Tallinna Ülikool

Digitehnoloogiate Instituut

Piiratud funktsionaalsusega seadmete asjade interneti andmevahetusprotokollid

Bakalaureusetöö

Autor: Toomas Häide

Juhendaja: Jaagup Kippar

Autor:....., ... ,, 2016

Juhendaja:....., ... ,, 2016

Instituudi direktor:....., ... ,, 2016

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina _____ (sünnikuupäev: _____)

(*autori nimi*)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

_____

_____

_____

(*lõputöö pealkiri*)

mille juhendaja on _____,

(*juhendaja nimi*)

säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas/Haapsalus/Rakveres/Helsingis, _____

(digitaalne) allkiri ja kuupäev

Sisukord

Tähiste loetelu

AMQP	*Advanced Message Queuing Protocol*, Edasijõudnud Sõnumite Järjestamis protokoll.
Broker	Maakler, server kes vahendab sõnumeid.
CoAP	*Constrained Applications Protocol*, Piirangutega rakenduste protokoll
DDS	*Data Distribution Service*, Andmete jagamis teenus
IEEE	*Institute of Electrical and Electronics Engineers*, Elektri ja Elektroonika inseneride instituut
IETF	*Internet Engineering Task Force*, Interneti Edasiarendus Grupp
MQTT	*Message Queuing Telemetry Protocol*, Sõnumite Järjekorrastamis protokoll
P2P	*Peer-to-Peer*, kasutajalt-kasutajale
Pub/Sub	*Publish/Subscribe*, Avalda/Telli muster
QoS	*Quality of Service*, Teenuse kvaliteedi tase
SSL	*Secure Socket Layer*, Turvasokklite kiht
TCP	*Transmission Control Protocol*, edastusohje protokoll
UDP	*User Datagram Protocol*, kasutajadatagrammi protokoll
XMPP	*Extensible Messaging and Presence Protocol*, laiendatav sõnumite ja kohaloleku jälgimis protokoll

Sissejuhatus

Cisco IBSG (ingl Cisco Internet Business Solutions Group) vaatluste kohaselt tekkis Asjade Internet aastail 2008-2009, kui internetiga ühendatud seadmete arv ületas 12,5 miljardi piiri. Inimkonna rahvaarv kasvas samuti 6,8 miljardini. Suuresti tänu nutitelefonide revolutsioonile tähendas see seda, et iga inimese kohta planeedil maa oli sellel perioodil internetiga ühendatud 1.84 seadet. Cisco IBSG ennustuste kohaselt kasvab internetiga ühendatud seadmete arv aastaks 2020 viiekümne miljardini (Evans, 2011). Samaaegselt hakkab arvutite, eriti protsessorite, arengut piirama Moore'i seaduse lõpu lähenemine. Iga järgnev generatsioon ränipõhiseid protsessoreid toob endaga kaasa suuremad tootmiskulud ja väiksemad jõudluse kasvud. Samuti ei saa piirangutega seadmete effektiivsemaks, kiiremaks ja energiasäästlikumaks muutmisel rõhuda riistvarale, vaid tarkvarale mis selle peal jookseb.

Käesoleva bakalaureusetöö autori (edaspidi tekstis autor) huvi antud teema vastu tekkis kodu renoveerimise käigus kui autoril tekkis soov enda kodu automatiseerida. Tekkis vajadus välja selgitada kodu automatiseerimiseks sobilikud tehnoloogiad ja protokollid mis oleksid piisavalt robustsed, turvalised ja töökindlad.

Käesoleva bakalaureusetöö lahendatavaks probleemiks on see, et alustades piirangutega keskkonnale asjade interneti lahenduse loomist, pole välja kujunenud kindlat soovituslikku standardit, mille põhjal protokollide valik teha. Tavaliselt algab rakenduse arendamine asjakohase tehnoloogia välja selgitamisega, mis omakorda dikteerib kogu ülejäänud arendusprotsessi.

Töö käigus otsitakse vastuseid järgnevatele küsimustele:

- Missugused on relevantsemad asjade interneti andmevahetusprotokollid?
- Missugused on protokollid, mida tasuks kasutada piirangutega seadmetel?
- Missugused on protokollid, mida tasuks kasutada kodu automatiseerimisel?

