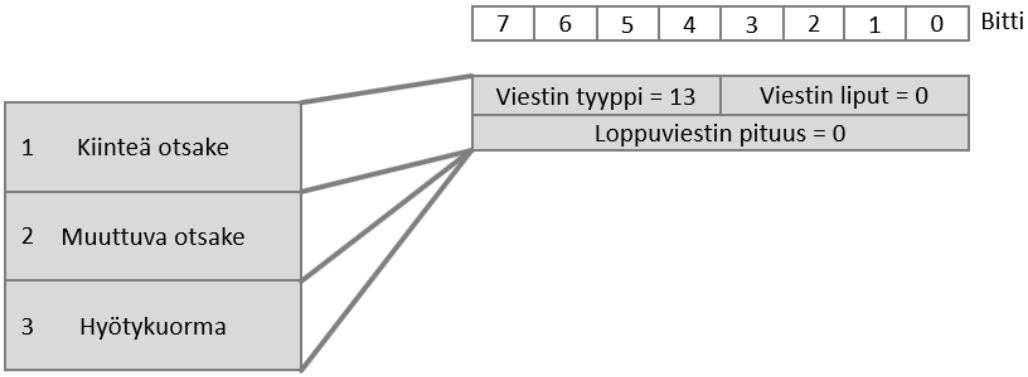
Yhteyden ylläpidon tarkistusaikaväli asettaa yhteydelle pisimmän mahdollisen aikavälin, minkä sisällä täytyy siirtää viestejä. Välittäjän täytyy varmistaa, että asiakas on lähettänyt jonkin viestin tai erityisen PINGREQ-viesti palvelimelle. Viesti täytyy lähettää, ennenkö asetettu määräaika umpeutuu. Palvelimen täytyy vastaa PINGRESP-viestillä asiakkaan PINGREQ-viestiin. Viestejä ei tarvitse lähettää, jos yhteyden aikana on muuta liikennettä. Tarkastusvälin avulla sekä välittäjä, että laite päättävät yhteyden voimassaolosta. Aikaväli voidaan asettaa nollaksi, jolloin tarkastuksia ei tehdä lainkaan. Yhteys katkaistaan, jos viestejä ei ole yhteydessä liikkunut 1,5 * tarkistusaikaväli. Yhteyden tarkistusaikaväli määritellään yhteydenotto viestissä 16 tavulla (ks. kuvio 52). PINGREQ-viestin rakenne on kuvattuna kuviossa 44 ja 45. PINGRESP-vastaus viesti on esitetty kuviossa 47 ja 54. Viestit ovat vain 2 tavun mittaisia, eli lyhimmän mahdollisen kiinteän otsakkeen mittaisia. 1. tavu neljä ensimmäistä bittiä määrittävät viestin tyypin ja toinen tavu pituuden mikä on aina nolla. 0xC* tyyppi on PINGREQ-viestin ja 0xD* PINGRESP-viestin tunniste. Yhteydenottoviestin tunniste on vastaavasti 0x1*. (MQTT Version 3.1.1 2014, 27-28.)



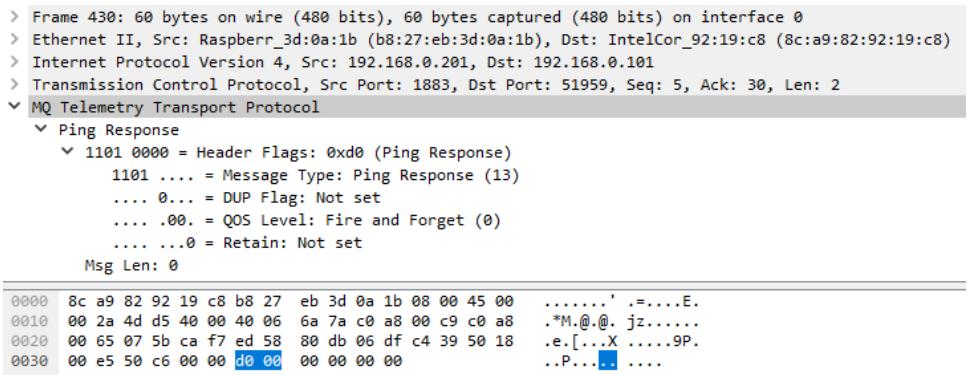
Kuvio 44 MQTT-PINGREQ -otsake (MQTT Version 3.1.1 2014, 48)



Kuvio 45. MQTT PINGREQ -viestin TCP/IP rakenne



Kuvio 46 MQTT-PINGRESP -otsake (MQTT Version 3.1.1 2014, 48–49)



Kuvio 47. MQTT PINGRESP -viestin TCP/IP rakenne

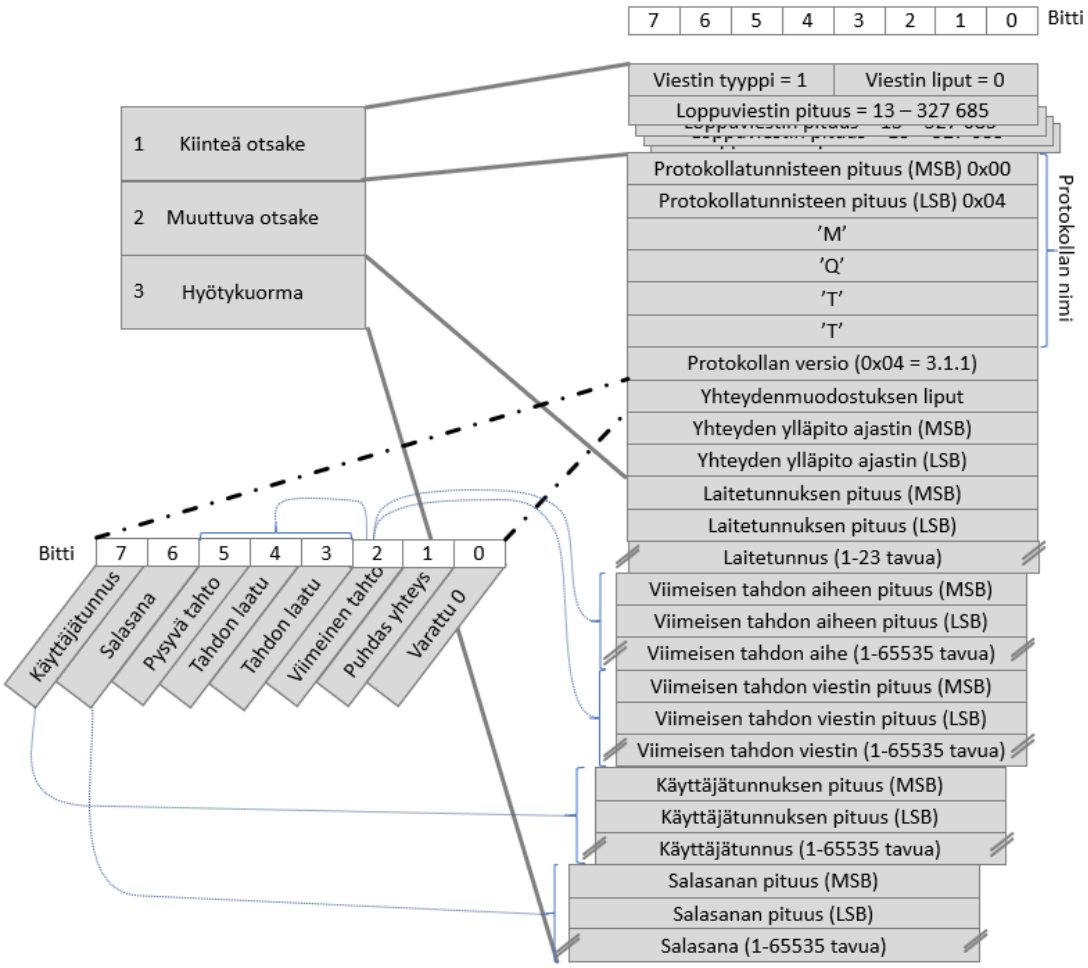
Puhdas yhteyslippu mahdollistaa aiemman yhteyden palauttamisen. Tämä tarkoittaa, että mahdolliset aiemmin luodut tilaukset pysyvät voimassa. Puskurissa olevat viestit myös välitetään asiakkaalle, riippuen laatuluokasta. (MQTT Version 3.1.1 2014, 24-26.)

Käyttäjätunnuksen ja salasanan käyttö on protokollamäärityksessä jätetty toteutuskohtaiseksi. Protokolla mahdollistaa käyttäjätunnuksen ja salasanan välittämisen selkokielisenä välittäjälle, mutta ei määrittele, mitä kyseisillä tiedoilla pitää tehdä. Käyttäjätunnus esitetään hyötykuormassa UTF-8 merkkijonona, mitä edeltää kahden tavun mittainen käyttäjätunnuskentän pituus. Käyttäjätunnusta seuraa binaariformaatissa salasanan pituus kahdella tavulla esitettynä ja sen jälkeen salasana. (MQTT Version 3.1.1 2014, 27, 30, 61.)

Yhteyden salausmekanismia ei protokollassa ole määriteltynä, vaan protokolla luottaa kuljetuskerroksen tietoturvaan. Salatun yhteyden muodostaminen määritetään käyttämällä ennalta määrättyä porttia (8883), missä suositellaan käytettäväksi TLS-menetelmää (kappale 3.2.4). Vaihtoehtoisena menetelmänä suositellaan VPN-yhteyttä (kappale 3.2.4). Protokollan otsakkeissa ei ole bittiä/bittejä varattuna salauksen käyttöön. (MQTT Version 3.1.1 2014, 60-64.)

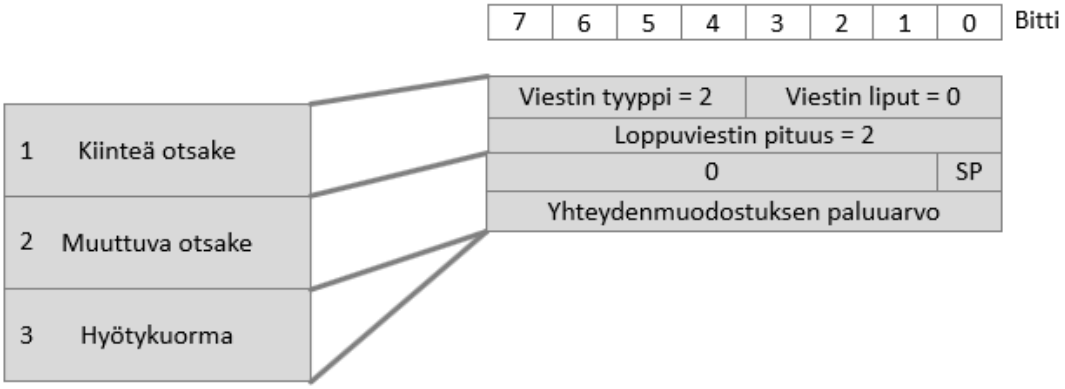
Viimeinen tahto mahdollistaa yhteyden katkeamisen julkaisun aiheen tilaajille. Julkaistavalle viestille täytyy määritellä sekä aihe, että laatutaso. Viimeinen tahto mekanismina on tarkemmin määriteltynä kappaleessa 4.1.7 (s. 46).

Yhteydenottoviestin eri parametrit muodostavat kuviossa 48 esitetyn viestin. Muuttuvassa otsakkeessa olevilla lipuilla määritetään, mitä parametreja hyötykuormaan tulee. Hyötykuormaan tulevat parametrit täytyy olla seuraavassa järjestyksessä: viimeisen tahdon aihe, viimeisen tahdon viesti, käyttäjätunnus ja salasana. (MQTT Version 3.1.1 2014, 23-30.)



Kuvio 48. MQTT-yhteydenottoviestin otsake (MQTT Version 3.1.1 2014, 23-30)

Yhteydenottoon vastataan CONACK-viestillä (ks. kuvio 49). Viestissä on SP-lippu (Session Present), jos yhteys on pyydetty ”Puhdas yhteys” -lipun arvolla 1 ja välittäjä on pystynyt palauttamaan yhteyden. Taulukossa 3 on esitettynä yhteydenmuodostuksen paluuarvot, joista arvo 0 merkitsee onnistunutta yhteyttä.

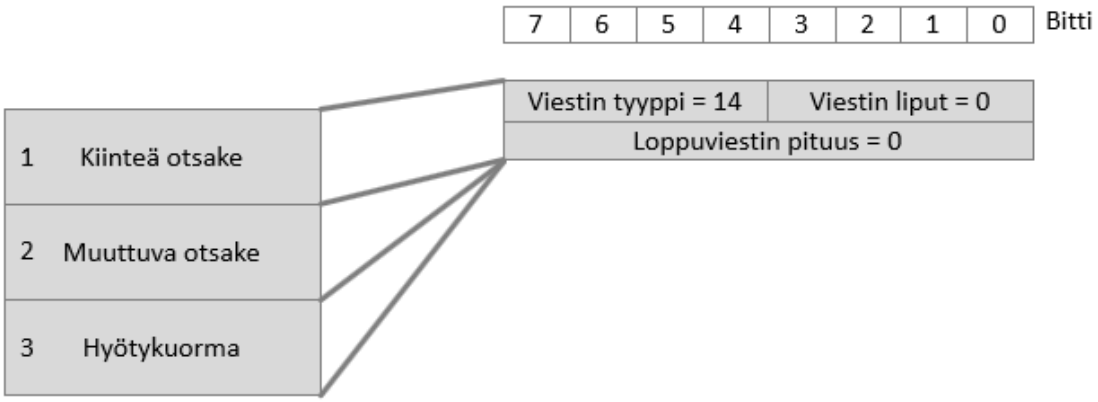


Kuvio 49. MQTT CONNACK -vastaus otsake (MQTT Version 3.1.1 2014, 31-33)

Taulukko 3. CONACK-viestin yhteydenmuodostuksen paluuarvot

Arvo	Selite
0	Yhteys muodostettu onnistuneesti
1	Yhteys hylätty, protokollan versio ei ole tuettu
2	Yhteys hylätty, laitetunnusta ei hyväksytty
3	Yhteys hylätty, välittäjä ei vastannut
4	Yhteys hylätty, käyttäjätunnus ja/tai salasana väärä
5	Yhteys hylätty, ei oikeutta muodostaa yhteyttä
6-255	Varattu

Yhteyden päättämisen viestirakenne on varsin yksinkertainen. Viestissä on vain kiinteä otsake, missä viestin tyypin arvo on 14 (ks. kuvio 50) (MQTT Version 3.1.1 2014, 49).



Kuvio 50. MQTT DISCONNECT -vastaus otsake (MQTT Version 3.1.1 2014, 49)

Tässä luvussa mainituilla yhteyden määritysarvoilla saadaan useita erilaisia vaihtoehtoja yhteyden luomiselle. Luvuissa 4.3.1 - 4.3.4 käydään muutama yleisin vaihtoehto yksityiskohtaisesti läpi.

4.3.1 Lähettäminen oletusasetuksilla

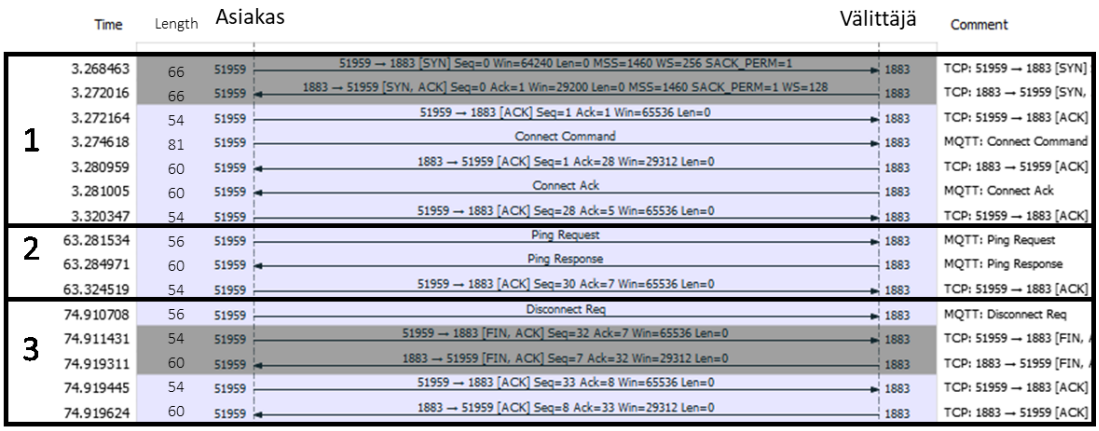
Oletusasetuksilla olevan yhteyden muodostus koostuu yleensä, mutta ei välttämättä seuraavista asetuksista:

- Asiakastunnus ”IoTProtokolla”
- 60s Asetus yhteyden ylläpidon tarkistamiselle
- Puhdas yhteys

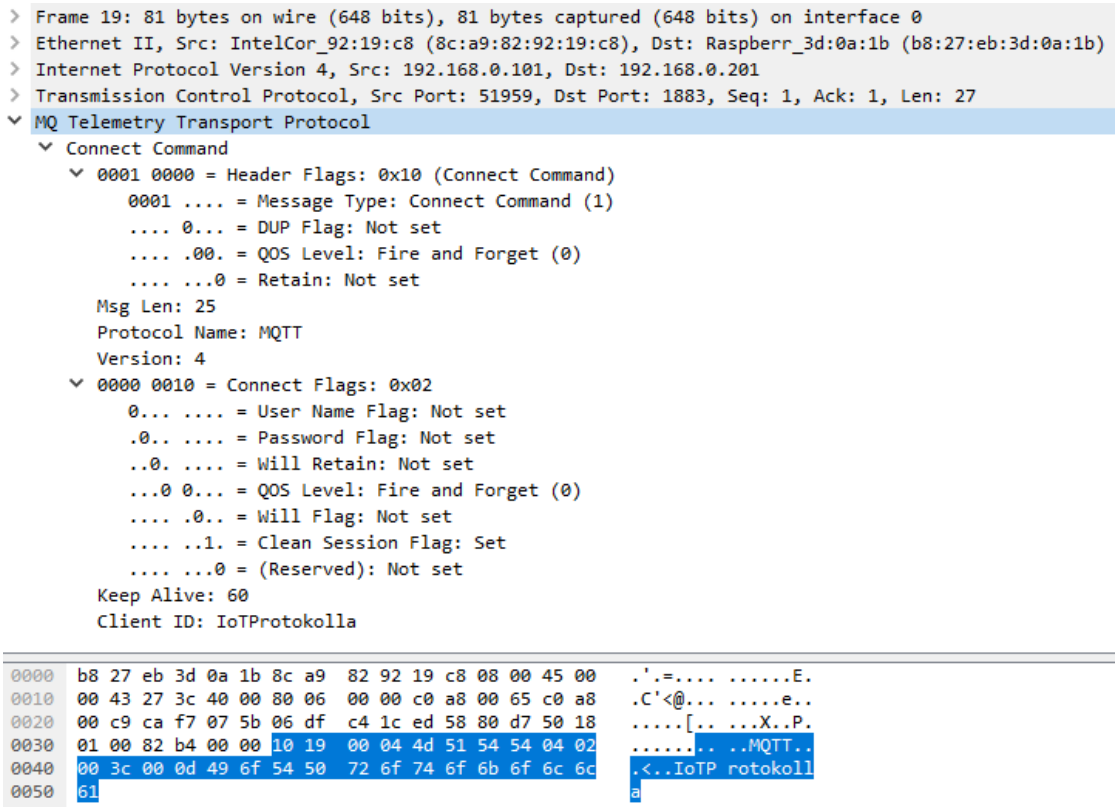
Yhteyden muodostus ja päättämien koostuu oletusasetuksia käyttäen kolmesta eri osasta, mitkä ovat eriteltynä myös kuviossa 51.

1. yhteyden muodostaminen aloitetaan TCP-portin avaamisella MQTT-porttiin (1883). TCP-yhteyden avauduttua lähetetään MQTT yhteydenottoisesti (kuviot 52 ja 53), mihin välittäjä vastaa kuittausviestillä. Tavumääräisesti IPv4-protokollaa käyttävässä lähiverkossa lähetetään yhteensä 441 tavua, missä muuttuvana tekijänä on vapaasti määriteltävä asiakastunnus.
2. Yhteyden ylläpidon tarkistusaikavälin aika esimerkissä on 60 sekuntia. Esimerkissä yhteydellä ei ole muuta liikennettä, joten asiakas lähettää PINGREQ-viestin, mihin välittäjä vastaa PINGRESP-viestillä (ks. kuvio 45 ja kuvio 47). Tätä vaihetta ei ole nähtävissä, jos asetetaan yhteyden ylläpidon tarkistusväli nollaksi.

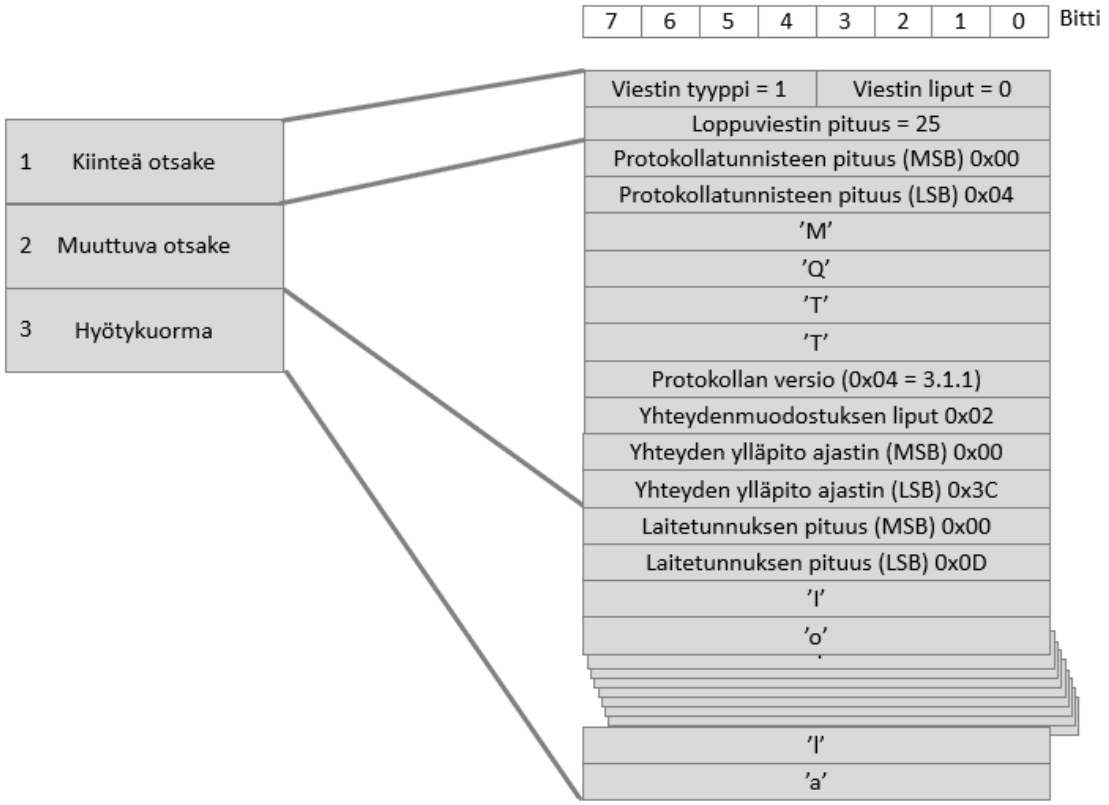
3. Yhteyden katkaisemisessa laite lähettää MQTT-viestin yhteyden katkaisemiseksi, missä viestin tyyppi on yhteyden päättäminen (ks. kuvio 54). Otsakkeessa päättämistä kuvaa arvo 0xE*.



Kuvio 51. MQTT Yhteyden muodostus- ja päätössekvenssi oletusasetuksilla (tarkempi kuva liitteessä 0)



Kuvio 52. MQTT-yhteydenottoviestin TCP/IP -rakenne



Kuvio 53. MQTT-yhteydenonttoviestin otsakkeet oletusasetuksilla

```
> Frame 521: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
> Ethernet II, Src: IntelCor_92:19:c8 (8c:a9:82:92:19:c8), Dst: Raspberr_3d:0a:1b (b8:27:eb:3d:0a:1b)
> Internet Protocol Version 4, Src: 192.168.0.101, Dst: 192.168.0.201
> Transmission Control Protocol, Src Port: 51959, Dst Port: 1883, Seq: 30, Ack: 7, Len: 2
v MQ Telemetry Transport Protocol
  v Disconnect Req
    v 1110 0000 = Header Flags: 0xe0 (Disconnect Req)
        1110 .... = Message Type: Disconnect Req (14)
        .... 0... = DUP Flag: Not set
        .... .00. = QOS Level: Fire and Forget (0)
        .... ...0 = Retain: Not set
      Msg Len: 0

0000  b8 27 eb 3d 0a 1b 8c a9  82 92 19 c8 08 00 45 00   .'.=.... ......E.
0010  00 2a 27 5e 40 00 80 06  00 00 c0 a8 00 65 c0 a8   .*'^@... .....e..
0020  00 c9 ca f7 07 5b 06 df  c4 39 ed 58 80 dd 50 18   .....[.. .9.X..P.
0030  01 00 82 9b 00 00 e0 00                           ........
```

Kuvio 54. MQTT-yhteyden lopetusviestin TCP/IP -rakenne

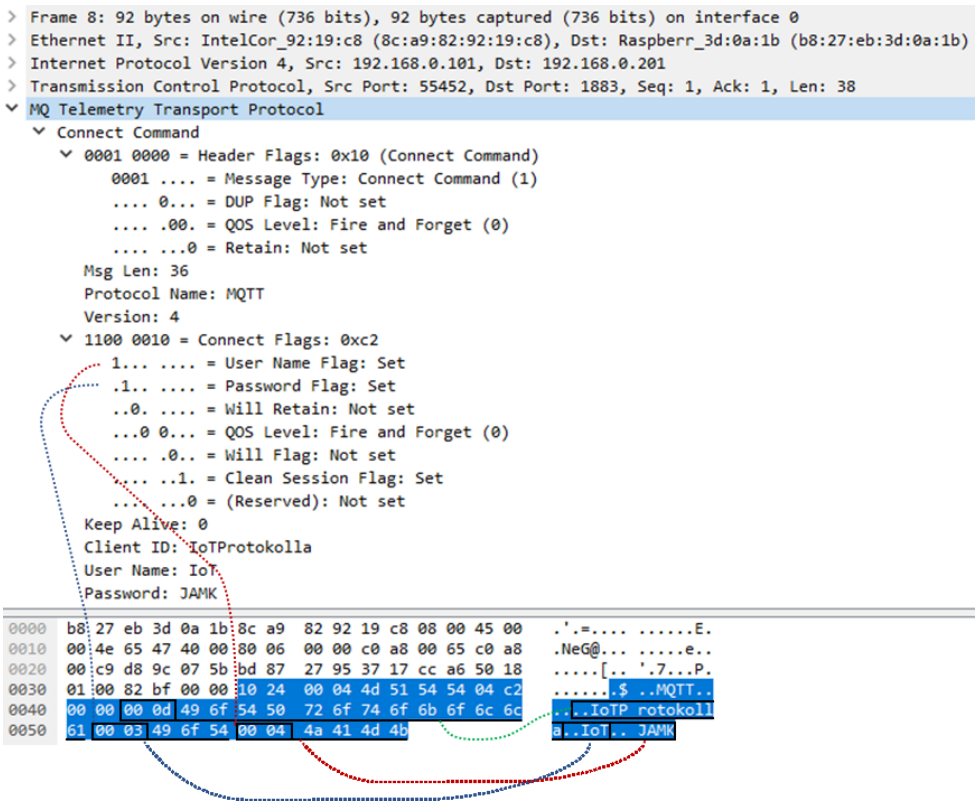
4.3.2 Lähettäminen käyttäjätunnuksella ja salasanalla

Yhteyden asetukset:

- Asetus yhteyden tarkistusaikaväli 0 sekuntia
- Puhdas yhteys
- Käyttäjätunnus ”IoT”

- Salasana ”JAMK”

Kuviossa 55 käy selvästi ilmi, miksi käyttäjätunnuksen ja salasanan käyttäminen ei käytännössä lisää tietoturvaa salaamattomassa yhteydessä. Kuviosta ilmenee myös, miten viestit määritellään MQTT-viestin hyötykuormassa. Parametreille on määritelty järjestys (ks. kuvio 48 s. 53) ja ennen jokaista parametria ilmoitetaan parametrin pituus kahdella tavulla. Pituusmääre noudattaa yleisesti verkkoliikenteessä käytettyä Big Endian -formaattia. Big Endian -formaatissa luku muodostuu suoraan arvosta, kun taas prosessoreissa yleisesti käytetään Little Endian -formaattia. Little Endian -formaatissa tavut ovat käännettynä päinvastaiseen järjestykseen (ensimmäinen on viimeinen, toiseksi ensimmäinen toiseksi viimeinen jne.). Toteutuksessa täytyy ottaa huomioon protokollan pituuskenttien formaatti sekä prosessorin käyttämä formaatti lukujen käsittelemiseen. Yhteyden muodostaminen ja päättäminen noudattavat kuviossa 51 esitettyä sekvenssiä, ilman kohtaa 2.



Kuvio 55. MQTT-yhteydenottoviesti käyttäjätunnuksella ja salasanalla

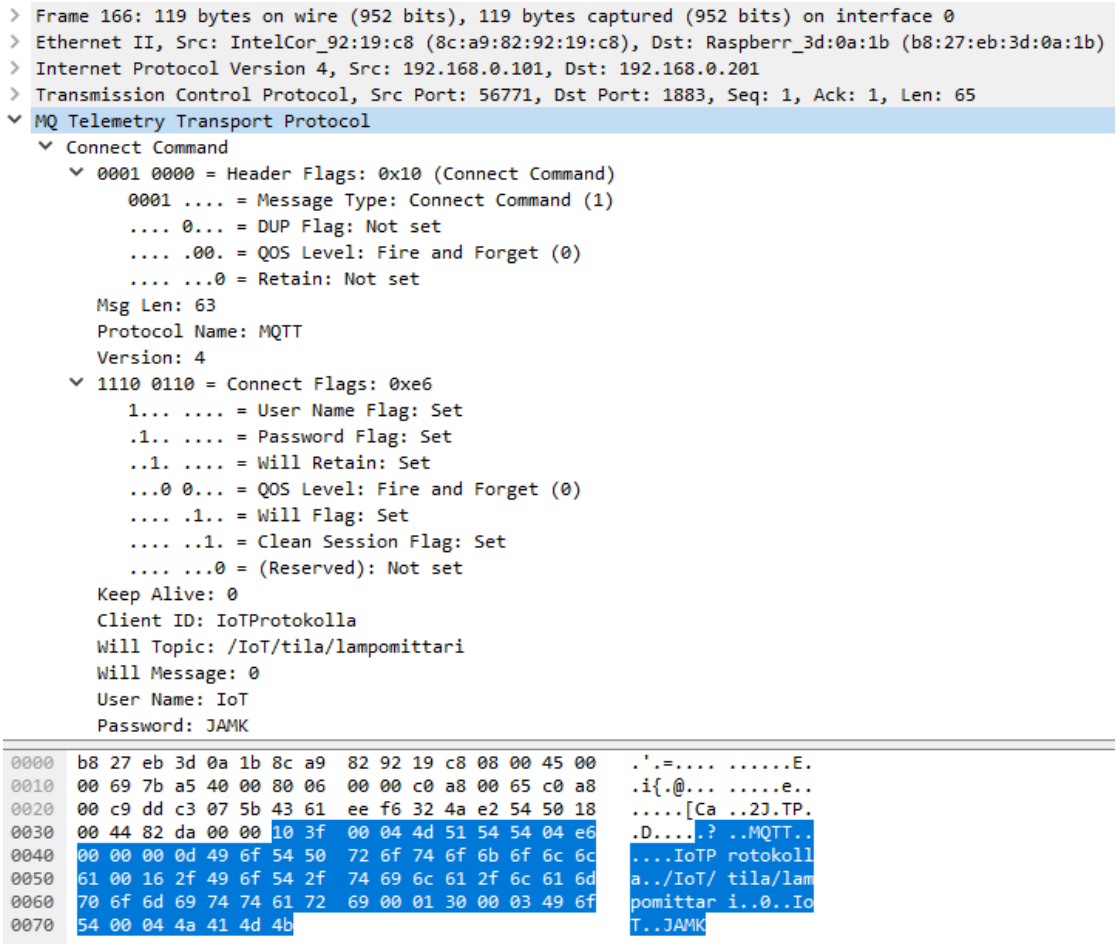
4.3.3 Lähettäminen käyttäjätunnuksella, salasanalla ja viimeisellä tahdolla

Yhteyden asetukset:

- Asetus yhteyden tarkistusaikaväli 0 sekuntia
- Puhdas yhteys
- Käyttäjätunnus ”IoT”
- Salasana ”JAMK”
- Viimeinen tahto
    - Aihe: /IoT/tila/lampomittari
    - Viesti: 0
    - Laatutaso (QoS): 0
    - Pysyvä viesti (Retain): kyllä

Ero aikaisempiin esimerkkeihin ilmenee kuviossa 56. Kuviossa on viimeisen tahdon liput, sekä hyötykuormaan sijoitettu viimeisen tahdo aihe ja aiheen sisältö. Viimeinen tahto sijoittuu hyötykuormassa asiakastunnuksen ja käyttäjätunnuksen väliin. Yhteyden muodostaminen ja päättäminen noudattavat kuviossa 51 esitettyä sekvenssiä, ilman 2. kohtaa.

Viimeiselle tahdolle voi myös määrittää oman laatutason. Viimeisen tahdon laatutaso saattaisikin olla järkevää asettaa vähintään tasolle 1, millä varmistetaan viimeisen tahdon läpimeno välittäjälle. Laatutason lisäksi viimeiselle tahdolle voi asettaa pysyvyys-lipun. Tällä lipulla välittäjä tallettaa viimeisen tahdon ja välittää sen aina kaikille viimeisen tahdon aiheen tilaajille. Viesti lähetetään myös niille tilaajille, jotka muodostavat yhteyden viimeinen tahto viestin lähetyksen jälkeen (MQTT Version 3.1.1 2014, 26-27, 34-35).



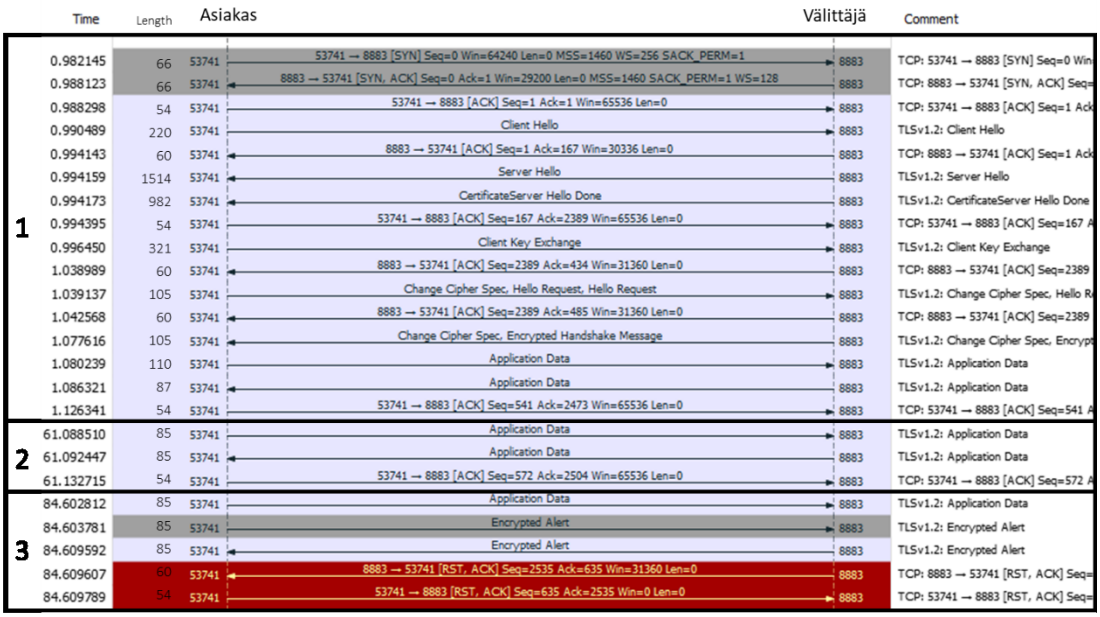
Kuvio 56. MQTT-yhteydenotto (käyttäjätunnus, salasana ja viimeinen tahto)

4.3.4 TLS-suojatun yhteyden muodostaminen

TLS-suojatussa yhteydessä asetukset ovat muuten samat kuin oletusasetuksissa paitsi, että tällöin käytetään TLS-porttia 8883 ja TLSv1.2 salausta. Salaukseen tarvittavien varmenteiden ja avaimien luonti on esitetty kappaleessa 4.1.6 ja liitteessä 4. Liitteessä 4 on esitettynä Mosquitto-välittäjän asetukset ja vastaavasti liitteessä 6 esimerkissä käytetyn MQTT.FX-asiakasohjelmiston TLS-asetukset.

Kuviossa 57 esitetty sekvenssi esitetään salattuna kuviossa 51. Merkittävin ero näiden sekvenssien välillä on yhteyden muodostamisessa (ks. kuvio 57, kohta 1), missä TLS-menetelmä aiheuttaa merkittävästi enemmän tiedonsiirtoa. Kuvion 57 ”Application Data” -viestit vastaavat kuviossa 51 esitettyjä viestejä.

TLS-menetelmällä yhteyden muodostamiseen menee esimerkissä 4083 tavua ja vastaavasti salaamattomassa yhteydessä 441 tavua (3642 enemmän salattuna). PING-viestit vaatii 54 tavua enemmän salattuna (ks. kuvio 57, kohta 2). Yhteyden päättäminen vaatii vastaavasti 85 tavua enemmän salattuna (ks. kuvio 57, kohta 3).



Kuvio 57. MQTT-sekvenssi yhteyden luomisesta ja päättämisestä TLS-salauksella (tarkempi kuva liitteessä 3)

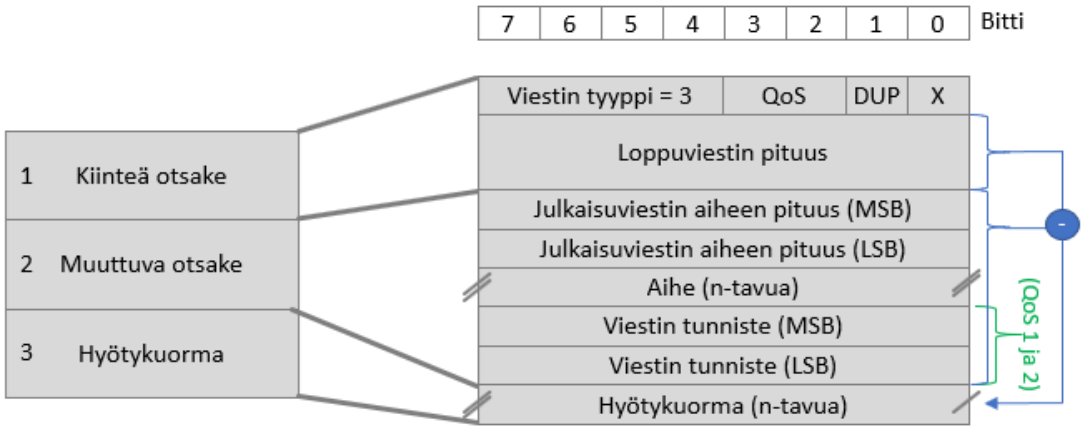
4.4 Viestin julkaiseminen

4.4.1 Yleinen viestirakenne

Julkaisuviestissä on normaali kiinteä otsake, jossa viestin tyyppi on arvoltaan 3 (ks. kuvio 58). Julkaisuviestissä käytetään hyväksi myös kiinteän otsakkeen lippuja DUP ja QoS. Liput liittyvät laatutasoihin 1 ja 2, jotka käsitellään kappaleissa 4.4.3 ja 4.4.4.

Julkaisuviestin muuttuva otsake pitää sisällään aiheen pituuden kahdella tavulla esitettynä (0 – 65kt) ja aiheen merkkijonona. Laatutasolla 1 ja 2 muuttuva otsake sisältää myös viestin tunnisteen (ks. kappale 4.4.3 ja 4.4.4).

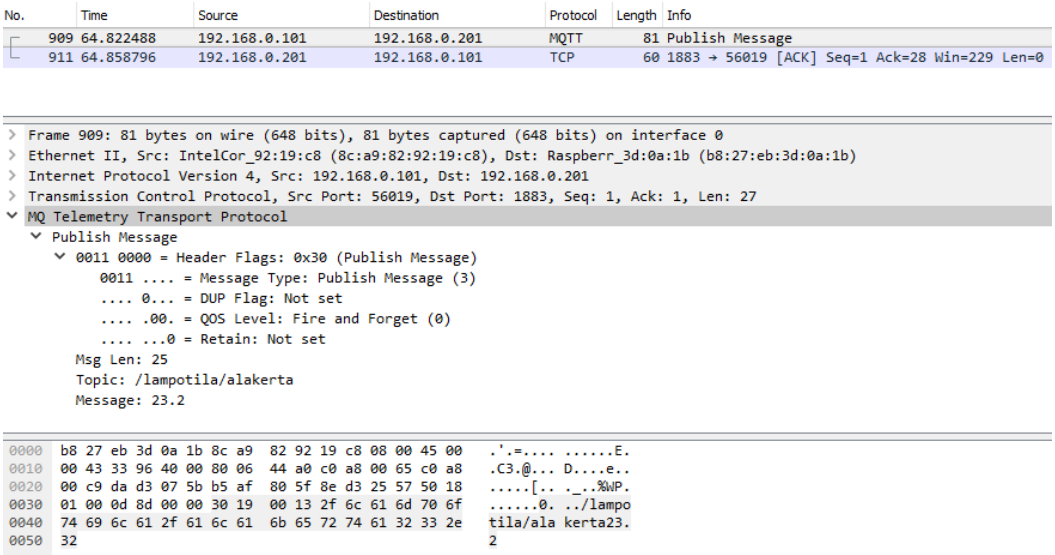
Hyötykuorma sisältää sovelluskohtaisen tietosisällön. Hyötykuorman pituus lasketaan vähentämällä kiinteän otsakkeen loppuviestin pituudesta muuttuvan otsakkeen pituus (julkaisuvistin aiheen pituus 2 tavua, aihe ja mahdollinen viestin tunniste 2 tavua).