Bakalaureusetöö koosneb kolmest peatükist. Töö esimeses peatükis tutvustatakse erinevaid asjade inteneti protokolle, võrreldakse nende omadusi ja otsitakse selliseid, mis sobiksid kasutada piirangutega seadmetel. Töö teises peatükis võrreldakse omavahel lähemalt kahte sobilikku protokolli, mille valik on tehtud esimese peatüki avastuste põhjal ja otsitakse sellist mida oleks otstarbekas kasutada kodu automatiseerimisel. Töö kolmandas peatükis luuakse

mõned näidisrakendused kasutades ühte sobilikku asjade interneti protokolli, et saada ülevaade selle implementeerimise keerukusest ja võimalustest.

1. Asjade interneti protokollid

Antud peatükis tutvustatakse määratletakse ära mis on piirangutega seadmed ja millised on tüüpilised piirangud. Samuti tutvustatakse erinevaid protokolle ja võrreldakse nende tugevusi ja nõrkusi.

1.1. Piirangutega seadmed

Kuna antud bakalaureusetöö võrdleb asjade interneti protokolle mis on mõeldud kasutamiseks piirangutega seadmetel, siis on tarvilik ära määratleda mis üldse on piirangutega seadmed. IETF definitsiooni kohaselt loetakse, antud bakalaureuse töö kirjutamise ajal, piirangutega sõlmeks sellist seadeldist mille puhul mõned, tihtipeale iseenesest mõistetavana võetavad karakteristikud, pole saadaval kas siis eelarve- ja/või füüsiliste piirangute tõttu. Füüsilised piirangud võivad olla siis näiteks suurus, kaal, saadaval olevad ressursid jõudluse ja elektrienergia näol. Tugevad riistvaralised piirid elektrienergiale, mäluhulgale ja protsesseerimisvõimsusele seavad paika ka ranged ülempiirid olekute hulgale, koodikogusele ja arvutustsükklitele. See omakorda muudab energia- ja ribalaiuse kasutamise optimiseerimise äärmiselt soosituks kõikides disainiaspektides (Bormann, Ersue, & Keranen, 2014).

Samuti jaotab IETF piirangutega seadmed tinglikult kolme klassi (Tabel 1).

Nimi	**Andmete suurus (nt. RAM)**	**Koodi suurus (nt. Välkmälu)**
Klass 0, C0	<< 10 KiB	<< 100 KiB
Klass 1, C1	~ 10 KiB	~ 100 KiB
Klass 2, C2	~ 50 KiB	~ 250 KiB

Tabel 1. Piirangutega seadmete klassid (Bormann, Ersue, & Keranen, 2014)

Antud bakalaureusetöös vaadeldakse protokolle, mis kuuluvad vähemalt klassi 2, 3 või enam. Kõrgemal kui klassi 2 kuuluvate seadmete puhul võib piiranguks lugeda näiteks saadaval oleva elektrienergia hulga või aku laadimistsükklite arvu.

1.2. Protokollide tutvustus

Järgnevalt vaadeldakse mõningaid asjade interneti rakenduskihi suhtlusprotokolle. Protokollide valik on tehtud IEEE CS (Ingl *Institute of Electrical and Electronics Engineers Computer Society*) poolt avaldatud uuringu tulemuste põhjal mille eesmärgiks oli anda ülevaade tähtsamatest asjade interneti protokollidest ja tehnoloogiatest (Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015). Mõned neist loodud väga komplekseid ülesandeid täitma ja on väga spetsiifiliselt kohandatavad. Teised on olulisemalt kergekaalulisemad ja sobivad ka piirangutega seadmetele. Kuna nad kõik on asjade interneti protokollid oleks viisakas neid kõiki lühidalt tutvustada ning nende nõrkuseid ja tugevusi otsida. Järgnevas tabelis on lühidalt kirjeldatud tähtsamate rakenduskihil jooksvate asjade interneti protokollide omadused (Tabel 2).