Kuvio 58. MQTT-viestinjulkaisuotsake (PUBLISH) (MQTT Version 3.1.1 2014, 33-37)

4.4.2 Laatutasolla 0 (QoS 0)

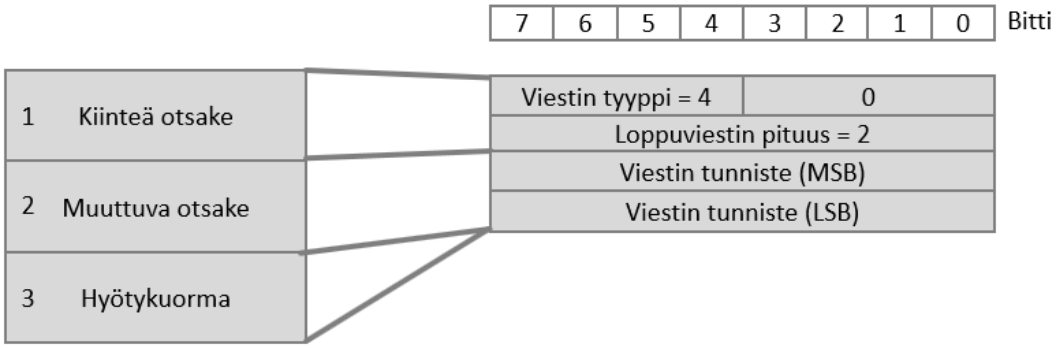
Laatutaso 0 varmistaa, että viesti lähetetään enintään kertaalleen (at most once / fire and forget), eli viestin läpimenoa ei tarkisteta (MQTT Version 3.1.1 2014, 52). Laatutaso 0:lla julkaisu sisältää vain yhden viestin ja se on julkaisuviesti. Julkaisuviestin sisältö on esitettynä kuvioissa 59 ja 58.



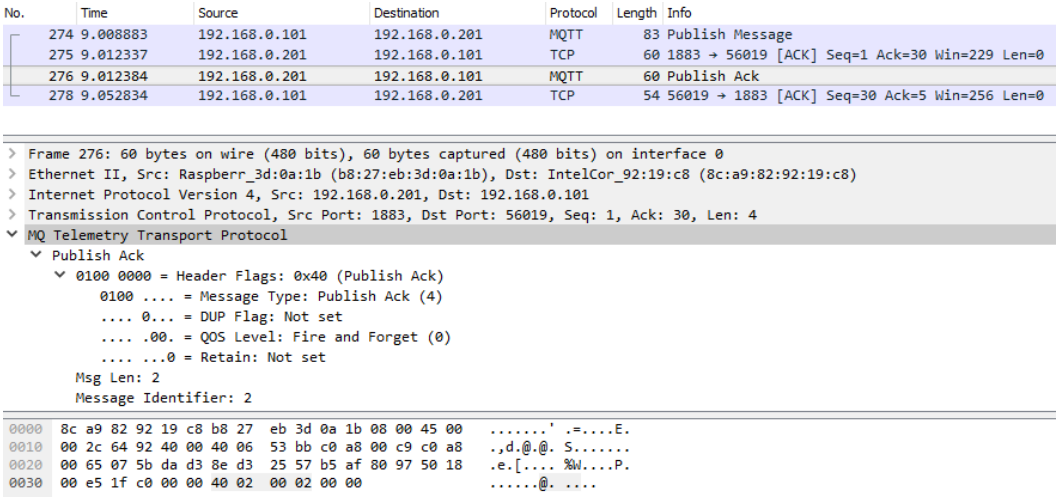
Kuvio 59. MQTT-viestin julkaisu laatutasolla 0 (TCP/IP rakenne)

4.4.3 Laatutasolla 1 (QoS 1)

Laatutasolla 1 varmistetaan, että julkaisu toteutuu vähintään kerran (at least once). Asiakasohjemisto tai välittäjä julkaisee tiedon ja odottaa vastaanottajalta kuittausta (PUBACK) oikealla viestitunnisteella (ks. kuviot 60 ja 61). Välittäjä lähettää kuittauksen sen jälkeen kun se on aloittanut julkaisun edelleenlähetyksen tilaajille. Laatutasoon 0 nähden MQTT-lipuista QoS-taso saa arvon 1 ja hyötykuormassa on viestin tunniste. Asiakas lähettää viestin uudelleen uudella viestitunnisteella, jos välittäjä ei lähetä kuittausviestiä ajoissa. Varsinaista uudelleenlähetysaikaa MQTT-määrityksessä ei anneta. Tällä menetelmällä asiakas voi lähettää useita julkaisuja, mitkä myös edelleenlähetetään tilaajille. (MQTT Version 3.1.1 2014, 53.)



Kuvio 60. MQTT-viestinjulkaisun kuittausotsake (PUBACK)(MQTT Version 3.1.1 2014, 37)

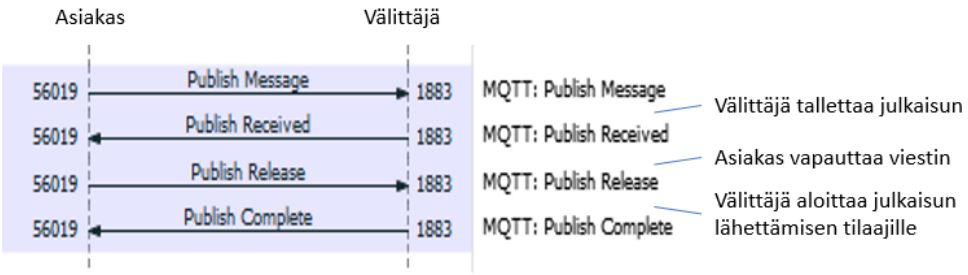


Kuvio 61. MQTT-julkaisun kuittaus laatutasolla 1 (TCP/IP rakenne)

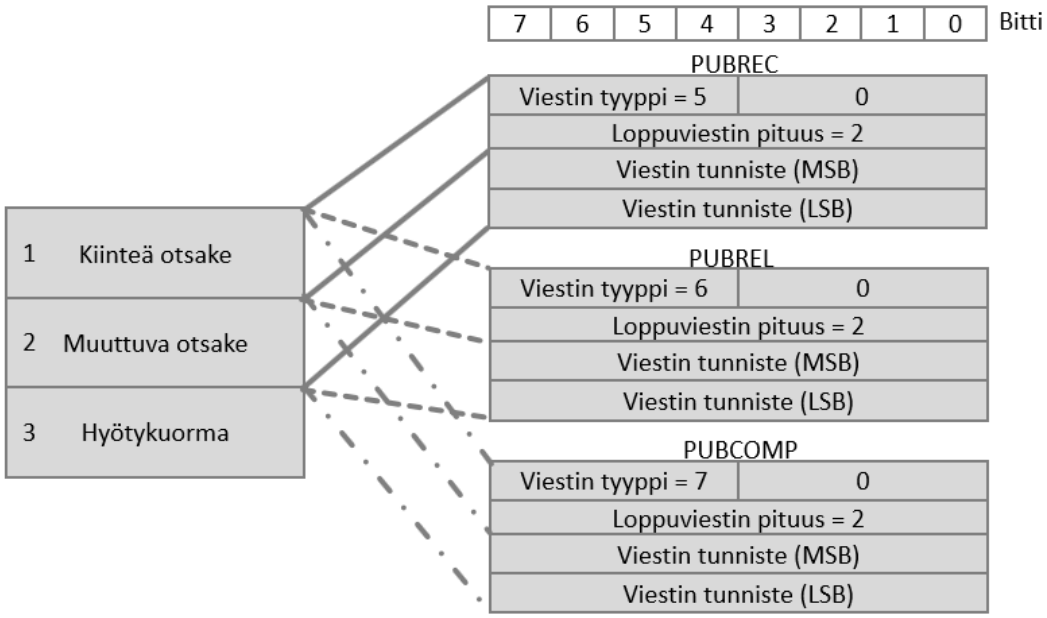
4.4.4 Laatutasolla 2 (QoS 2)

Laatutasolla 2 varmistetaan, että julkaisu lähetetään välittäjälle kerran (exactly once). Asiakkaan julkaisun saavuttua välittäjälle, välittäjä vastaa viestin vastaanotetuksi ja samalla se tallettaa viestin (PUBREC) (ks. kuviot 62 ja 63). Asiakaan saadessa viestin, se vapauttaa julkaistun aiheen varaaman muistin ja vastaa julkaisu vapautettu -viestillä (PUBREL). Välittäjän saatua viestin, aloittaa välittäjä julkaisun jakelemisen tilaajille ja samaan aikaan lähettää asiakkaalle julkaisu suoritettu viestin (PUBCOM). Näillä viesteillä varmistetaan, että välittäjä ja asiakas ovat kummatkin käsitelleet viestin. (MQTT Version 3.1.1 2014, 54-55.)

Julkaisuviestissä ero edellisiin laatutasoihin on, että laatutaso on asetettu arvoon 2. Talletus-, vapautus- ja suoritusviestit noudattavat laatutason 1 kuittausviestisyntaksia, vain viestin tyyppiä määrittelevien lippujen arvo on eri ja vastaa taulukossa 2 (s. 49) ja kuviossa 63 esitettyjä arvoja.



Kuvio 62. Laatutason 2 MQTT-julkaisun viestisekvenssi



Kuvio 63. MQTT laatutason 2 varmennusviestit MQTT Version 3.1.1 2014, 37-40)

4.4.5 Pysyvä viesti

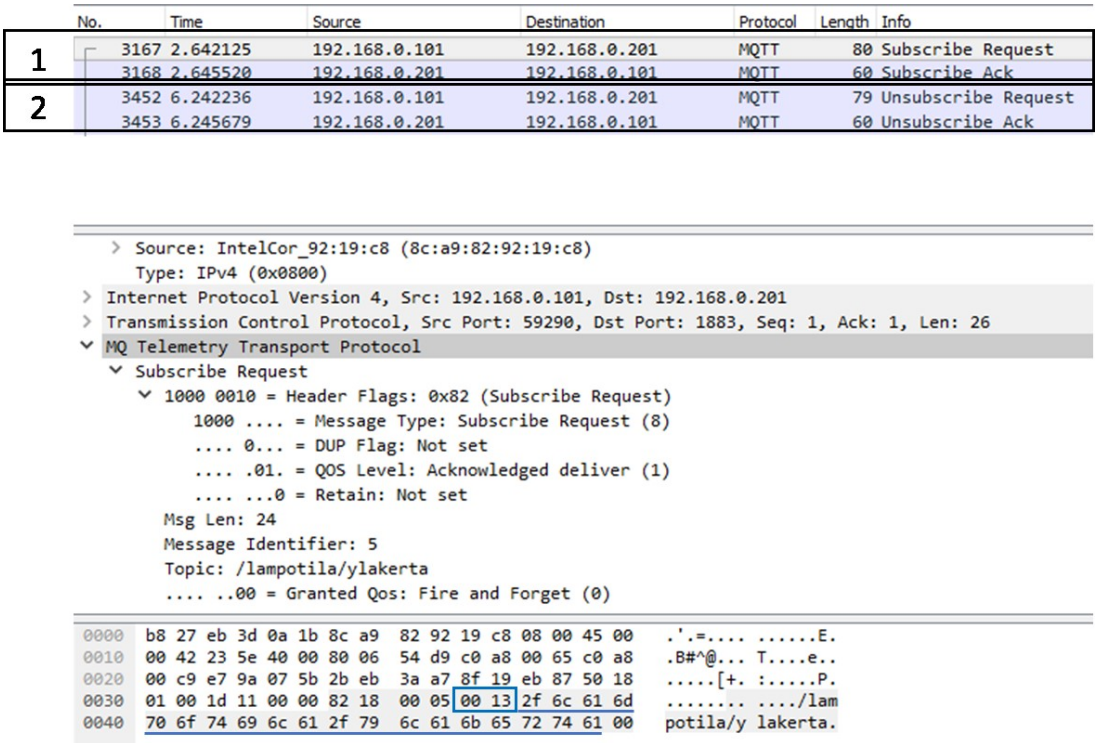
Julkaistaessa on mahdollista asettaa pysyvyys lippu aktiiviseksi (retain). Tällä lipulla välittäjä tallettaa aina aiheeseen lähetetyn viimeisimmän viestin ja sen laatutason. Talletettu viesti lähetetään aiheen tilauksille tai vanhoille yhteyksille yhteyden palautuessa. Välittäjän ei ole pakko tallettaa laatutasolla 0 lähetettyjä viestejä, mutta se voi halutessaan niin tehdä. (MQTT Version 3.1.1 2014, 34-35.)

4.5 Aiheen tilaaminen, tilauksen peruminen ja tiedon vastaanotto

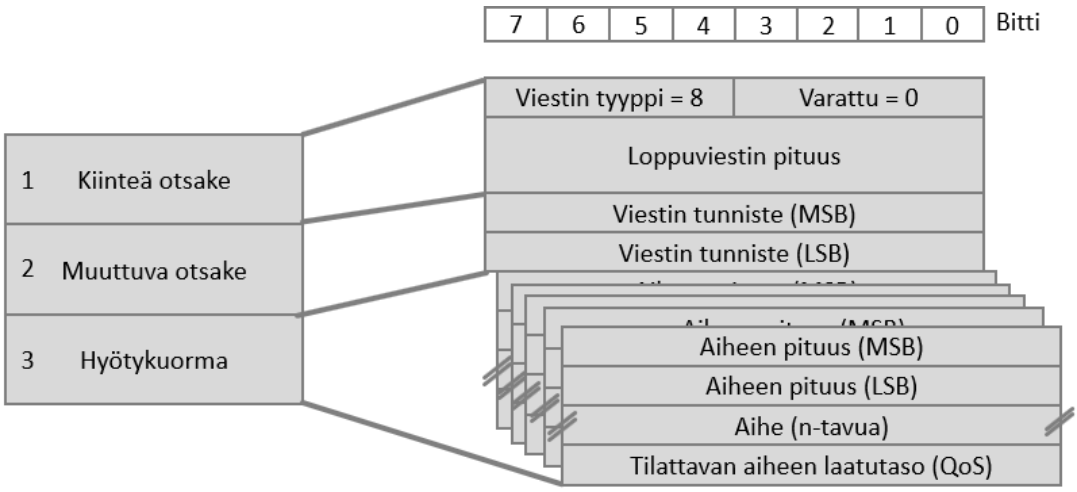
4.5.1 Aiheen tilaaminen ja peruminen

Aiheen tilaaminen välittäjältä ja tilauksen purkaminen ovat varsin suoraviivaisia operaatioita. Kuvio 64 esittää tilauksen (1) ja purkamisen (2). Viestit ovat identtisiä, eli viestissä on määriteltynä tilattava/peruttava aihe ja tilauksen liput. Viestin tyyppi määrittää, että onko kyseessä tilaus- vai purkuviesti. Kuittausviestit noudattavat samalla analogialla kuviossa 61 esitettyä tilauksen kuittausta (PUBACK). (MQTT Version 3.1.1 2014, 40-48.)

Tilausviesti mahdollistaa usean tilauksen tilaamisen yhdellä viestillä. Tällaisessa tapauksessa viestin hyötykuormassa on lista tilattavista aiheista. Listan elementti muodostuu tilattavan aiheen pituudesta, aiheesta ja aiheen laatutasosta (ks. kuvio 65). Välittäjä hyväksyy tai hylkää kaikki tilaukset yhdellä viestillä (SUBACK).



Kuvio 64. MQTT-aiheen tilaus ja peruminen (TCP/IP -rakenne)



Kuvio 65. MQTT -aiheen tilausviesti (SUBSCRIBE) (MQTT Version 3.1.1 2014, 40-43)

4.5.2 Tilattuun aiheeseen tulleen tiedon vastaanotto

Välittäjä julkaisee tilaajalle viestin noudattaen kappaleessa 4.4 esitettyä menetelmää. Erona kappaleessa esitettyyn menetelmään on roolien vaihtuminen, eli tilauksessa välittäjän ja asiakkaan roolit vaihtuvat.

Kuviossa 66 on sekvenssi aiheen tilauksesta (1), viestin välityksestä välittäjältä asiakkaalle laatutasolla 2 ja lopuksi tilauksen peruminen (3). Kuiviosta ilmenee selvästi, että viestin välitys välittäjältä asiakkaalle noudattaa julkaisun viestejä.

	Time	Asiakas		Välittäjä	Comment
1	10.331486	59290	Subscribe Request →	1883	MQTT: Subscribe Request
	10.339273	59290	← Subscribe Ack	1883	MQTT: Subscribe Ack
2	15.801715	59290	← Publish Message	1883	MQTT: Publish Message
	15.802180	59290	Publish Received →	1883	MQTT: Publish Received
	15.808502	59290	← Publish Release	1883	MQTT: Publish Release
	15.810574	59290	Publish Complete →	1883	MQTT: Publish Complete
3	22.930365	59290	Unsubscribe Request →	1883	MQTT: Unsubscribe Request
	22.936900	59290	← Unsubscribe Ack	1883	MQTT: Unsubscribe Ack

Kuvio 66. MQTT-viestin välitys asiakkaalle sekvenssi

5 MQTT-asiakasohjelmiston toteutus

5.1 Ohjelmistoarkkitehtuuri

5.1.1 Yleinen kuvaus

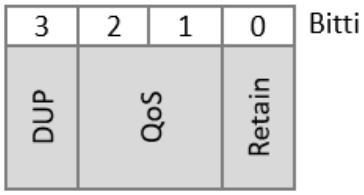
MQTT-protokollan toiminnan vuoksi asiakasohjelmiston täytyy pitää kirjaa yhteyden tilasta ja toisaalta sallia viestien saapuminen odottamattomasti. Käytännössä tulevia viestejä varten täytyisi olla oma säie, joka pystyisi vastaanottamaan ja prosessoimaan viestejä niiden saapuessa. Tämä säie voisi myös tarkoittaa keskeytyskontekstissa tapahtuvaa prosessointia. Toinen säie käsittelee yhteyden tilaa, varmistaa yhteyden ylläpidon (PING-viestit) ja hoitaa viestien lähetyksen. Näiden kahden säikeen välillä täytyy olla tietorakenne, jonka avulla ne vaihtavat tietoja. Säikeet ja muut käyttöjärjestelmien ominaisuudet jäävät kuitenkin tämän opinnäytetyön ulkopuolelle. Opinnäytetyössä käsitellään toteutuksen arkkitehtuuri ja toiminta, sekä esitellään API-funktiot.

Muistinvarauksen ongelma ratkaistiin jättämällä erilaisten puskureiden varaus ylemmän ohjelmistokerroksen hoidettavaksi. Laatutasolla 0 ei ole tarvetta pitää kirjaa viestien tilasta, joten viestien lähetys ja vastaanotto ovat käytännössä synkronisia operaatioita. Synkroninen toiminta vastaavasti tarkoittaa, ettei viestejä tarvitse tallettaa ja näin ollen muistinkäsittely jää yksinkertaiseksi.

Viestirakenteiden kasaaminen ja purkaminen toteutettiin C-kielen ”struct”-rakenteilla ja bittikentillä. Bittikenttien käytössä täytyy aina varmistaa, että kääntäjä käsittelee sekä bittejä, että bittikentälle määriteltyä muuttujaa oikein. Eräänä vaihtoehtona oli määritellä MQTT-viestit käyttäen Googlen Protocol Buffers -työkalua, mutta tästä työkalusta puuttui virallinen tuki C-kielelle. Työkalu olisi mahdollistanut protokollan määrittelyn abstraktilla tasolla usealle eri kielelle (Protocol Buffers n.d.).

5.1.2 Liput ja viestin tyyppi

Viestin tyypille on varattu kiinteästä otsakkeesta 4 ylintä bittiä. Viestin tyypin sallitut arvot on esitetty taulukossa 2 (sivu 49). Viestin liput on esitetty kuviossa 67. Lippujen käyttö on riippuvainen käytetystä komennosta.



Kuvio 67. Kiinteän otsakkeen liput (MQTT Version 3.1.1 2014, 33)

5.1.3 Viestin pituus

Viestin pituus tai oikeastaan kiinteän otsakkeen jälkeisten tavujen määrä muodostuu 1–4 tavusta, kuten kappaleessa 4.2.1 (s. 47) aiemmin todettiin. Tarkemmin tarkasteltuna pituuskentässä olevan tavun ylin bitti asetetaan ykköseksi, silloin kun tarvitaan lisätilaa seuraavasta tavusta. Merkitsevin tavu on ensimmäisenä ja vähiten merkitsevä viimeisenä tavuna.

Alla C-kielellä esitetty esimerkkitoteutus muodostaa MQTT-protokollan mukaisen pituuskentän (set_size). Vastaavasti seuraava funktio purkaa MQTT-viestistä koneellisesti käsiteltävän pituusarvon (get_size) (MQTT Version 3.1.1 2014, 19).

```
uint8_t set_size(uint8_t * output, size_t msgSize)
{
    uint8_t returnValue = 0;
    do {
        uint8_t encodedByte = msgSize % 128;
        msgSize = msgSize / 128;
        if (0 < msgSize)
            encodedByte = encodedByte | 128;
        output [returnValue] = encodedByte;
        returnValue++;
    } while (0 < msgSize);
    return returnValue;
}
```

```
uint32_t get_size(uint8_t * input)
{
    uint32_t multiplier = 1;
    uint32_t value      = 0;
    uint8_t  cnt        = 1;
    uint8_t  aByte      = 0;
    do {
        aByte  = input[cnt++];
        value += (aByte & 127) * multiplier;
        if (multiplier > (128*128*128))
            return -1;
        multiplier *= 128;
    } while (0 != (aByte & 128));
    return value;
}
```

5.1.4 Ensimmäisen tason ohjelmistorajapinta

Ensimmäisen tason ohjelmistorajapintana toimii yksi funktio, jolle annetaan parametrina tehtävä ja toisena parametrina tehtävään liittyvät tiedot. Funktion määrittely on seuraavanlainen:

```
MQTTErrorCodes_t mqtt(MQTTAction_t a_action,  MQTT_action_data_t * a_action_ptr)
```

Tehtävät on määritelty seuraavanlaisesti:

```
typedef enum MQTTAction
{
    ACTION_DISCONNECT,
    ACTION_CONNECT,
    ACTION_PUBLISH,
    ACTION_SUBSCRIBE,
    ACTION_KEEPALIVE,
    ACTION_INIT,
    ACTION_PARSE_INPUT_STREAM
} MQTTAction_t;
```

Tehtävään liittyvät parametrit riippuvat tehtävästä ja jokaiselle tehtävälle on määritelty oma tietorakenteensa.

```
typedef struct MQTT_action_data
{
    union {
        MQTT_shared_data_t  * shared_ptr;
        MQTT_connect_t      * connect_ptr;
        uint32_t              epalsed_time_in_ms;
        MQTT_input_stream_t * input_stream_ptr;
        MQTT_publish_t      * publish_ptr;
        MQTT_subscribe_t    * subscribe_ptr;
    } action_argument;
} MQTT_action_data_t;
```

5.1.5 Optimoitu ohjelmistorajapinta

Optimoitu ohjelmistorajapinta piilottaa ensimmäisen tason ohjelmistorajapinnan (kappale 5.1.4) tietorakenteet ja tarjoaa käyttäjälle yksiselitteiset funktiokutsut.

```
bool mqtt_connect(char                    * a_client_name_ptr,
                  uint16_t                  a_keepalive_timeout,
                  uint8_t                 * a_username_str_ptr,
                  uint8_t                 * a_password_str_ptr,
                  uint8_t                 * a_last_will_topic_str_ptr,
                  uint8_t                 * a_last_will_str_ptr,
                  MQTT_shared_data_t      * mqtt_shared_data_ptr,
                  uint8_t                 * a_output_buffer_ptr,
                  size_t                    a_output_buffer_size,
                  bool                      a_clean_session,
                  data_stream_out_fptr_t    a_out_write_fptr,
                  connected_fptr_t          a_connected_fptr,
                  subscrbe_fptr_t           a_subscribe_fptr,
                  uint8_t                   a_timeout_in_sec);

bool mqtt_disconnect();

bool mqtt_publish(char * a_topic_ptr,
                  size_t a_topic_size,
                  char * a_msg_ptr,
                  size_t a_msg_size);

bool mqtt_publish_buf(char    * a_topic_ptr,
                      size_t    a_topic_size,
                      char    * a_msg_ptr,
                      size_t    a_msg_size,
                      uint8_t * a_output_buffer_ptr,
                      uint32_t  a_output_buffer_size);

bool mqtt_subscribe(char    * a_topic,
                    uint8_t   a_timeout_in_sec);

bool mqtt_keepalive(uint32_t a_duration_in_ms);

bool mqtt_receive(uint8_t * a_data, size_t a_amount);

int (*data_stream_in_fptr_t)(uint8_t * a_data_ptr, size_t a_amount);

int (*data_stream_out_fptr_t)(uint8_t * a_data_ptr, size_t a_amount);

void (*connected_fptr_t)(MQTTErrorCodes_t a_status);

void (*subscrbe_fptr_t)(MQTTErrorCodes_t    a_status,
                        uint8_t           * a_data_ptr,
                        uint32_t            a_data_len,
                        uint8_t           * a_topic_ptr,
                        uint16_t            a_topic_len);
```

5.2 Versionhallinta

Opinnäytetyössä käytettiin versionhallintana suosittua Git-versionhallintaa, mikä kehitettiin alun perin Linux-käyttöjärjestelmän ytimen kehittämisen tarpeisiin. Git-versionhallinta perustuu muutosten huomaamiseen ja niiden liittämiseen. Tämä mahdollistaa lähdekoodin tai yleisesti tekstipohjaisen dokumentin yhtäaikaisen muokkaamisen. Git-versionhallinta mahdollistaisi myös toteutuksen haaroittamisen. Suurissa järjestelmissä haaroittaminen helpottaa esimerkiksi koodin jakamisen koodin valmiusasteen mukaan, jolloin yksi haara voisi olla tuotannossa oleva ja toinen tuotekehityksessä oleva haara (Git, 2016). Opinnäytetyössä ei varsinaisesti ollut tarvetta haaroittamiselle, eikä usean käyttäjän yhtäaikaiselle lähdekoodin muokkaamiselle.

5.3 Tekijänoikeus ja lisensointi

Opinnäytetyössä toteutettu ohjelmisto on avoimen lähdekoodin periaatteen mukaan vapaasti käytettävissä ja muokattavissa. Tuotetusta ohjelmistosta ei haluta rahallista korvausta, mutta ohjelmistosta ei myöskään kanneta vastuuta. Tähän tarkoitukseen MIT-lisenssi sopii erinomaisesti, mikä antaa täydet vapaudet muokata koodia ja samalla siirtää vastuun ohjelmiston käyttäjälle (The MIT License, n.d.).

5.4 Lähdekoodin dokumentointi

Lähdekoodin dokumentointi noudattaa Doxygenin dokumentointisääntöjä (Doxygen, 2016). Doxygen-dokumentaation muodostamiseen käytettiin automatisoitua CodeDocs-järjestelmää. CodeDocs muodostaa dokumentit automaattisesti, kun koodit ovat päivittyneet GitHub-palveluun (CodeDocs, 2016).

5.5 Käännösympäristö

Ohjelmiston kääntämisen make-tiedostojen muodostamiseen käytettiin ylemmän tason CMake-määrittelyä (versio 3.2.2.). CMake mahdollistaa käännössääntöjen kuvaamisen abstraktilla tasolla käyttöjärjestelmästä riippumatta. Käännös tapahtui Bash-komentokehotteessa, joka suoritettiin Windows 10- ja Linux -käyttöjärjestelmissä. C-kääntäjänä toimi GNU 6.3.0 cc-kääntäjä.

FreeRTOS-toteutusesimerkki teki käännösympäristöön poikkeuksen, sillä se käännetään Visual Studio -kääntäjällä. Alkuperäinen FreeRTOS esimerkki Windows-ympäristössä oli toteutettu Visual Studio -kääntäjällä, joten oli luontevaa käyttää samaa ympäristöä esimerkissä.

5.6 Testaus

5.6.1 TDD ja yksikkötestaus

Toteutus suoritettiin testiohjattuna toteutuksena (TDD), missä toteutettavalle koodille määritellään ja toteutetaan testit ennen varsinaisen toiminnallisuuden toteutusta. Menetelmä varmistaa sen, että toteutettu ohjelmisto on julkaisukelpoista heti ensimmäisestä toteutuksesta lähtien (Galezowski 2017, 5-7).

Yksikkötestauksessa hyödynnettiin Unity-yksikkötestausohjelmistoa. Unity on yksinkertainen ohjelmisto, jolla voi testata erilaisten muuttujien ja muistialueiden arvoja (Unity n.d.). Unity on yhdistetty CMake-käännöstyökalussa mukana tulevaan CTest-testaustyökaluun. CTest mahdollistaa useiden Unity-testitapausten suorittamisen yhdellä kerralla (CMake/Testing With CTest 2016).

5.6.2 Staattinen tarkistus

Staattisella tarkistuksella minimoidaan yleisimmät virheet ja vaaranpaikat. Analyysillä tarkistetaan myös koodaustyyliä yleisellä tasolla. Asiakasohjelmiston staattinen tarkistus varmistettiin avoimen lähdekoodin Cppcheck-työkalulla. Työkalu tarkistaa yleisimpiä virheitä, mutta ei ota kantaa tyyliin (Cppcheck n.d.). Cppcheck oli integroituna VisualStudio Code -kehitystyökaluun ja kehitystyökalu ilmoitti välittömästi mahdolliset ongelmakohdat koodissa. Kääntäjän -Wall -parametri tarkistaa myös varsin kattavasti virheitä. MQTT-protokollapino tarkistettiin myös Visual Studion -kääntäjän staattisella tarkistuksella. Koodaus tyyliin ei ollut staattista tarkistusta käytössä.

5.6.3 Järjestelmätestaus

Järjestelmätestaus toteutettiin manuaalisena testauksena. Testauksessa käytettiin Mosquitto-välittäjää, johon asiakasohjelmistot olivat yhteydessä. Asiakasohjelmistoista voitiin käyttää hyödyksi olemassa olevia ohjelmistoja kuten mosquitto_pub,

mosquitto_sub, sekä MQTT FX. Tietovirran oikeellisuuden tarkistamiseen käytössä oli Wireshark-ohjelmisto.

5.7 Sprintit / Osakokonaisuudet

5.7.1 Yleistä

Kappaleessa 5.7 käydään läpi, missä järjestyksessä ja millaisissa osakokonaisuuksissa ohjelmiston toteutus suoritettiin. Osakokonaisuudet voisivat olla Scrum-tyyppisen projektinhallinnan sprinttejä (ks. Scrum 2017).

5.7.2 Ohjelmistoarkkitehtuurin pääpiirteet, käännös- ja testausympäristö

Ensimmäisen sprintin sisältönä oli käännös- ja testausympäristö. Käännös- ja testausympäristöt kuvataan tarkemmin kappaleissa 5.5 ja 5.1.4.

5.7.3 Kiinteä otsake

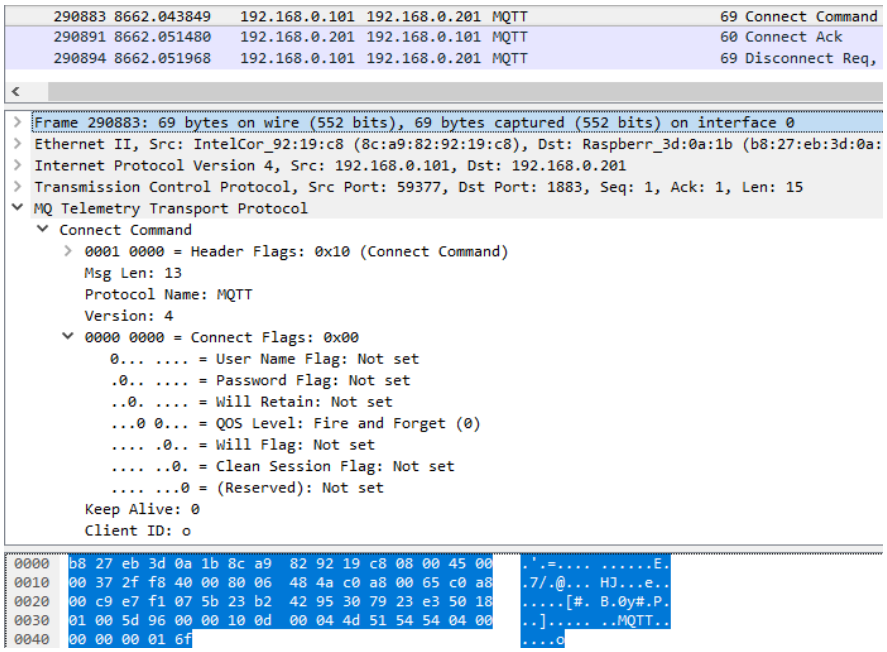
Toteutettiin kiinteä otsake sisältäen liput, viestin tyypin ja loppuviestin pituuden. Liput ja viestin tyypit määritellään bittikenttien avulla, jotka on kuvailtu tarkemmin kappaleessa 5.1.2. Loppuviestin pituuden vaatima logiikka on kuvattu kappaleessa 5.1.3.

5.7.4 Yhteydenmuodostus välittäjään

Opinnäytetyötä varten tehdyssä toteutuksessa ensimmäinen varsinainen toimiva tuotos oli toisen sprintin sisältö. Toteutuksessa haettiin pienintä mahdollista toimivaa yksikköä, jota voisi testata Mosquitto-välittäjän kanssa. Toteutuksessa yhteydenmuodostus tehtiin QoS0-tasolla. Yhteyden muodostusviestissä olevat ylläpitoajastin (keep alive), viimeinen tahto (last will/testament), käyttäjätunnus ja salasana jätettiin myöhäisempiin toteutuksiin (sprintteihin). Yhteyden muodostuksen liput edellä lueteltujen ominaisuuksien mukaan määrittävät otsakkeen lippukentän arvoksi 0. MQTT-viesteistä toiminnallisuuteen tarvittiin kuviossa 48 (s. 53) esitetty yhteyden muodostus yhteyden ylläpitoajastimen arvolla 0 ja kuviossa 49 (s. 54) esitetty muodostuksen vastausviesti.

Testauksessa välittäjänä toimi Mosquitto-ohjelmisto, jota ajettiin Linux-käyttöjärjestelmässä. Asiakasohjelmisto ajettiin toiselta koneelta ja näiden kahden koneen välinen viestiliikenne talletettiin Wireshark-ohjelmistolla. Wireshark-ohjelma osaa käsitellä MQTT-viestejä suoraan, mikä teki testaamisesta helppoa. Yhteydenoton viestiliikenne on esitelty kuviossa 68. Mosquitto-välittäjä käynnistettiin seuraavalla komennolla käyttäen oletusalustusarvoja:

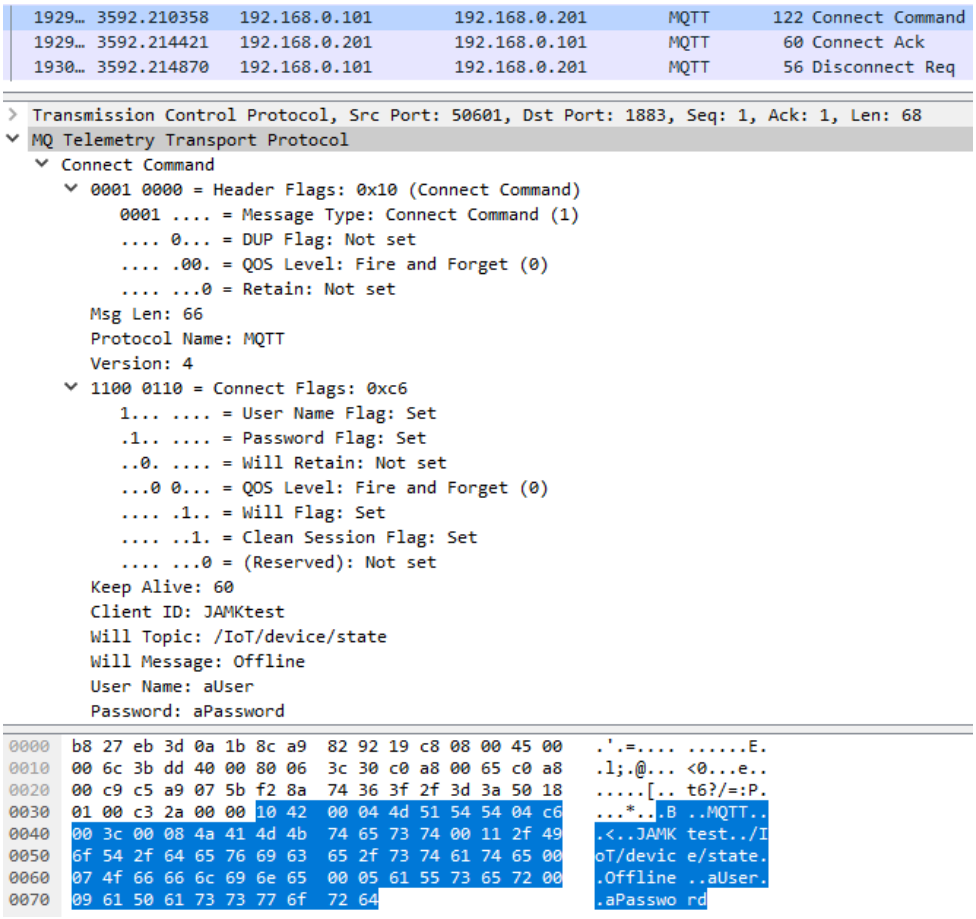
```
sudo mosquitto -v -c /etc/mosquitto/mosquitto.conf
```



Kuvio 68. Yhteyden muodostuksen parametrit

5.7.5 Yhteydenottoviestin parametrien lisäys (QoS0)

Kappaleesta 5.7.4 pois jätetyt parametrit lisättiin yhteydenottoviestiin muuttamatta laatutasoa. Laatutaso jätettiin 0-tasolle, sillä sen testaaminen vaatisi logiikkaa, jota opinnäytetyössä toteutettu ohjelmisto ei tule tukemaan.



Kuvio 69. Yhteyden muodostuksen parametrit

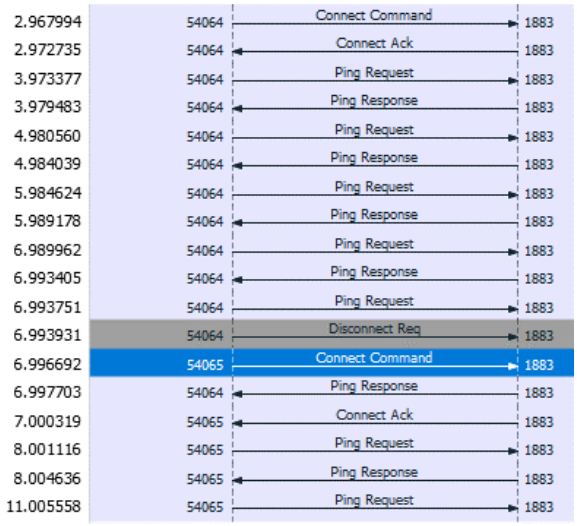
```
New connection from 192.168.0.101 on port 1883.
New client connected from 192.168.0.101 as JAMKtest (c1, k60, uaUser).
Sending CONNACK to JAMKtest (0)
Received DISCONNECT from JAMKtest
```

Kuvio 70. Välittäjän tulosteet yhteydenmuodostamisesta ja -purkamisesta

5.7.6 Ylläpitoajastin

Ylläpitoajastin vaatii PINGREQ- (ks. kuvio 44) ja PINGRESP (ks. kuvio 46) -viestien käsittelyyn logiikan. Asiakasohjelmiston täytyy lähettää PINGREQ-viesti välittäjälle yhteyden muodostuksessa määräämänsä keep alive -parametrin mukaisesti. Viestiä ei tarvitse lähettää, jos asiakkaan ja välittäjän välillä on muuta liikennettä. Yksinkertaisessa toteutuksessa muuta liikennettä välittäjän ja asiakasohjelmiston välillä ei ole.

Kuviosta 71 ilmenee välittäjän ja asiakasohjelmiston välinen MQTT-viestintä ja kuviossa 72 esitetään välittäjän tulosteet samasta käyttötapauksesta. Tulosteet ovat toteutetun MQTT-protokollapinon ja Mosquitto-välittäjän välisestä liikenteestä. Yhteyden ylläpidolle toteutettiin kuvion 71 mukaisesti kaksi erillistä testitapausta, joista ensimmäinen pitää yhteyden onnistuneesti ja lopetus tapahtuu protokollan mukaisesti. Jälkimmäinen yhteydenotto päättyy välittäjän toimesta, sillä PINGREQ-viestiä ei ole lähetetty riittävän ajoissa. Jälkimmäisessä tapauksessa asiakasohjelmisto pystyy lähettämään vaaditun PINGREQ-viestin, mutta välittäjä ei enää käsittele asiakkaalta tulevia viestejä.