	DDS	**MQTT**	**AMQP**	**REST/HTTP**	**CoAP**	**XMPP**
Suhtlus abstraktsioon	Pub/Sub	Pub/Sub	P2P või Pub/Sub	Päring/Vastus	Päring/Vastus	P2P või Pub/Sub
Implementeeritud arhitektuur	Globaalne andmeruum	Maakler (kõige tavalisem)	Maakler (kõige tavalisem)	Klient-Server	Klient-Server	XMPP Server (Maakler)
Kasutaja poolt konfigureeritavad kvaliteeditasemed (QoS)	22	3	3	Mitte ühtegi	Kinnitusega ja kinnituseta sõnumid	Mitte ühtegi
Transpordi protokoll	Vaikimisi UDP aga teisi protokolle on samuti võimalik valida, nt. TCP	TCP	TCP	TCP	UDP	TCP
Litsentsid	Avatud lähtekoodiga & Kommertslitsents	Avatud lähtekoodiga & Kommertslitsents	Avatud lähtekoodiga & Kommertslitsents	HTTP on saadaval tasuta enamikel platvormidel	Avatud lähtekoodiga & Kommertslitsents	Avatud lähtekoodiga & Kommertslitsents
Turvalisus	Implementatsioonipõhine, enamasti SSL või TLS	Lihtne Kasutajanime/Parooli mudel, SSL andmete krüpteerimiseks	SASL autentimine, TLS andmete krüpteerimiseks	Enamasti SSL või TLS põhinev	DTLS	TLS ja SASL

Tabel 2. Erinevate protokollide põhiomaduste võrdlus (Foster, 2015)

1.2.1. DDS

DDS spetsifikatsioon (OMG - Object Management Group, 2015) kirjeldab andmetepõhist avalda/telli (ingl *Publish/Subscribe*) mudelit hajussüsteemide omavahelise suhtluse aretamiseks ja sidumiseks. DDS on rajatud globaalse andmeruumi (ingl *Global data space*) põhimõttele. See globaalne andmeruum on kättesaadav kõigile huvitatud rakendustele. Rakendused mis tahavad andmete kogumisest ja jagamisest osa võtta peavad väljendama enda soovi hakata avaldajateks (ingl *Publisher*). Rakendused mis soovivad saada ligipääsu andmeosadele peavad väljendama enda soovi hakata tellijateks (ingl *Subscriber*). Iga kord kui avaldaja postitab uusi andmeid ettemääratud andmeruumi saadetakse need andmed vahevara (ingl *Middleware*) kasutades tellijatele. DCSP (ingl *Data-Centric Publish-Subscribe*) mudelit kasutavad rakendused, antud juhul DDS protokolli kasutavad rakendused, eeldavad et DCPS implementatsioon pakub neile suurt jõudlust, ettearvatavust ja effektiivset ressurside kasutamist. Vajadus skaleerida sadu või tuhandeid avaldajaid ja tellijaid robustselt on samuti tähtis nõue. See pole lihtsalt skaleerimisnõue vaid ka paindlikkusnõue: paljudel süsteemidel, mis on rajatud DDS'ile, puudub vajadus/võimalus tervet süsteemi ümber ehitada kui võrku lisatakse mõni uus rakendus. Andmetepõhine suhtlussüsteem eraldab saatjad tellijatest; Mida vähem seotud omavahel on avaldajad ja tellijad, seda kergemaks sellised laiendused muutuvad. DDS sõltub sõnumite saatmis kvaliteeditaseme kasutamisest (ingl *Quality of Service*). DDS kirjeldab 22 erinevat QoS poliisi, mille üles loetlemine ja kirjeldamine on käesoleva bakalaureusetöö uurimisulatusest väljas. DDS raamistiku eesmärk, kokkuvõtlikult, on pakkuda effektiivset ja robustset informatsioonivahetussüsteemi mis vahendab õiget informatsiooni õigesse kohta õigel ajal. RTI pakub üheks DDS'i kasutuskohaks meditsiinivaldkonda, et näiteks jälgida haiglasiseselt patsientide tervislikku seisundit (RTI International, kuupäev puudub).

1.2.2. HTTP

HTTP protokoll on sama vana kui WWW ise, see on väga stabiilne protokoll. Igal suuremal programmeerimis keelel on oma HTTP teek. Alates HTTP/2 versioonist, mida iga suurem netibrauser ka tänaseks toetab, on HTTP/2 binaarkeelne, ehk ta ei ole enam inimloetav, aga sellega võidetakse ribalaiust (ingl *bandwidth*). Samuti toetab HTTP/2 aktiivsuhtlust (push notifications). Lisaks sellele on HTTP/2 päis tihendatud, mis samuti vähendab saadetava info hulka (Undertow, 2015). Kõik need omadused muudavad HTTP üheks kandidaatiks asjade interneti seadmetel kasutamiseks.