Kuvio 71. Yhteyden ylläpito ylläpitoajastimen mukaisesti

```
1497453883: New client connected from 192.168.0.101 as JAMKtest (c1, k2).
1497453883: Sending CONNACK to JAMKtest (0)
1497453884: Received PINGREQ from JAMKtest
1497453884: Sending PINGRESP to JAMKtest
1497453885: Received PINGREQ from JAMKtest
1497453885: Sending PINGRESP to JAMKtest
1497453886: Received PINGREQ from JAMKtest
1497453886: Sending PINGRESP to JAMKtest
1497453887: Received PINGREQ from JAMKtest
1497453887: Sending PINGRESP to JAMKtest
1497453887: Received PINGREQ from JAMKtest
1497453887: Sending PINGRESP to JAMKtest
1497453887: Received DISCONNECT from JAMKtest
1497453887: New connection from 192.168.0.101 on port 1883.
1497453887: New client connected from 192.168.0.101 as JAMKtest (c1, k2).
1497453887: Sending CONNACK to JAMKtest (0)
1497453888: Received PINGREQ from JAMKtest
1497453888: Sending PINGRESP to JAMKtest
1497453890: Client JAMKtest has exceeded timeout, disconnecting.
```

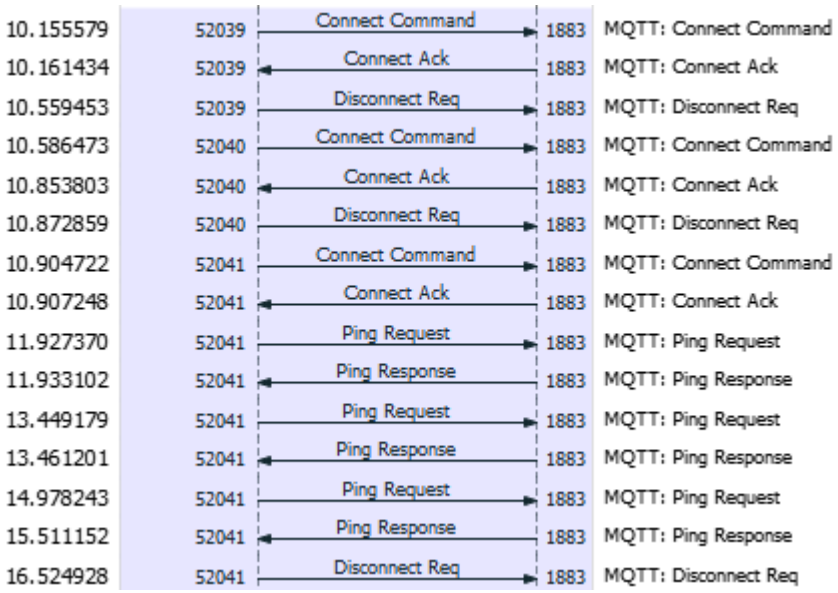
Kuvio 72. Välittäjän tulosteet yhteyden ylläpidosta

5.7.7 Linux-riippuvuuksien purkaminen

Toteutuksessa muutettiin suorat Socket-kutsut epäsuoriksi hyväksikäyttäen funktiopointtereita sekä tiedon lähettämiseen, että vastaanottamiseen. Menetelmällä varmistettiin yhteensopivuus eri järjestelmiin. Toteutuksesta löytyy valmiina adaptaatiokerros Linux-ympäristölle.

5.7.8 Yhteyden muodostus ja ylläpito

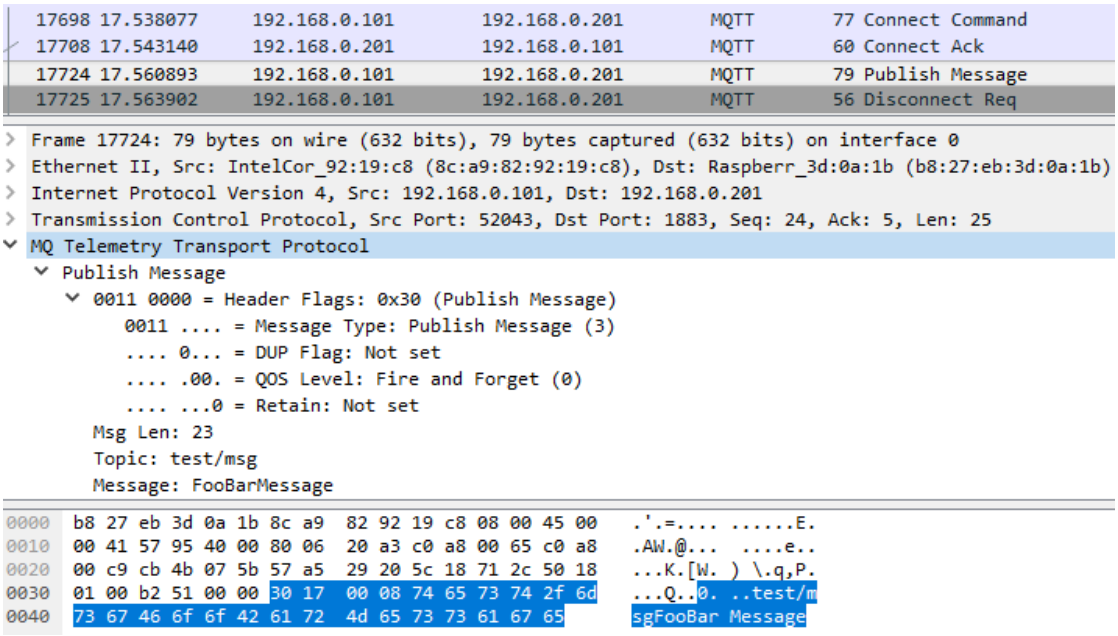
Toteutettiin kappaleessa 5.1 esitetty yhteyden muodostaminen, ylläpito ja katkaisu. Toteutuksen testitulokset ovat esillä kuviossa 73.



Kuvio 73. Yhteyden muodostus ja ylläpito

5.7.9 Viestin julkaisu

Viestin julkaisu (publish) laatutasolla 0 on varsin suoraviivainen operaatio, sillä välittäjä ei vastaa julkaisuun. Viesti siis lähetetään ja jatketaan toimintaa. Tarkempi kuvaus viestin julkaisemisesta on esitettynä kappaleessa 4.4 (s. 61). Kuvioissa 74 ja 75 on esitettynä toteutuksen oikean toiminnan toteaminen Wireshark- ja mosquitto_sub-ohjelmistoilla.



Kuvio 74. Yhteydenotto, viestin julkaisu ja yhteyden lopetus

```
rojala@KotiKone:~$ mosquitto_sub -h 192.168.0.201 -t test/msg -d
Received CONNACK
Received SUBACK
Subscribed (mid: 1): 0
Received PUBLISH (d0, q0, r0, m0, 'test/msg', ... (13 bytes))
FooBarMessage
```

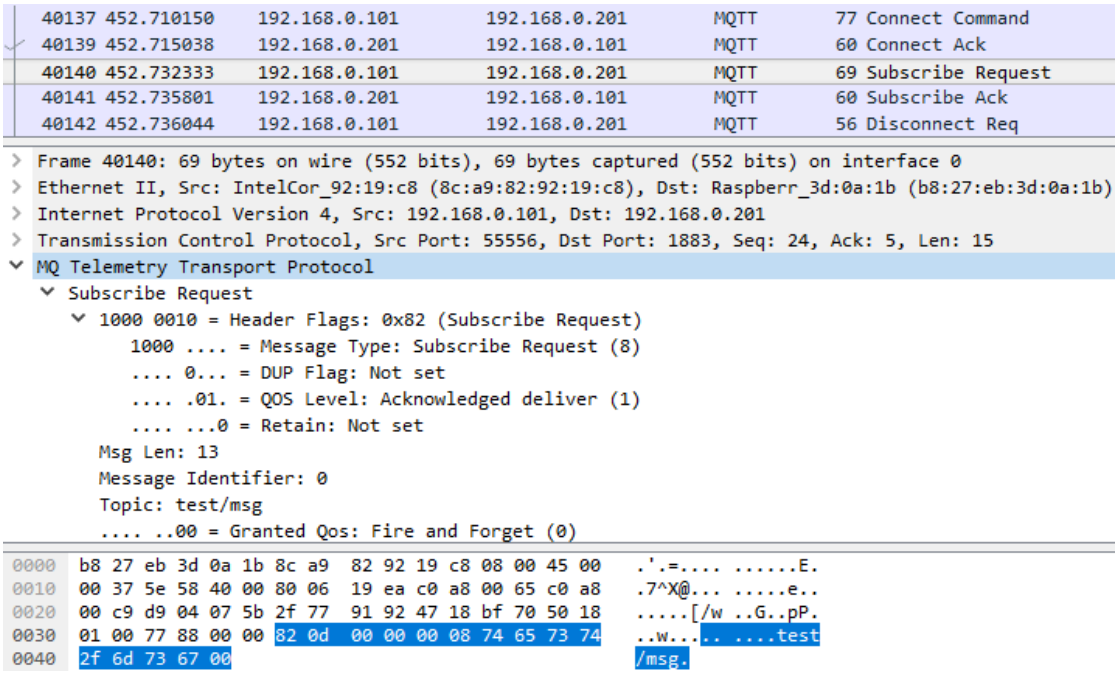
Kuvio 75. Julkaisun varmistaminen mosquiotto_sub-ohjelmistolla

5.7.10 Viestin tilaus

Viestin tilauksessa (subscribe) hyötykuorma pitää sisällään listan tilattavista aiheista. Jokaiselle tilattavalle aiheelle on 2 tavua pitkä aiheen pituuskenttä, jota seuraa itse aihe. Tietorakenteen lopussa on toive tilattavan viestin laatutasolle, sillä laatutason päättää välittäjä. Usean aiheen tilaus voidaan hoitaa lähettämällä useita tilauksia, joten tässä toteutuksessa yhdellä tilauksella voi tilata vain yhden aiheen. Välittäjä varmistaa aiheen saapumisen lähettämällä kuittauksen. Tarkempi kuvaus viestin tilaamisesta on esitetty kappaleessa 4.5 (s. 66).

Erityishuomiota täytyy kiinnittää kiinteän otsakkeen arvoihin. MQTT-määrittelyssä on mainittu, että neljästä alimmasta bitistä bitti numero 1 täytyy olla yksi ja loput arvossa nolla. Tämä tarkoittaa tilausviestissä, että laatutaso on aina 1 (MQTT Version 3.1.1 2014, 29) ja tästä syystä viestin onnistuneeseen tilaukseen tulee aina SUBACK kuittausviesti. Käytännön testeissä ilmeni, että viestin tilaus toimii myös laatutason

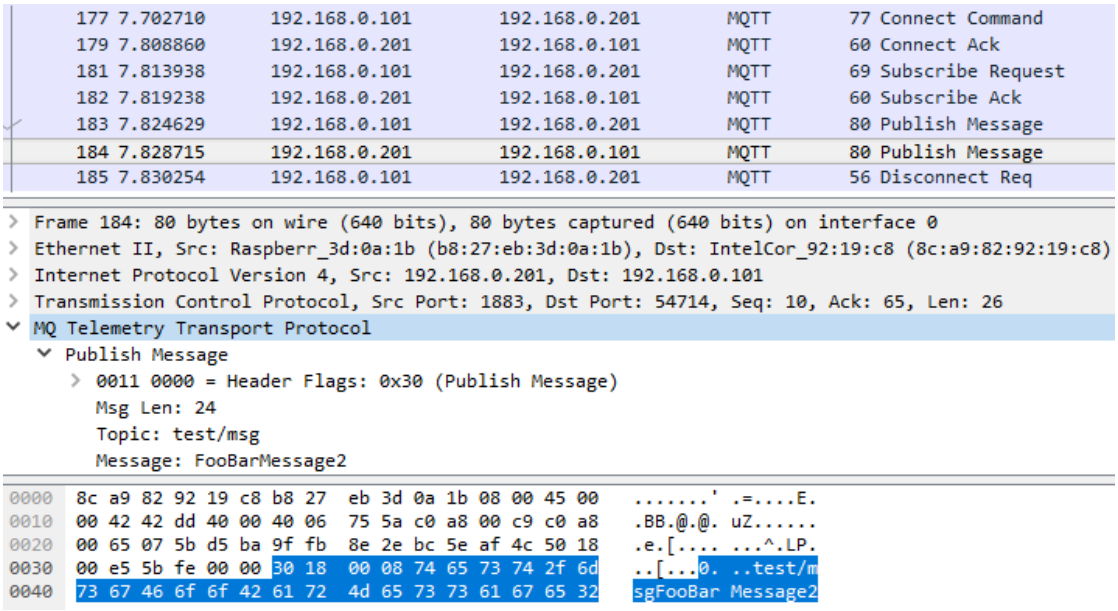
ollessa 0, eikä välittäjä tällöin lähetä SUBACK-viestiä. Kuvio 76 esittää viestin tilaamisesta otetun Wireshark-lokin, jolla varmistettiin toteutuksen oikeellisuus.



Kuvio 76. Viestin tilauksen viestirakenteen varmistaminen

5.7.11 Tilatun viestin vastaanotto

Asiakasohjelmisto pyytää välittäjää julkaisemaan viestit halutulta aiheelta asiakkaalle. Julkaistu viesti välitetään siis PUBLISH-viestillä, samaan tapaan kuin viesti lähetetään välittäjälle, mutta nyt suunta on vastakkainen. Toteutuksen toiminnan varmistavasta kuviossa 77 ilmenee hyvin, kuinka asiakas (192.168.0.101) lähettää viestin välittäjälle ja välittäjä (192.168.0.201) lähettää sen takaisin asiakkaalle. Toteutettavan osuuden viestirakenteet on esitelty kappaleessa 4.4 (s. 61).



Kuvio 77. Viestin tilauksen ja vastaanoton toiminnan varmistaminen

5.7.12 MVP (Minimum Viable Product)

Pienin mahdollinen toimiva toteutus voisi olla yhteydenotto välittäjään ilman yhteyden ylläpitoajastinta. Sellaisella toteutuksella ei tosin voisi tehdä juurikaan mitään. Viimeistä tahtoa voisi käyttää yksinkertaisissa laitteissa tai jopa painonapeissa binaarisen tiedon välittämisessä, mutta tällöin toiminnolla olisi melko pitkä viive (yhteyden ylläpitoajastimen aika kertaa 1.5).

Pienimmässä mahdollisessa tuotteessa toteutettiin yhteydenotto ja ylläpito, viestien tilaus, tilausten vastaanotto ja viestien julkaisu sekä yhteyden katkaiseminen. Kaikki toiminnot toteutettiin laatutasolla 0, paitsi viestin tilaus, jolle oli MQTT-määritelmässä määrätty laatutaso 1.

MVP-ohjelmistossa on yksi rajapintafunktio, jota käytetään viestien lähettämiseen ja vastaanottamiseen. Funktion ensimmäinen parametri määrittää operaation ja toinen parametri sisältää operaation liittyvän tietorakenteen. Rajapintafunktio on kuvattu tarkemmin kappaleessa 5.1.4.

Kuviossa 78 on esitetty MVP:n testitulos, jossa yhteys pidetään päällä 60 sekuntia ja 10 sekunnin välein lähetetään ja vastaanotetaan lähetetty viesti (Publish Message).

Viestien lähetyksen ja vastaanoton välillä pidetään yhteyttä yllä (Ping Request), niin ettei 4 sekunnin yhteyden ylläpitoon asetettu aika ylity.

241.597261	192.168.0.101	192.168.0.201	MQTT	80 Connect Command
241.602569	192.168.0.201	192.168.0.101	MQTT	60 Connect Ack
241.699267	192.168.0.101	192.168.0.201	MQTT	70 Subscribe Request
241.706197	192.168.0.201	192.168.0.101	MQTT	60 Subscribe Ack
241.706693	192.168.0.101	192.168.0.201	MQTT	78 Publish Message
241.710850	192.168.0.201	192.168.0.101	MQTT	78 Publish Message
245.710634	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
245.713974	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
249.721148	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
249.797873	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
251.729399	192.168.0.101	192.168.0.201	MQTT	78 Publish Message
251.736120	192.168.0.201	192.168.0.101	MQTT	78 Publish Message
255.741096	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
255.744808	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
259.754384	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
259.757926	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
261.762461	192.168.0.101	192.168.0.201	MQTT	78 Publish Message
261.766142	192.168.0.201	192.168.0.101	MQTT	78 Publish Message
265.771260	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
265.774623	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
269.784243	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
269.787368	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
271.792995	192.168.0.101	192.168.0.201	MQTT	78 Publish Message
271.796850	192.168.0.201	192.168.0.101	MQTT	78 Publish Message
275.802287	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
275.806376	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
279.812987	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
279.818146	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
281.821994	192.168.0.101	192.168.0.201	MQTT	78 Publish Message
281.825650	192.168.0.201	192.168.0.101	MQTT	78 Publish Message
285.832918	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
286.159581	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
289.844660	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
289.848962	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
291.853240	192.168.0.101	192.168.0.201	MQTT	78 Publish Message
291.856750	192.168.0.201	192.168.0.101	MQTT	78 Publish Message
295.865651	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
295.869341	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
299.879369	192.168.0.101	192.168.0.201	MQTT	56 Ping Request
299.883085	192.168.0.201	192.168.0.101	MQTT	60 Ping Response
301.887844	192.168.0.101	192.168.0.201	MQTT	56 Disconnect Req

Kuvio 78. MVP-toteutuksen testitulos

5.7.13 Tuotteistettu versio

MVP-ohjelmiston ohjelmistorajapinta mahdollisti toiminnan toteamisen, mutta se ei ollut helppokäyttöinen. MVP-ohjelmiston testitapauksia luodessa, ilmeni ohjelmisto-rajapinnan puutteita, jotka korjattiin testikoodissa. Testikoodista pystyikin helposti muokkaamaan tuotteistettuun versioon soveltuvat ohjelmistorajapinnat.

Ohjelmistorajapintojen lisäksi täytyi toteuttaa varsinainen tuote ja tässä tapauksessa yksinkertaisinta oli toteuttaa komentorivipohjainen ohjelmisto Linux-ympäristöön. Ohjelmisto noudattaa Mosquitto-ohjelmiston sub- ja pub-testiohjelmistojen toimintatapaa, missä komentoriville annetaan välittäjä, aihe, sekä muut tarpeelliset parametrit. Komentorivityökalun tavoitteena oli hioa ohjelmistorajapinta helpoksi, tarkastella muistinkäyttöä sekä mahdollistaa testaaminen isoilla tietomäärillä. Soket-toiminnallisuuden lisäksi ohjelmistossa käytettiin käyttöjärjestelmän sleep-toiminnallisuutta viiveiden aikaansaamiseksi, sekä pthread-kirjastoa säikeiden toteutukseen. Käytetyt käyttöjärjestelmän palvelut on kaikki koottu h-tiedostoon omaksi määrittelyksi, mikä mahdollistaa helpon siirron toiseen käyttöjärjestelmään/ympäristöön.

MVP-toteutuksesta löytyikin muutamia toiminnallisia ongelmia isojen tietomäärien lähetyksessä. Paketin koon laskennassa oli virhe, joka ilmeni, kun kiinteänotsakkeen pituutta täytyi kasvattaa. Toinen isompi ongelma liittyi testikoodin Soket-ominaisuuden toteutukseen, sillä Soket-yhteys välittäjään täytyy olla tavuvirta-tyyppinen (byte stream). Tavuvirta-tyypistä seurasi se, että vastaanotettavien tavujen määrä vaihteli yhdestä kilotavusta neljään kilotavuun. Vastaanottoon täytyi toteuttaa viestin pituuden laskenta perustuen kiinteän otsakkeen tietoihin.

Toteutettu ohjelmistorajapinta lisäsi käytännössä abstraktiokerroksen MVP-rajapinnan yläpuolelle. Ylimääräinen kerros mahdollisti sen, ettei loppukäyttäjän tarvitse välittää ohjelmiston tietorakenteista. Ohjelmistorajapinnan funktioiden nimi määrittää yksiselitteisesti funktion toiminnan. Ohjelmistorajapinta on esitelty kappaleessa 5.1.5 (sivulla 71).

Komentorivityökalun käyttöesimerkki viestin julkaisusta ja vastaanotosta.

```
./rmc -b 192.168.0.201 -s 1883 -k 10 -t iot/sensor/a -m [20.0]

./rmc -b 192.168.0.201 -s 1883 -k 10 -t iot/sensor/+
```

Komentorivityökalun erikoisuutena on mahdollisuus siirtää tiedostoja. Tämä helpotti huomattavasti isojen tietomäärien testausta.

```
./rmc -b 192.168.0.201 -s 1883 -k 10 -t iot/sensor/a -f file_out.snt

./rmc -b 192.168.0.201 -s 1883 -t iot/sensor/+ -r -f file_in.rcv
```

Taulukossa 4 on esitetty komentorivityökalun muistinkulutus käännettynä x86 kohteeseen. Komentorivityökalu on käännetty kahdella erilaisella parametrimäärityksellä. Debug-versiossa kääntäjän optimointitaso on -O0, käytössä on tulosteet ja kääntäjän virheenhavainnointiominaisuudet. Release-versiossa virheiden havainnointi ei ole käytössä ja optimointitaso on -Os. Muistinkulutustiedot ovat peräisin linkittäjän map-tiedostosta (Special sections in Linux binaries n.d.).

Taulukko 4. Komentorivityökalun muistinkulutus (x86)

Muistialue\Tyyppi	Debug	Release
koodi (.text)	8.9kB	4.0kB
Data (.rodata/.bss)	1.1kB / 8B	22B / 8B

Komentorivi-työkalulla isojen tietomäärien testaus toteutettiin tiedostojen siirrolla. Esimerkiksi valokuvan tai dokumentin siirtäminen testasi ison koon lisäksi myös tiedonsiirron oikeellisuuden.

5.7.14 Testauksen laajentaminen

Yhden alaviivan puute CMake-tiedostossa oli aiheuttanut sen, etteivät kaikki kääntäjälle asetetut parametrit olleetkaan käytössä. Korjauksen jälkeen löytyi muutamia kohtia toteutuksesta, joita oli syytä muuttaa.

Testaaminen RaspberryPi-laitteistolla toi esiin muutaman ongelman testikoodeista. Ensimmäinen liittyi liian pitkän MQTT-viestin testaukseen, missä arvo ylitti 32-bittisen arvon rajan ja tästä syystä käyttäytyminen ei ollutkaan sama kuin 64-bittisellä PC:llä testatessa. Toinen ongelma oli haastavampi, sillä ongelma tapahtui vain joskus. Syyksi selvisi alustamaton parametri (bitti) yhteydenotto-viestissä. Tämän virheen avulla selvisi myös se, ettei Mosquitto-välittäjä palauta virhearvoa asiakkaalle vaan sulkee kylmästi asiakkaan Soket-yhteyden välittäjän päästä. Bitti-virheen havaitsemisessa auttoi testitapausten pilkkominen pienempiin yksiköihin, jolloin erityisen useasti virheellisesti toimivan testitapauksen pystyi suorittamaan nopeasti ja tarvittaessa useaan kertaan. Samalla testitapausten päällekkäiset toteutukset siirrettiin jaettuun kirjastoon, mikä yksinkertaisti testitapausten toteutusta ja teki niistä helpommin hallittavia sekä luettavia.

Tässä vaiheessa testauksessa käytettiin myös muita kuin omia Mosquitto-välittäjiä. Testauksessa käytettiin seuraavia, ilmaisia ja vapaassa käytössä olevia välittäjiä:

- HiveMQ-välittäjä osoitteessa 35.156.5.61 (broker.hivemq.com)
- Mosquitto-välittäjä osoitteessa 37.187.106.16 (test.mosquitto.org)

Aiemmin oli testattu yksittäisen yhteyden aktiivisena pysymistä. Nyt pidempiaikainen testaus laajeni kahdella alueella:

- CTest-testien ajo useaan kertaan. Testiajot ja testitulokset automatisoitiin python-ohjelmalla. 500 kertaa ajetuissa CTest-testeissä ei löytynyt yhtään virhettä (windows-Bash). Yksi CTest-testiajo kesti 2 minuuttia, jolloin kokonaistestiajaksi tuli 16 tuntia.
    - Kaikki testit menivät läpi Windows Bash -ympäristössä.
    - Raspberry Pi -ympäristössä 25% CTest-testikerroista epäonnistui. Onneksi CTest-testeistä epäonnistui aina yksi testi, mikä helpotti korjaamista. Ongelma oli testiympäristössä ja korjauksen jälkeen kaikki testit suoritettiin onnistuneesti läpi 2000 kertaa (noin 3 päivää).
- Komentorivityökalulla viestien tilauksen ja pitkäaikaisen yhteyden ylläpidon testaus. Varmistaakseni yhteensopivuuden, käytössä oli yleisen välittäjän (test.mosquitto.org). Yhteyden ylläpitoajastin oli asetettuna 300 sekunnin arvoon. Erillisellä työkalulla (MQTT.Fx) lähetettiin tilattuun aiheeseen viestejä, millä varmistettiin komentorivityökalun yhteys välittäjään. Testin kesto oli 21 tuntia.
    - Windows Bash -ympäristössä toimi moitteettomasti
    - Raspberry Pi (ubuntu) -ympäristö toimi moitteettomasti.

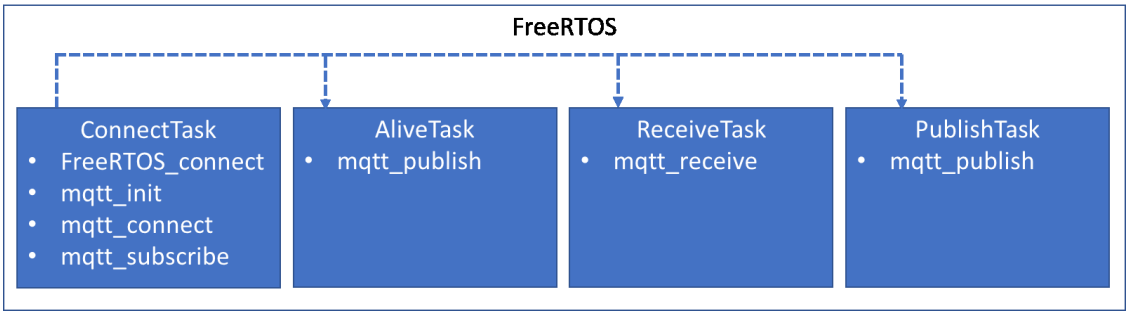
5.7.15 FreeRTOS

IoT-laitteet ovat yleisesti ottaen pienellä prosessorikapasiteetilla varustettuja laitteita, missä käyttöjärjestelmä on erittäin kevyt tai sitä ei ole ollenkaan. FreeRTOS on yksi IoT-laitteissa käytetty käyttöjärjestelmä. Tässä sprintissä varmistettiin aikaisemmissa sprinteissä tuotetun ja testatun MQTT-ohjelmiston toiminta FreeRTOS-ympäristössä. Testialustana toimi FreeRTOS-emulaattori, mitä ajettiin Windows-ympäristössä (FreeRTOS n.d.). Alkuperäinen suunnitelma oli käyttää Atmelin SAME70 xplained kehityskorttia, mutta kyseisen kortin sisäänrakennettu debug-piiri (EDBG) ei suostunut toimimaan, eikä vaihtoehtoista JTAG-laitteistoa ollut saatavilla.

FreeRTOS-emulaattorin kääntämiseen tarvittiin Visual Studio -ohjelmisto. Ohjelmistosta oli saatavilla ilmainen avoimen lähdekoodin käyttöön sallittu ja kaupalliseen käyttöön tarkoitetut versiot (Visual Studio downloads n.d.). Opinnäytetyötä tehdessä

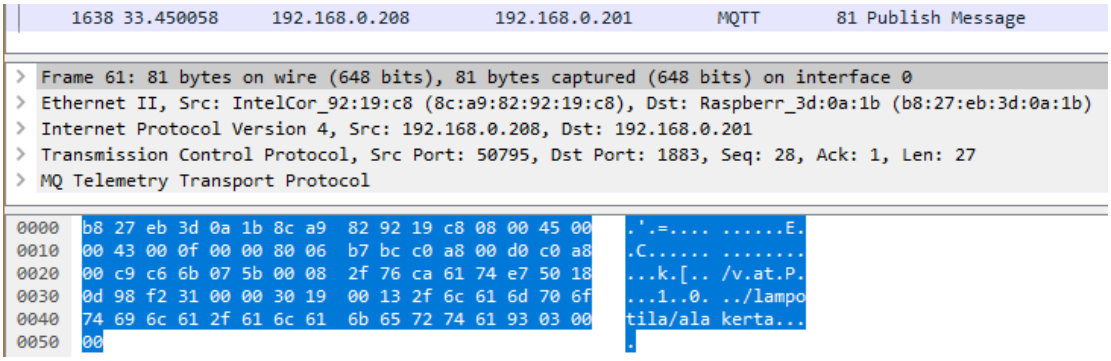
käytössä olivat kummatkin ohjelmistot. FreeRTOS-sivustolla oli valmiina ladattavissa FreeRTOS+TCP esimerkkitoteutus, jota muokkaamalla pystyi helposti saamaan Soket-yhteyden muodostavan version (FreeRTOS n.d.).

Arkkitehtuuriltaan FreeRTOS esimerkkisovellus koostuu neljästä eri taskista. Kaikki neljä taskia luodaan yhtä aikaa, mutta vain ConnectTask on suorituksessa. AliveTask, ReveiveTask ja PublishTask odottavat kaikki ConnectTask:n indikaatiota onnistuneesta yhteydestä ja vasta sen jälkeen aloittavat omat tehtävänsä. ConnectTask luo Soket- ja MQTT-yhteydet, minkä jälkeen se lähettää tilauksen halutusta aiheesta välittäjälle. AliveTask varmistaa PingReq-viestin lähetyksen yhteyden ylläpitoajastimen mukaisesti. ReceiveTask odottaa MQTT-viestien saapumista Soket-yhteyden yli ja prosessoi viestit niiden saavuttua. PublishTask julkaisee kuviossa 29 (sivu 33) esitetyn mukaista MQTT-viestiä yhden sekunnin välein. Arkkitehtuurimielessä FreeRTOS-käyttöjärjestelmän suorittaminen Windows-käyttöjärjestelmässä asettaa rajoituksen FreeRTOS-käyttöjärjestelmän tick-laskurin tarkkuudelle, mikä vastaavasti aiheutti ongelmia esimerkiksi AliveTask:ssa käytetyn viiveen osalta.



Kuvio 79. FreeRTOS-taskit

Kuviossa 80 on esitetty FreeRTOS-ympäristöstä lähetetty julkaisuviesti. Viesti on sama, kuin opinnäytetyön alussa esitetyssä kuviossa 29 (sivu 33). Tässä vaiheessa voidaankin todeta, että MQTT-, TCP- ja IP-otsakkeet ovat oletuksen mukaiset. Lähiverkko-otsakkeesta puuttuu lopussa olevat neljä tavua (FCS), jotka ilmaisevat viestin päättymistä. Wireshark-työkalun keskustelupalstalla todettiin, että kyseessä on ohjelmistovirhe. Voidaankin todeta, että alussa esitetty oletus viestin rakenteesta piti paikkansa.



Kuvio 80. FreeRTOS-viestin julkaisu

5.7.16 Parametrien tarkistus

Virheellisesti käytetty ohjelmistorajapinta voi aiheuttaa laitteiston kaatumisen tai avata tietoturva-aukon. Ohjelmistorajapinta ja käytettyjen parametrien sallitut arvot on syytä aina tarkistaa toiseenkin kertaan. Tämä kokonaisuus/sprint keskittyi rajapinnan parametrien tarkistukseen sekä siihen, miten niitä testattiin. Samalla löytyi yli-pitkä funktio, mikä jaettiin kolmeen pienempään funktioon.

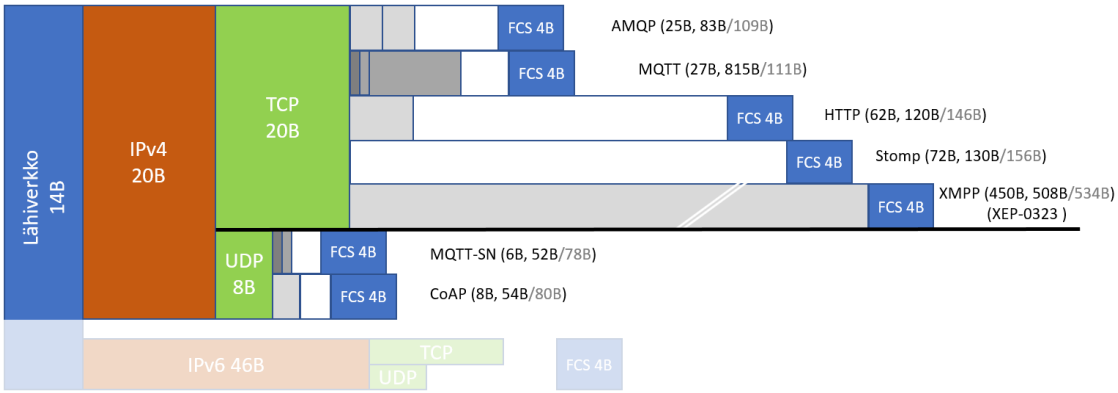
5.7.17 Kiillotus

Tässä vaiheessa määritelty asiakasohjelmisto oli toteutettu, mutta vielä tarvittiin lopullista kiillotusta. Vaikka miten tarkkaan ohjelmistoa kävisi läpi, niin aina jää jäljelle joitain koodi- ja kirjoitusvirheitä. Opinnäytetyö ei tehnyt tähän poikkeusta, vaan aiheen pituuskentän laskennassa oli selvä virhe, mikä löytyi vasta kun testi-tapauksista yhteen muutettiin pidempi aihe. Kaikkia ohjelmiston haaroja ei siis oltu käyty läpi. Kirjoitus- sekä tyylivirheitä korjattiin lisäksi muutamista kohdista.

6 Pohdinta

IoT-laitteiden määrä kasvaa lähitulevaisuudessa merkittävästi. Laitteiden määrällinen kasvu aiheuttaa vääjäämättä tietoliikenteen määrän kasvua. Siirtotiellä siirrettävän tietomäärän ja varsinaisen hyötykuorman välinen suhde eroaa merkittävästi käytetyimpien IoT-laitteiden protokollissa. Yhteenveto esiteltyjen protokollien vaatimasta siirtotien tilankäytöstä 4-tavun hyötykuorman siirrossa on esitetty

kuviossa 81. Kuviossa on esitetty TCP- ja UDP-pohjaiset protokollat, sekä IPv6-kehyksen merkitys tilankäyttöön. Yhden mittaustuloksen siirron analysointi ei kerro koko totuutta, vaan eri protokollien vaatimat yhteyden hallintaan ja ylläpitoon vaadittavat tietomäärät pitäisi analysoida, erityisesti UDP-pohjaisten protokollien osalta. MQTT-SN-protokolla ja protokollassa olevat ominaisuudet, kuten yhteyden ylläpidon dynaaminen muuttaminen ja viestien puskurointi välittäjään saattavat nousta merkittäviksi ominaisuuksiksi. Erityisesti 5G mMTC (massive Machine-type Communication) -verkkojen dynaamisen yhteyden ylläpidon yhdistäminen MQTT-SN-protokollan vastaavaan ominaisuuteen voisi tulevaisuudessa mahdollistaa varsin energiatehokkaan yhdistelmän.



Kuvio 81. Protokollien tilantarpeen yhteenveto

Laite- ja käyttäjähallintaa ei missään protokollassa ollut toteutettuna. IoT-laitteiden määrän merkittävä kasvu tulee todennäköisesti aiheuttamaan haasteita laitehallinnalle. Laitehallintaan täytyy käyttää jotain muuta menetelmää kuin opinnäytetyössä esitettyjä protokollia. Muutamassa IoT-protokollassa oli alustava mahdollisuus laite-/käyttäjähallinnan toteuttamiseen. Se, miten hyvin hallinnan ja protokollan sitomisessa onnistutaan, jää nähtäväksi.

Tietoturvan osalta kaikki protokollat luottavat alemman kerroksen tarjoamaan tietoturvaan (TLS/IPSec). Menetelmä sinänsä on looginen, mutta tilankäytöllisesti ei välttämättä paras mahdollinen isojen lohkokokojen vuoksi. Toisaalta alimman tason IPSec- eli VPN-yhteys lienee ainut, jota kannattaa käyttää kriittisissä yhteyksissä, sillä menetelmä varmistaa esimerkiksi IP-osoitteen muuttumattomuuden matkalla.

MQTT-protokolla käsiteltiin tarkemmin, sekä siitä tehtiin c-kielinen toteutus. Protokollan toiminnan tutkinnan sekä toteutuksen aikana ilmeni protokollasta muutamia optimoitavia ominaisuuksia:

Kiinteä otsake ei ole oikeasti kiinteä, vaan siinä on 1-4 tavun mittainen pituuskenttä. Protokollan mukaista data-pakettia rakennettaessa ensimmäisenä oleva kiinteä otsake on kuitenkin viimeinen osa viestistä, jolle voi varata tilan. Esimerkkitoteutuksessa ongelma ratkaistiin niin, että kiinteälle otsakkeelle varattiin aina tila, johon mahtui 4-tavua pitkä pituuskenttä. Viestiä kasatessa viimeisenä vaiheena kiinteä otsake kopioidaan siirrettävän datapaketin alkuun oikealle kohdalle, jättäen tarvittaessa alkuun tyhjää.

Viestiä julkaistaessa lähetetään aina aihe osana julkaistavaa viestiä. Aihe on yleensä usean kirjaimen mittainen, mikä aiheuttaa turhaa verkkoliikennettä. MQTT-SN-protokollassa ongelma on korjattu lähettämällä aiheelle annettu indeksi, mikä vähentää merkittävästi siirrettävän tiedon määrää.

Viestien tilaaminen täytyy tehdä aina laatutasolla 1, jolloin välittäjä lähettää kuittauksen onnistumisesta. Puhtaasti laatutason 0 toteutusta ei voi siis tehdä. Tilaamisen osalta Mosquitto-välittäjä salli viestien tilaamisen myös laatutasolla 0, jolloin välittäjä ei myöskään lähettänyt kuittausta.

Yhteydenotossa määritellään protokollan nimi tekstinä, mutta versionumero esitettynä yhdellä tavulla. Periaatteessa olisi ollut hyvinkin mahdollista jättää protokollan nimi pois. tässä olisi voinut hyvinkin jättää protokollan nimen pois. Onneksi yhteydenottoviestejä ei kuitenkaan tarvitse lähettää kovin usein.

Mosquitto-välittäjä katkaisi Soket-yhteyden, jos yhteydenottoviestissä oli yksikin väärä parametri. MQTT-määritelmä mahdollistaisi virhekoodin palautuksen, mutta tätä mahdollisuutta Mosquito-välittäjässä ei käytetty hyväksi.

Toteutuksessa käytettiin C-ohjelmointikieltä ja toteutustapana TDD-tapaa. Ohjelmointikieli oli ennestään tuttu, kuten myös TDD. Testitapausten määritteleminen ja toteuttaminen ennen varsinaisen ohjelmiston toteutusta osoitti jälleen tehokkuutensa (TDD). Alussa aikaa kului merkittävästi siihen, että testijärjestelmän ja testitapaukset sai luotettavasti toimimaan. Alussa käytetty aika maksoi tässäkin toteutuksessa itsensä takaisin, sillä useamman kerran muutaman bitin muutos vaikuttikin oletettua useampaan paikkaan. Toteutus sinänsä oli kohtalaisen suoraviivainen operaatio, sillä MQTT-määritelmä oli saatavilla ja välittäjästä useita toteutuksia tarjolla. Viestien ja viestisekvenssien analysointiin Wireshark-työkalu oli erinomainen apuväline.

Toteutuksen testaamista helpotti Linux-ympäristöön toteutettu komentorivityökalu, jonka osana oli toteutettu MQTT-protokollapino. Komentorivityökalulla pystyi testaamaan helposti eri parametreja sekä julkaisemaan erikokoisia viestejä. Työkalu mahdollisti myös pitkäkestoiset testaukset.

IoT-laitteiden protokollien tutkiminen ja MQTT-asiakasohjelmiston toteutus antoivat hyvän kuvan IoT-laitteiden kommunikoinnin tärkeydestä ja toiminnasta. MQTT-protokollan toiminta tuli varsin tutuksi opinnäytetyötä tehdessä. Mielestäni protokolla ja erityisesti MQTT-SN-protokolla ovat hyviä vaihtoehtoja IoT-laitteen protokolliksi. MQTT-protokolla on yksinkertainen ja se on ollut käytössä useita vuosia. MQTT-protokollaan löytyy tuki useille eri ohjelmointikielille sekä käyttöjärjestelmille. MQTT-protokollalle löytyy useita erilaisia työkaluja, jotka helpottavat testaamista. Yleisesti tilaaja- ja tuottajamalli (publish/subscribe) mahdollistaa joustavan arkkitehtuurin, mikä on varsin tärkeä näkökulma nopeasti muuttuvassa IoT-maailmassa.