1.2.3. XMPP

XMPP protokolli ajalugu ulatub aastasse 1999, kui Jeremie Miller kuulutas Jabberi olemasolust. Jabberist kasvas hiljem välja XMPP (XMPP, kuupäev puudub). XMPP on suurtäht lühend nimest *Extensible Messaging and Presence Protocol* - kogum avatud tehnoloogiaid kiirsuhtluse, kohaloleku, mitme osapoolega jututoa, hääl- ja videosuhtluse, koostöö, kergekaalulise vahevara, sisuhalduse ja üldise XML andmevahetuse korraldamise tarbeks. XMPP puhul tasub ära mainida, et tegemist on tasuta, avaliku ja kergesti mõistetava protokolliga. Protokoll on IETF'i poolt kuulutatud standardiks. Samuti on XMPP leidnud laialdaselt kasutust miljonite inimeste poolt jututubade näol (XMPP, kuupäev puudub). Bakalaureuse töö kirjutamise hetkel puudub XMPP protokollis sõnumite kohale toimetamisteenuse kvaliteeditaseme reguleerimine (ingl Quality of Service). Kuigi Bakalaureusetöö kirjutamise ajal on esitatud ettepanek lisada antud funktsionaalsus ka XMPP protokollile (Waher, 2015).

1.2.4. CoAP

CoAP (ingl *Constrained Application Protocol*) tähendaks eesti keelde tõlgituna piirangutega rakenduste protokolli, mis üritab kirjeldada fakti, et tegemist on spetsialiseeritud veebisuhtlusprotokolliga mis on mõeldud kasutamiseks seadmetel, millel on piiratud ressursid nii riistvara- kui ka internetiühenduse näol. Riistvara millele CoAP on disainitud tihtipeale omab piiratud ROM (ingl *Read-Only Memory*) ja RAM (ingl *Random Access Memory*) mälu, samaaegselt võttes arvesse fakti, et piiratud võrkudel, nagu näiteks IPv6 vähese energiatarbega traadita personaalne kohtvõrk (ingl 6LoWPAN), on tihtipeale kõrge paketi veamäär ja tüüpiline läbilaskevõime (ingl *throughput*) ulatub kümnetesse kilobittidesse sekundis. Antud protokoll on disainitud Masin-Masin tüüpi rakenduste tarbeks nagu näiteks energianutivõrgud või hoonete automaatika seadmete juhtimiseks. CoAP pakub omalt poolt päring/vastus (ingl Request/Reply) interaktsioonimudelit rakenduste lõppsõlmede vahel, toetab teenuste ja ressursside avastamist ja toetab veebi põhimõisteid nagu näiteks URI ja interneti meediatüübid. CoAP on disainitud nii et seda oleks kergesti võimalik siduda HTTP protokolliga samaaegselt täites väga spetsiiflisi nõudeid nagu näiteks multiedastustugi, väga madalad talituskulud ja lihtsus, piiratud keskkondades kasutamiseks (Shelby, Hartke, & Bormann, 2014).

1.2.5. AMQP

AQMP (ingl *Advanced Message Queuing Protocol*) on avalik interneti andmevahetusprotokoll ärisõnumite saatmiseks. AQMP defineerib juhtmetaseme protokolli, mis tagab töökindla ärisõnumite vahetamise kahe huvitatud osapoole vahel. AQMP koosneb mitmest kihist. Kõige

madalam neist pakub effektiivset, binaarset, P2P protokolli sõnumite vahendamiseks kahe protsessi vahel mingisuguses võrgus. Selle peal paikneb sõnumite saatmiskiht, mis defineerib konkreetse ja standardse kodeeringuga abstraktse sõnumiformaadi. Iga reeglikuulekas AMQP protsess peab olema võimeline saatma ja vastu võtma sõnumeid standardses kodeeringus (OASIS - Organization for the Advancement of Structured Information Standards, 2012). AMQP on bakalaureusetöö kirjutamise ajal laialdaselt kasutatud. Üks märkimisväärne AMQP protokolli kasutaja on JPMorgan, kes saadab päevas miljard sõnumit kasutades AMQP'd ja kasutab seda umbes tosinas missioonikriitilises süsteemis üle maailmselt. Samuti leiab AMQP kasutust Deutsche Börse, NASA, Red Hat ja veel paljude teiste suurkorporatsioonide poolt (AMQP Working Group, kuupäev puudub).