Lähteet

AMQP 1.0 support in Service Bus. 2017. Microsoft Azure verkkosivusto. Viitattu 20.4.2017. https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview.

Ashton, K. 2009. RFID-sivuston julkaisu: That 'Internet of Things' Thing. Viitattu 19.3.2017. http://www.rfidjournal.com/articles/view?4986.

AWS protocols. 2017. AWS IoT developer guide sivusto. Viitattu 31.10.2017. http://docs.aws.amazon.com/iot/latest/developerguide/protocols.html.

Azure IoT protocol gateway. 2017. Microsoft Azure IoT Hub dokumentaatio. Viitattu 31.10.2017. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-protocol-gateway.

Benchmark of MQTT servers. 2015. Versio 1.1. Viitattu 17.5.2017. http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

Boyd, B., Gauci, J., Robertson, M., P., Duy, N., V., Gupta, R., Gucer, V., Kislicins, V. 2014. Building Real-time Mobile Solutions with MQTT and IBM MessageSight. IBM Redbook julkaisu. Viitattu 17.5.2017. http://www.redbooks.ibm.com/redbooks/pdfs/sg248228.pdf

Chapter 2. Exchanges. N.d. Red Hat Message routing verkkosivusto. Viitattu 24.3.2017. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/1.1/html/Messaging_User_Guide/chap-Messaging_User_Guide-Exchanges.html.

Clements, V. 2015. Johdatus AMQP-teknologiaan. Slideshare verkkosivusto (https://creativecommons.org/licenses/by/4.0/). Viitattu 25.3.2017. https://www.slideshare.net/ClemensVasters/amqp-10-introduction.

Clients & Developer Tools. N.d. Rabbit MQ verkkosivusto. Viitattu 20.4.2017. http://www.rabbitmq.com/devtools.html.

CMake. N.d. CMake-verkkosivusto. Viitattu: 19.8.2017. https://cmake.org.

CMake/Testing With CTest. 2016. CMake wiki -sivusto. Viitattu 18.6.2017. https://cmake.org/Wiki/CMake/Testing_With_CTest.

CodeDocs. 18.5.2016. CodeDocs wiki github -verkkosivustolla. https://github.com/CodeDocs/CodeDocs/wiki.

Columbus, L. 2016. Roundup Of Internet Of Things Forecasts And Market Estimates. Viitattu 9.4.2017. https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#56254f60292d.

Cope, S. 2017. Mosquitto MQTT Bridge-Usage and Configuration. Steve's internet Guide -verkkosivusto. Viitattu 17.5.2017. http://www.steves-internet-guide.com/mosquitto-bridge-configuration.

Cppcheck. N.d. Cppcheck-sivusto. Viitattu 18.6.2017. http://cppcheck.sourceforge.net.

Designing the Internet of Things. N.d. Micrium-verkkosivusto. Viitattu 25.3.2017. https://www.micrium.com/iot/internet-protocols.

Documentation | Mosquitto. N.d. Dokumentaatio Mosquitto-verkkosivustolla. Viitattu 4.5.2017. https://mosquitto.org/documentation.

Dostálek, L & Kabelová, A. 2006. Understanding TCP/IP. Packt Publishing. ISBN:9781847190567.

Doxygen. 29.12.2016. Doxygen-verkkosivusto. Viitattu 19.8.2017. http://www.stack.nl/~dimitri/doxygen/index.html.

Evans, D. 2013. Answering the Two Most-Asked Questions About the Internet of Everything IoE. Viitattu 23.3.2017. http://blogs.cisco.com/digital/answering-the-two-most-asked-questions-about-the-internet-of-everything.

FreeRTOS. N.d. FreeRTOS-verkkosivusto. Viitattu 18.9.2017. http://www.freertos.org/index.html.

Friedel, S. 2005. An Illustrated Guide to IPsec. Viitattu 27.3.2017. http://www.unixwiz.net/techtips/iguide-ipsec.html.

Galezowski, G. 2017. Test-Driven Development: Extensive Tutorial. Viitattu 16.9.2017. https://github.com/grzesiek-galezowski/tdd-ebook.

Gartner's 2015 hype cycle for emerging technologies identifies the computing innovations that organizations should monitor. 2015. Viitattu 18.3.2017. http://www.gartner.com/newsroom/id/3114217.

Gast, M. 2005. 802.11 Wireless Networks: The Definitive Guide, 2.p. Sebastopol US. O'Reilly Media. ISBN:9781449319526.

Gilchrist, A. 2016. Industry 4.0: The Industrial Internet of Things. Apress. ISBN:9781484220467.

Git. 19.3.2016. Linux wikipedia. Viitattu 18.8.2017. https://www.linux.fi/wiki/Git.

Google Trends. 2017. Googlen statistiikka verkkosivusto, IoT hakusanojen käyttö Google palvelussa. Viitattu 17.4.2017. https://trends.google.com/trends/explore?date=2007-01-01%202017-04-17&q=IOT&hl=en-US.

Grigorik, I. 2014. Google's Ilya Grigorik on HTTP 2.0. Viitattu 2.4.2017. https://www.slideshare.net/heavybit/heavybit-presents-ilya-grigorik-on.

Gupta, S. 2016. About MQTT Servers & Brokers. Viitattu 17.5.2017. https://sachingpta.gitlab.io/_posts/mqtt-servers-or-brokers.html

How the AWS IoT Platform Works. N.d. Amazon Webservices verkkosivusto. Viitattu 22.4.2017. https://aws.amazon.com/iot-platform/how-it-works.

Hunter, J. 2015. Crunch Network verkkosivusto, The Hierarchy of IoT “Thing” Needs . Crunch Network. Viitattu 19.32017. https://techcrunch.com/2015/09/05/the-hierarchy-of-iot-thing-needs.

Huston. G. 2003. Waiting for IP version 6. Viitattu 26.3.2017. http://www.potaroo.net/ispcol/2003-01/Waiting.html.

IEEE 802.11. 2012. Wireless LANs. Viitattu 9.4.2017. http://standards.ieee.org/about/get/802/802.11.html.

IEEE 802.3. 2015. Ethernet. Viimeisin muutos 17.3.2017. Viitattu 9.4.2017. http://standards.ieee.org/about/get/802/802.3.html.

Implementations. N.d. CoAP-verkkosivusto. Viitattu 21.4.2017. http://coap.technology/impls.html.

Internet of Things: Visualise the Impact. 2016. SAS instituutin julkisen- ja yksityisen sektorin IoT kyselyn pohjalta tehty IoT-teknologian tulevaisuus analyysi. Viitattu 17.4.2017. https://www.sas.com/sas/offers/16/iot-visualise-the-impact-2458546.html?gclid=CMic2oK24tICFRBeGQodDZUIpA.

ISO/IEC:19464. 2014. Information technology -- Advanced Message Queuing Protocol (AMQP) v1.0 specification. ISO-organisaation verkkosivusto. Viitattu 20.42017. http://standards.iso.org/ittf/PubliclyAvailableStandards/c064955_ISO_IEC_19464_2014.zip.

Kovatsch, M. 2017. Copper laajennoksen GitHub sivusto. Viitattu 31.10.2017. https://github.com/mkovatsc/Copper.

Lampkin, V., Leong, W., T., Olivera, L., Rawat, S., Subrahmanyam, N., Xiang, R. 2012. Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry. IBM Redbook julkaisu. Viitattu 17.5.2017. http://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf

Lewis, J. N.d. MQTT Tutorial for Raspberry Pi, Arduino, and ESP8266. Viitattu 17.5.2017. https://www.baldengineer.com/mqtt-tutorial.html

Libelium Smart World. 2013. Infografiikka Libelium-yhtiön verkkosivustolla. Viitattu 17.4.2017. http://www.libelium.com/wp-content/themes/libelium/images/content/applications/libelium_smart_world_infographic_big.png.

Moffitt, J. 2010. Professional XMPP Programming with JavaScript and JQuery. John Wiley & Sons. ISBN:9780470606773.

MQTT Essentials Part 5: MQTT Topics & Best Practices. N.d. HiveMQ verkkosivuston MQTT-ohje. Viitattu 30.4.2017. http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices.

MQTT Essentials: Part 1 – Introducing MQTT. N.d. HiveMQ-verkkosivusto. Viitattu 22.4.2017. http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt.

MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2. 2013. Viitattu 15.4.2017. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

MQTT Version 3.1.1. 2014. OASIS-yhteisön standardi. Viitattu 22.4.2017. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.

Node-RED Flow-based programming for the Internet of Things. N.d. Viitattu 9.4.2017. https://nodered.org.

Paterson, P., G. & AlFardan, N., J. 2013. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. Royal Holloway, University of London. Viitattu 20.4.2017. http://www.isg.rhul.ac.uk/tls/TLStiming.pdf.

Perez, A. 2014. Network Security. Wiley. ISBN:9781119043966.

Protocol Buffers. N.d. Googlen Protocol Buffers -työkalusivusto. Viitattu 18.6.2017. https://developers.google.com/protocol-buffers.

RabbitMQ Simulator. N.d. Ohjelmisto verkkosivulla. Viitattu 24.3.2017. http://tryrabbitmq.com.

RFC2409. 1998. The Internet Key Exchange (IKE). Viitattu: 28.3.2017. https://www.ietf.org/rfc/rfc2409.txt.

RFC2616. 1999. Hypertext Transfer Protocol -- HTTP/1.1. Viitattu 2.4.2017. https://tools.ietf.org/html/rfc2616.

RFC4302. 2005. IP Authentication Header. Viitattu 28.3.2017. https://tools.ietf.org/html/rfc4302.

RFC5246. 2008. The Transport Layer Security (TLS) Protocol. Viitattu 23.3.2017. https://tools.ietf.org/html/rfc5246.

RFC6101. 2011. The Secure Sockets Layer (SSL) Protocol Version 3.0. Viitattu 19.4.2017. https://tools.ietf.org/html/rfc6101.

RFC6347. 2012. Datagram Transport Layer Security Version 1.2. Viitattu 30.3.2017. https://tools.ietf.org/html/rfc6347.

RFC7252. 2014. The Constrained Application Protocol (CoAP). IEFT-verkkosivusto. Viitattu 2.4.2017. https://tools.ietf.org/html/rfc7252.

RFC7540. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). Viitattu 2.4.2017. https://tools.ietf.org/html/rfc7540.

Schneider, S. 2013. Understanding The Protocols Behind The Internet Of Things. Viitattu 23.3.2017. http://electronicdesign.com/iot/understanding-protocols-behind-internet-things.

Scrum. 2017. Scrum-termin määritelmä Wikipediassa. Viitattu 2.10.2017. https://fi.wikipedia.org/wiki/Scrum.

Shelby, Z. 2013. CoAP esittely slideshare verkkosivustolla. Viitattu 2.4.2017. https://www.slideshare.net/zdshelby/coap-tutorial.

Special sections in Linux binaries. N.d. LWN.net-verkkosivusto. Viitattu 16.9.2017. https://lwn.net/Articles/531148.

SSL/TLS Client Certs to Secure MQTT. 2015. Rocking Labs -verkkosivusto. http://rockingdlabs.dunmire.org/exercises-experiments/ssl-client-certs-to-secure-mqtt.

STOMP Protocol Specification, Version 1.2. 2012. Viitattu 9.4.2017. https://stomp.github.io/stomp-specification-1.2.html.

Stupp, C. 2016. Euroactive verkkosivuston artikkeli: Commission plans cybersecurity rules for internet-connected machines. Viitattu 29.4.2017. http://www.euractiv.com/section/innovation-industry/news/commission-plans-cybersecurity-rules-for-internet-connected-machines.

The MIT License. N.d. Open Source Initiative -verkkosivusto. Viitattu 18.8.2017. https://opensource.org/licenses/MIT.

Towards a definition of the Internet of Things (IoT). 2015. IEEE työryhmän IoT-termin määritys. Viitattu 17.4.2017. http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf.

Unity. N.d. Throw The Switch verkkosivusto. Viitattu 18.6.2017. http://www.throwtheswitch.org/unity.

Visual Studio downloads. N.d. Visual Studio -verkkosivusto. Viitattu 18.9.2017. https://www.visualstudio.com/downloads.

Waher, P. 2015. Learning Internet Of Things. Birmingham: Packt Publishing Ltd. ISBN:978-1-78355-353-2.

Welcome to the home of MQTT.fx. N.d. MQTT.FX-asiakasohjelmiston verkkosivusto. Viitattu 4.5.2017. http://www.mqttfx.org.

XEP-0163. 2010. XMPP XEP-0163: Personal Eventing Protocol. Viitattu 1.4.2017. https://xmpp.org/extensions/xep-0163.pdf.

XEP-0323. 2015. XAMP XEP-0323: Internet of Things - Sensor Data. Viitattu 1.4.2017. https://xmpp.org/extensions/xep-0323.pdf.

XMPP software. N.d. XMPP serverit, asiakasohjelmistot ja kirjastot XMPP verkkosivustolla. Viitattu 22.4.2017. https://xmpp.org/software.

XMPP Specifications. N.d. XMPP verkkosivusto, XMPP määritykset. Viitattu 22.4.2017. https://xmpp.org/extensions.

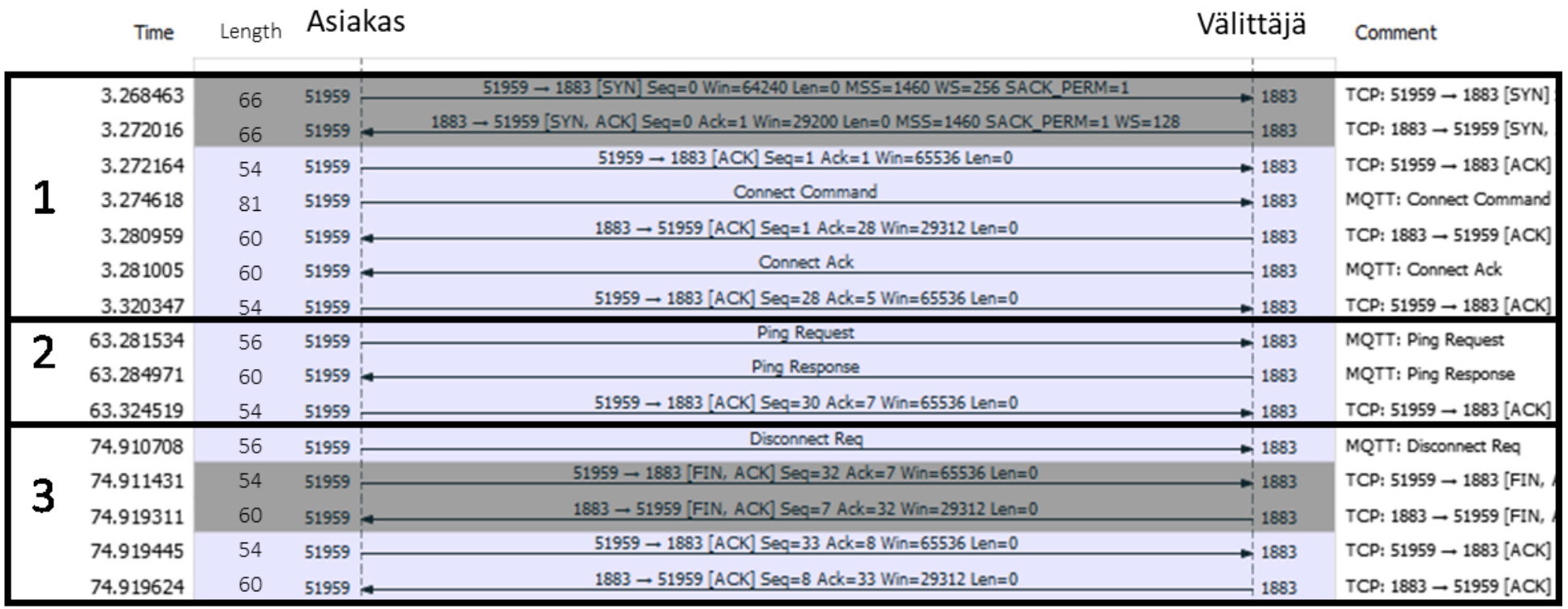
Liitteet

Liite 1. IoT-protokollien lyhyt yhteenveto

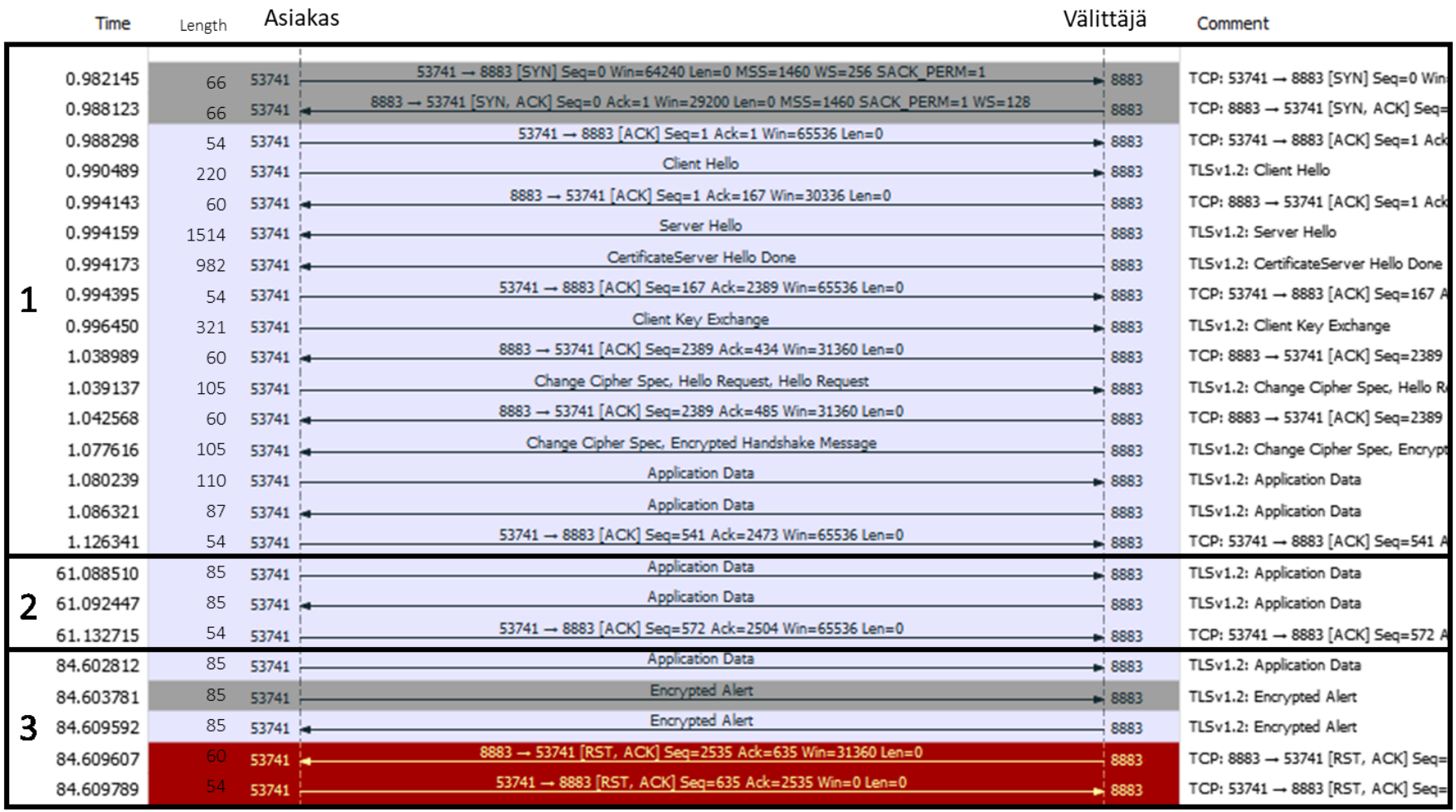
	AMQP	**CoAP**	**HTTP/HTTPS**	**MQTT**	**MQTT-SN**	**STOMP**	**XMPP**
Kuljetuskerros	TCP	UDP	TCP	TCP	UDP	TCP	TCP
Salaus	TLS/SASL	DTLS	TLS	TLS/WSS	-	TLS	TLS/SASL
Hyötykuorma	Formaali ja vapaa	Formaali	Vapaa	Vapaa	Vapaa	Vapaa teksti	Formaali
IoT-pilvipalvelujen tuki *	Suurin osa tukee	Vähäinen tuki	Suurin osa tukee	Suurin osa tukee	Ei mainintaa	Rajoitettu tuki lisäosilla	Ei mainintaa
Esimerkkiviestin koko lähiverkossa	83B	54B	120B	85B	54B	134B	508B
Toimintatapa	Tilaaja, tuottaja	Pyyntö, vastaus	Pyyntö, vastaus	Tilaaja, tuottaja	Tilaaja, tuottaja	Tilaaja, tuottaja	Tilaaja, tuottaja
Standardi	ISO/IEC 19464 (OASIS)	IEFT RFC7257	IEFT RFC2616 / IEFT RFC7540	ISO/IEC 20922 (OASIS)	(IBM MQTT-SN v1.2)	(STOMP org.)	IEFT RFC6120 (XMPP org.)
Muuta	Tarjoaa luotettavan yhteyden.	Yleinen omasta laiteverkosta koostuva järjestelmä.	Yksinkertainen, laaja ohjelmisto tuki ja paljon työkaluja.	Yleinen IoT-protokolla.	Analogia CoAP protokollaan – suljettu sensori verkko.	Käyttö mahdollista Telnet-yhteydellä.	Tuki IoT-viestirakenteille, mutta käyttö vähäistä.

** ARTIK, Azure, AWS, Bluemix, thethings.io*

Liite 2. MQTT-ytheyden muodostaminen ja päättäminen



Liite 3. TLS-menetelmällä salatun MQTT-ytheyden muodostaminen ja päättäminen



Liite 4. Varmenteiden ja avaimien luominen

Luotetun varmentajan sekä välittäjän avaimien luominen valmista komentosarjaa käyttäen Linux-käyttöjärjestelmässä (SSL/TLS Client Certs - - 2015):

```
wget https://github.com/owntracks/tools/raw/master/TLS/generate-CA.sh .
bash ./generate-CA.sh
sudo cp ca.crt /etc/mosquitto/ca_certificates/
sudo cp myhost.crt __host_name__.key /etc/mosquitto/certs
```

Asiakkaan varmenteen ja avaimen luominen:

```
openssl genrsa -out client.key 2048
openssl req -new -out client.csr -key client.key  -subj "/CN=client/O=example.com"
openssl x509 -req -in client.csr -CA ca.crt  -CAkey ca.key -CAserial ./ca.srl -out cli-
ent.crt  -days 3650 -addtrust clientAuth
```

MQTT.FX vaatii toimiakseen varmenteet der-formaatissa, mihin muuntaminen onnistuu seuraavilla komennoilla:

```
openssl x509 -outform der -in ca.crt -out ca.der
openssl x509 -outform der -in client.crt -out client.der
```

Vaihtoehtoinen tapa on käyttää valmista Linux-ympäristössä toimivaa komentorivityökalua, mistä esimerkki alla:

```
wget https://raw.githubusercontent.com/owntracks/tools/master/TLS/generate-CA.sh
./geterate-CA.sh do.rmot.xyz
sudo cp ca.crt /etc/mosquitto/ca_certificates/.
sudo cp __hostname__.crt /etc/mosquitto/certs/.
sudo cp  __hostname__.key /etc/mosquitto/certs/.
```

Liite 5. Mosquitto-välittäjän asetukset salatulle ja salaamattomalle yhteydelle

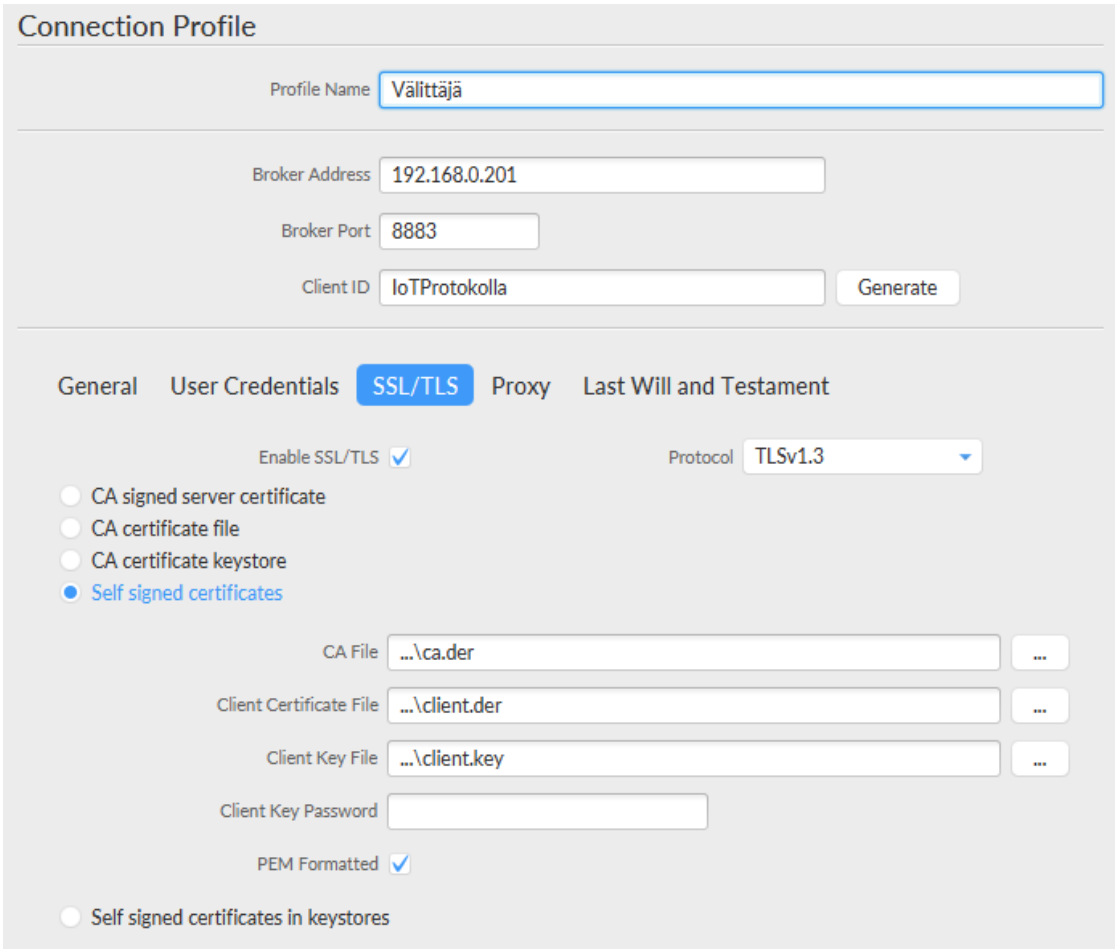
Ubuntu Linux -jakelussa muokattava Mosquitto-ohjelmiston alustustiedosto on: /etc/mosquitto/mosquitto.conf. Oletuksena luotetun varmentajan varmenne sijaitsee hakemistossa /etc/mosquitto/ca_certificate. Vastaavasti välittäjän varmenne sijaitsee oletusarvoisesti hakemistossa /etc/mosquitto/certs.

```
pid_file /var/run/mosquitto.pid
persistence_location /var/lib/mosquitto/
persistence_file mosquitto.db
persistence true
# log_dest file /var/log/mosquitto/mosquitto.log
include_dir /etc/mosquitto/conf.d
connection_messages true

listener 1883
# Salaamaton yhteys

listener 8883
# Salattu yhteys
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/__host_name__.crt
keyfile /etc/mosquitto/certs/__host_name__.key
```

Liite 6. TLS-asetukset MQTT.FX-ohjelmistoon



Liite 7. Yksinkertainen MQTT-viestin tilaus ja julkaisu Python-kielellä

```
import paho.mqtt.client as mqtt # pip install paho-mqtt

mqttserver = "192.168.0.203"

def on_connect(client, userdata, rc):
    print("Yhdistetty " + str(rc))
    #Julkaistaan oma tila
    client.publish("iot/laitex/nappula/1", 1)
    client.publish("iot/laitex/nappula/2", 0)
    #Tilataan ohjausviestit
    client.subscribe("iot/laitex/nappula/+",0)

def on_message(client, userdata, msg):
    print(msg.payload)
    print(msg.topic)

def main():
    mqttc = mqtt.Client()
    # Yhteyden muodostumisen jalkeen kutsutaan on_connect funktiota
    mqttc.on_connect = on_connect
    # Tilattuun aiheeseen saapuvat viestit menevat on_message funktioon
    mqttc.on_message = on_message
    # Yhdistetaan valittajaan
    mqttc.connect(host=mqttserver, port=1883, keepalive=60)
    print("MQTT")
    mqttc.loop_forever()

if __name__ == "__main__":
    main()
```

Liite 8. MQTT Python asiakkaan tilausten talletus Mongo-tietokantaan

Python-asiakas käyttää paho- ja pymongo-moduuleita hyväkseen. Kyseiset moduulit voi asentaa suoraan Python PIP-palvelusta.

```
import paho.mqtt.client as mqtt # pip install paho-mqtt
import json
import pymongo # pip install pymongo

mqttserver  = "127.0.0.1"
mongoserver = "127.0.0.1"

def on_connect(client, userdata, rc):
    # Tilataan kaksi aihetta laatutasolla 0.
    client.subscribe([("IoT/nappula"   ,0),
                      ("lampotila/#"   ,0)])

def on_message(client, userdata, msg):
    try:
        # Listataan MQTT viestin sisalto
        # msg.topic voidaan kayttaa jakamaan tieto eri dokumentteihin
        mongoc.iot.data.insert_one(json.loads(msg.payload))
    except:
        pass

def main():
    # Yhdistetaan MongoDB palvelimeen
    mongoc = pymongo.MongoClient(mongoserver, serverSelectionTimeoutMS=2)
    mongoc.server_info()

    # Luodaan MQTT asiakas
    mqttc = mqtt.Client()
    # Yhteden modostuessa kutsutaan on_connect funktiota
    mqttc.on_connect = on_connect
    # Viestin saapuessa kutsutaan on_message funktiota
    mqttc.on_message = on_message
    # Yhdistetaan valittajaan
    mqttc.connect(mqttserver, 1883, 60)
    # Palvellaan MQTT asiakasta loputtomasti
    mqttc.loop_forever()

if __name__ == "__main__":
    main()
```

Liite 9. VPN-asetukset

Liitteessä on käytetty lähteenä DigitalOcean-sivuston OpenVPN käyttöönotto-ohjetta: https://www.digitalocean.com/community/tutorials/how-to-set-up-an-openvpn-server-on-ubuntu-16-04 ja OpenVPN virallista sivustoa https://openvpn.net/.

Palvelimen asetukset

VPN-palvelin tarvitsee toimiakseen palvelinohjelmiston, varmenteet ja avaimet. Palvelinohjelmistoksi tässä on valittu yleinen OpenVPN ja avainten hallintaan OpenVPN:n easy-rsa -ohjelmisto. Kyseiset ohjelmistot asentuvat Ubuntuun seuraavilla komennoilla:

```
sudo apt-get update
sudo apt-get install openvpn easy-rsa
```

Asennettujen ohjelmistojen jälkeen luodaan varmentaja (CA – certificate authority):

```
make-cadir ~/openvpn-ca
cd ~/openvpn-ca
```

Varmentajan yhteystiedot täytyy päivittää:

```
nano ~/openvpn-ca/vars
...
export KEY_COUNTRY="FI"
export KEY_PROVINCE="NA"
export KEY_CITY="Pirkkala"
export KEY_ORG="Oma org"
export KEY_EMAIL="admin@oma.org"
export KEY_OU="IOT"
...
export KEY_NAME=”IoTvalittaja”
...
```

Varmentajan yhteystietojen ja palvelimen nimen päivittämisen jälkeen voidaan luoda varsinainen varmentaja. ”build-ca”-komento kysyy vielä erikseen aiemmin määritellyt varmentajan tiedot, joten nyt nämä kysymykset voi ohittaa oletuksin.

```
cd ~/openvpn-ca
source vars
./clean-all
./build-ca
```

Seuraavaksi luodaan palvelimelle oma avain, varmenne ja salaustiedostot. Seuraavalle komennolle annettu parametri täytyy olla sama, kuin aiemmin määrätty varmentajan KEY_NAME -,arvo. Seuraavan komennon oletusparametrit ovat sopivat, kunhan valitaan että varmenne allekirjoitetaan (Sign the certificate = y), eikä aseteta varmenteelle salasanaa (challenge password). Diffie-Helmman avaimet generoidaan ”build-dh” -ohjelmalla ja avaimiin lisätään allekirjoitus openvpn-komennolla.

```
./build-key-server server
./build-dh
openvpn --genkey --secret keys/ta.key
```

Palvelimen konfigurointi

Luodut varmentaja, varmenne ja avaimet täytyy kopioida oikeaan paikkaan (/etc/openvpn):

```
cd ~/openvpn-ca/keys
sudo cp ca.crt ca.key server.crt server.key ta.key dh2048.pem /etc/openvpn
```

Palvelimen konfiguraatio täytyy määrittää. Seuraavalla komennoilla otetaan oletuskonfiguraatio käyttöön. Sen lisäksi täytyy muutama parametri muuttaa kyseisestä tiedostosta:

```
gunzip -c /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz | sudo tee
```

```
/etc/openvpn/server.conf

sudo nano /etc/openvpn/server.conf
… ota “redirect-gateway” kommenteista
… ota “dhcp-option” kommenteista pois (kaksi riviä)
… aseta ”tls-auth ta.key 0” ja lisää:
… key-direction 0
… lopuksi otetaan kommenteista pois seuraavat rivit ”user nobody” ja ”group nogroup”
```

OpenVPN-asetusten tiedosto (/etc/openvpn/server.conf):

```
port 1194                               # portti
proto udp                               # UDP protokolla
dev tun                                 # Reititetty IP tunneli
ca ca.cert                              # Luetun varmentajan varmenne
cert server.crt                         # Valittajan varmenne
key server.key                          # Valittajan salainen avain
dh dh2048.pem                           # Diffie-Helman määritys
server 10.8.0.0 255.255.255.0           # VPN verkon osoite ja aliverkon peite
ifconfig-pool-presists ipp.txt          # Asiakkaiden IP tallennus tiedosto
client-config-dir /etc/openvpn/ccd      # Asiakkaiden IP määrittelyjen hakemisto
route 10.8.0.0 255.255.255.252
push ”redirect-gateway def1 bypass-dhcp”# Asiakkaiden kaikki liikenne VPN serverin läpi
push ”dhcp-options DNS 8.8.8.8 ”        # Asiakkaiden DNS palvelimen osoite
keepalive 10 60                         # 10s ping-viestien lähetystiheys ja 60s timeout
tls-auth ta.key 0
key-direction 0
```

```
comp-lzo                                # Liikenteen pakkaus
user nobody
group nogroup
persist-key
persist-tun
```

VPN-konfiguroinnin lisäksi täytyy muutama verkkoparametri muuttaa. Ensiksi täytyy sallia IP-pakettien uudelleenlähetys:

```
sudo nano /etc/sysctl.conf
net.ipv4.ip_forward=1
```

Myös palomuurin sääntöjä täytyy päivittää sallimalla liikenne VPN-yhteyksiltä tässä tapauksessa eth0 -verkkokortin läpi tapahtuvaan liikenteeseen:

```
sudo nano /etc/ufw/before.rules
…
# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to eth0
-A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
COMMIT
# END OPENVPN RULES
```

Sallitaan vielä VPN-liikenteen porttiin sisään tulevat yhteydet ja käynnistetään palomuuri uudelleen. Samalla sallitaan VPN-verkosta kaikki liikenne palvelimelle.

```
sudo ufw allow 1194/udp
sudo ufw allow from 10.8.0.0/24
sudo ufw disable
sudo ufw enable
```

Seuraava paketti kannattaa asentaa myös, että palomuurin säännöt tallentuvat pysyvästi:

```
apt-get install iptables-persistent
```

Uudelleen käynnistetään VPN-serveri:

```
sudo service openvpn restart
```

Palvelimen lisämääritykset

Keepalive-määritys kannattaa ottaa käyttöön palvelimen VPN-asetuksista (/etc/openvpn/server.conf), silloin kun on valittuna automaattinen asiakkaan yhteydenmuodostus. Tämä mahdollistaa sen, että asiakas yrittää yhteydenottoa uudelleen, jos palvelin ei ole vastannut keepalive-viesteihin.

VPN-asiakkaille voi antaa kiinteän IP-osoitteen tekemällä asiakkaan nimellä (varmenteelle annettu nimi) tiedoston, jossa määritellään asiakkaan IP-osoite sekä palvelimen IP-osoite.

```
sudo nano /etc/openvpn/server.conf
ota seuraava rivi kommenteista client-config-dir ccd
echo ifconfig-push 10.8.0.11 10.8.0.1 | sudo tee /etc/openvpn/ccd/oma_iot_laite
```

Asiakkaiden asetusten ja varmenteiden asetukset

Asiakkaiden konfigurointitiedostojen luominen aloitetaan tekemällä palvelimelle hakemisto, mihin vain pääkäyttäjällä on oikeus. Seuraavaksi kopioidaan oletus konfigurointitiedosto, jota sitten muokataan sopivaksi.

```
mkdir -p ~/client-configs/files
chmod 700 ~/client-configs/files
cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf ~/client-con-
figs/base.conf
nano ~/client-configs/base.conf
… asetetaan serverin IP osoite … remote server_IP_address 1194
… poistetaan “user nobody” ja “group nogroup” määritykset kommenteista.
… kommentoidaan ”ca ca.crt”, ”cert client.crt” ja “key client.key” rivit.
… lisätään “key-direction 1” tiedostoon
```

Asiakkaan OpenVPN -konfigurointi asetukset (base.conf):

```
client
dev tun
proto udp
resolv-retry infinite # yritetaan ottaa yhteytta loputtomasti
nobind
group nogroup
persist-key
persist-tun
key-direction 1
renite.cert.tks sever
com-lzo
```

Asiakkaiden OpenVPN konfigurointi- ja varmennetiedoston luontia varten täytyy suorittaa seuraavaa ohjelma, jossa käytetään aiemmin määriteltyjä hakemistoja:

```
nano ~/client-configs/make_config.sh
… Lisätään seuraavat rivit:
```

```
#!/bin/bash

# First argument: Client identifier

KEY_DIR=~/openvpn-ca/keys
OUTPUT_DIR=~/client-configs/files
BASE_CONFIG=~/client-configs/base.conf

cat ${BASE_CONFIG} \
    <(echo -e '<ca>') \
    ${KEY_DIR}/ca.crt \
    <(echo -e '</ca>\n<cert>') \
    ${KEY_DIR}/${1}.crt \
    <(echo -e '</cert>\n<key>') \
    ${KEY_DIR}/${1}.key \
    <(echo -e '</key>\n<tls-auth>') \
    ${KEY_DIR}/ta.key \
    <(echo -e '</tls-auth>') \
    > ${OUTPUT_DIR}/${1}.ovpn
```

Annetaan tiedostolle vielä sopivat oikeudet:

```
chmod 700 ~/client-configs/make_config.sh
```

Asiakkaiden varmenteiden luonti

Asiakkaiden varmenteet luodaan palvelimella samassa hakemistossa, missä varmentajan tiedot ja palvelimen avaimet yms. luotiin (~/openvpn-ca). Asiakkaiden varmenteiden luonti onnistuu seuraavilla komennoilla. ”build-key”-komennon parametri määrittää asiakkaan nimen, eli se kannattaa olla kuvaava ja yksilöllinen. Asiakkaan nimen perusteella voi määrittää asiakkaalle esim. kiinteän IP-osoitteen. Seuraaviin kysymyksiin täytyy vastata y: Sign the certificate ja certificate requests certified, commit.

```
cd ~/openvpn-ca
source vars
./build-key ilto

cd ~/client-configs
./make_config.sh yhteys
```

Luotu konfigurointi ja varmenne tiedosto löytyy hakemistosta:

```
~/client-configs/files
```

Asiakkaan asetukset

Ensimmäiseksi täytyy asentaa OpenVPN-ohjelmisto:

```
sudo apt-get install openvpn
```

Palvelimelta kopioidaan xxxx.ovpn tiedosto (~/client-configs/files/oma_iot_laite.ovpn), joka pitää sisällään kaikki yhteyden muodostamiseen tarvittavat tiedot. Yhteyden muodostaminen onnistuu seuraavalla komennolla:

```
sudo opevpn --config oma_iot_laite.ovpn
```

Kopioimalla tiedosto /etc/openvpn -hakemistoon conf-päätteellä mahdollistaa automaattisen yhteyden muodostamisen. Asiakkaan automaattisen yhteydenmuodostuksen asetukset voi asettaa päälle asetustiedostossa: /etc/default/openvpn. Asetustiedostoon vaadittaan seuraavat muutokset:

```
AUTOSTART=”all” # Muutos alkuperäiseen nähden
OPTARGS=””
OMIT_SENDSIGS=0
```

Muutoksen jälkeen täytyy palvelu käynnistää vielä uudestaan. ”daemon-reload” on tarpeen ennen uudelleenkäynnistystä.

```
sudo systemctl daemon-reload
sudo service openvpn restart
```

OpenVPN-palvelun tilan voi katsoa Linux-palveluista sekä verkkoyhteyksistä komennoilla:

- sudo service openvpn status
  - Komento kertoo openvpn-palvelun tilan
- ifconfig
  - Pitäisi näkyä tun0-verkkoyhteys IP-osoitteella 10.8.0.x, käytettäessä esimerkin asetuksia