1.2.6. MQTT

OASIS standardi kohaselt (Banks & Gupta, 2015) on MQTT (ingl *Message Queuing Telemetry Transport*) klient-maakler (ingl *Client/Broker*) tüüpi avalda/telli (ingl *Publish/Subscribe*) sõnumite transpordiprotokoll. Protokoll on kergekaaluline, avalik, lihtne ja disaitud lihtsasti implementeeritavaks. Need omadused muudavad MQTT protokolli ideaalseks kandidaadiks mitmel pool kasutamiseks, kaasaarvatud piiratud ressurssidega võrkudes, Masin-Masin suhtluse puhul ja paljude teiste asjade interneti rakenduste kontekstis, kus väike koodijälg on üheks nõudmiseks ja ribalaius on luksus. Protokoll põhineb TCP/IP komplektil või mõnel muul võrguprotokollil mis tagab järjestatud, kadudetta, kahesuunalise võrguliikluse. Protokolli tunnused on:

- Avalda/Telli sõnummudeli kasutamine, mis tagab üks-mitmele sõnumite jaotuse ja rakenduste eraldatuse üksteisest.
- Sõnumite transportteenus, mis ei sõltu saadetava sõnumi sisust.
- Kolm sõnumite kohale toimetamise (ingl *Quality of Service*) taset:
    - "Kõige rohkem üks kord" kus sõnum saadetakse välja ainult üks kord. Sõltuvalt ühenduse kvaliteedist võib mõni sõnum vahele jääda. Võib kasutada näiteks keskkonna sensorite andmete saatmiseks mille puhul pole üksiksõnumite kohale jõudmine kriitilise tähtsusega
    - "Vähemalt üks kord" mille puhul on garanteeritud, et sõnum jõuab kohale, aga võib esineda duplikaate.
    - "Täpselt üks kord" mis tagab selle, et sõnumid jõuavad kohale täpselt ühe korra. Sellist varianti on võimalik kasutada näiteks maksesüsteemide puhul mis tagab selle, et arve esitatakse vaid üks kord.

- Väiksemahuline transpordikiht, et minimiseerida võrguliiklust.
- Mehanism mis annab huvitatud osapooltele märku kui peaks toimuma mittetahtlik võrgukatkestus.

1.3. Protokollide hüpoteetiline lühivõrdlus

Lühidalt kokku võttes, kaks protokolli mida autor soovitab kasutada piirangutega võrkudes ja seadmetel oleks CoAP ja MQTT kuna mõlemad on üsna lihtsasti mõistetavad ja implementeeritavad erinevatel platvormidel.

CoAP protokolliga seotud implementatsioonid on üsna lihtsasti arusaadavad ka algajatele ja lihtsama näite, LED tulekese juhtimine üle kohaliku võrgu, saab samuti tööle minutitega ja ilma suuremate probleemidetta.

MQTT on disainitud kasutamiseks paljudel, väikestel, piirangutega seadmetel mis saadavad lühikesi sõnumeid väikese ribalaiusega võrkudel.

AMQP puhul on aga tegemist protokolliga, mis on loodud täitma kõikvõimalikke sõnumite saatmisstsenaariumeid, mida viimase 25 aasta jooksul on nähtud (Cohn, kuupäev puudub). Sellest tulenevalt ei soovita autor kasutada ka AMQP protokolli lihtsamate stsenaariumite puhul nagu seda on kodu automatiseerimine, kuna pakutavad lisavõimalused lisavad liialt palju keerukust. Samuti pole antud protokoll mõeldud kasutamiseks piirangutega võrkudes või seadmetel.

Kui on aga vajadus veakindla ja väga paljusid spetsiifilsi juhtumeid käsitleva protokolli järele, siis on selleks sobilikeim DDS. DDS protokoll on ainukene protokoll antud bakalaureuse töös, mis suudab täita range reaalaja rakenduse (ingl *Hard Real-Time*) nõudmisi mida nõuavad näiteks südamestimulaatorid ja kriitilise tähtsusega rakendused. DDS protokolli miinuseks võib aga lugeda just selle paigaldamiskeerukuse. Samuti pole antud protokollile palju avatud lähtekoodiga implementatsioone.

Võib tekkida küsimus, miks mitte kasutada HTTP protokolli asjade interneti lahenduste loomisel? HTTP miinuseks piirangutega seadmetel kasutamisel võib lugeda, et see disaniti eeldusega, et serveriga ühendatud klient jääb serveriga ühendatuks kuni sessiooni lõpuni. Mis tähendab, et kui kasutada HTTP sõnumite saatmis protokollina jääks ühendus serveriga igaveseks lahti. Lisaks praegusel hetkel puuduvad HTTP puhul ka kvaliteeditasemed, nagu MQTT puhul. HTTP/2 on veel üsna värske, võibolla pööratakse antud murekohtadele tulevikus

tähelepanu, aga bakalaureusetöö kirjutamise hetkel on veel liiga vara öelda (Drager, 2015).Lisafunktsionaalsusena ei oma HTTP võrreldes CoAP protokolliga, sisseehitatud ressursside avastamist.

1.4. Protokollide jõudlustestid

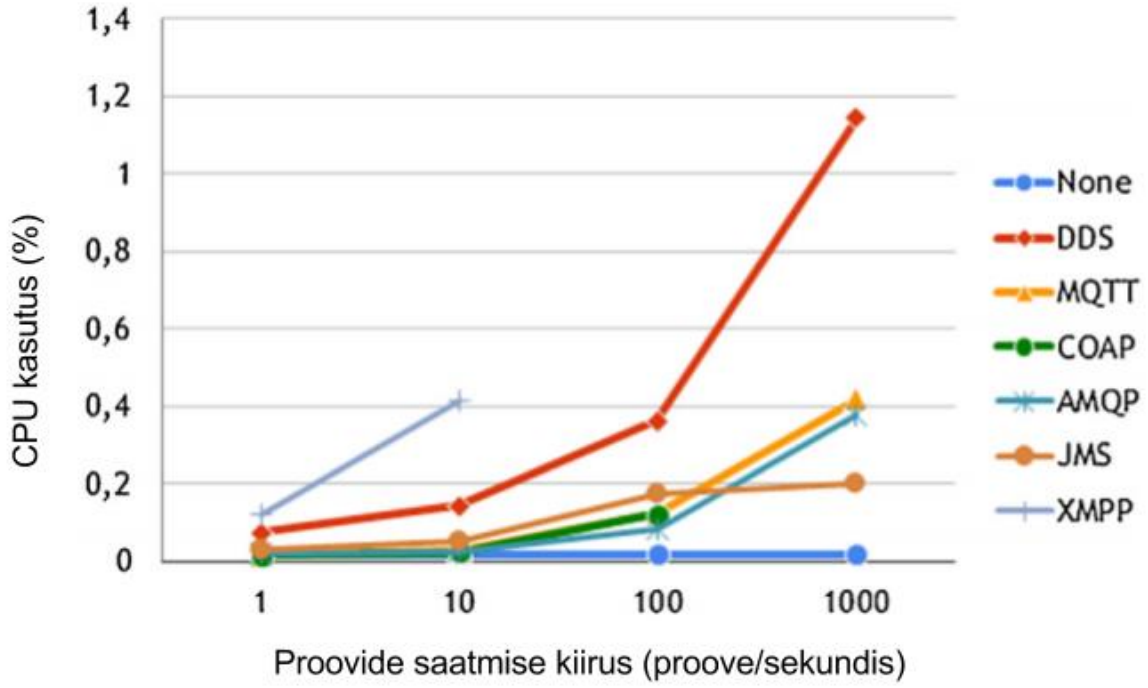
Autori seniseid hüpoteese kinnitab 2016 aasta märtsis avalikustatud uuring asjade interneti protokollide kohta (Barroso, Valderrama, Roa, Tosin, & Ruiz, 2016). Uuringu eesmärgiks oli luua tööriist mis võimaldaks sooritada, analüüsi eesmärgil, jõudlusteste erinevatel asjade interneti suhtlusprotokollidel, et objektiivselt valida kõige sobilikum protokoll meditsiiniliste rakenduste tarbeks. Võrreldi DDS, MQTT, CoAP, JMS, AMQP ja XMPP protokolle erinevate näitajate põhjal sealhulgas protsessori kasutus, mälu kasutus, ribalaiuse kasutus, latentsiaeg ja signaali kõikuvus.

Jõudlustestid viidi läbi laboratoorsetes, isoleeritud tingimustes. Kõikide masinate kellad sünkroniseeriti omavahel kasutades NTP (ingl *Network Time Protocol*) protokolli. Testide läbi viimisel kasutati 41 alluvmasinat (ingl *slave*) ja ühte ülemmasinat (ingl *master*). Nendest 20 olid avaldajad (ingl *publishers*) ja 20 tellijad (ingl *subscribers*) ja ühte ülemmasinat kasutati kui maaklerit või serverit tsentraalsete protokollide puhul. Testi tulemusi verifitseeriti kasutades virtuaalmasinatel sarnast seadistust ja ei avastatud mitte ühtegi märkimisväärset erinevust võrreldes eelneva testi tulemustega.

1.4.1. Jõudlustestide tulemused

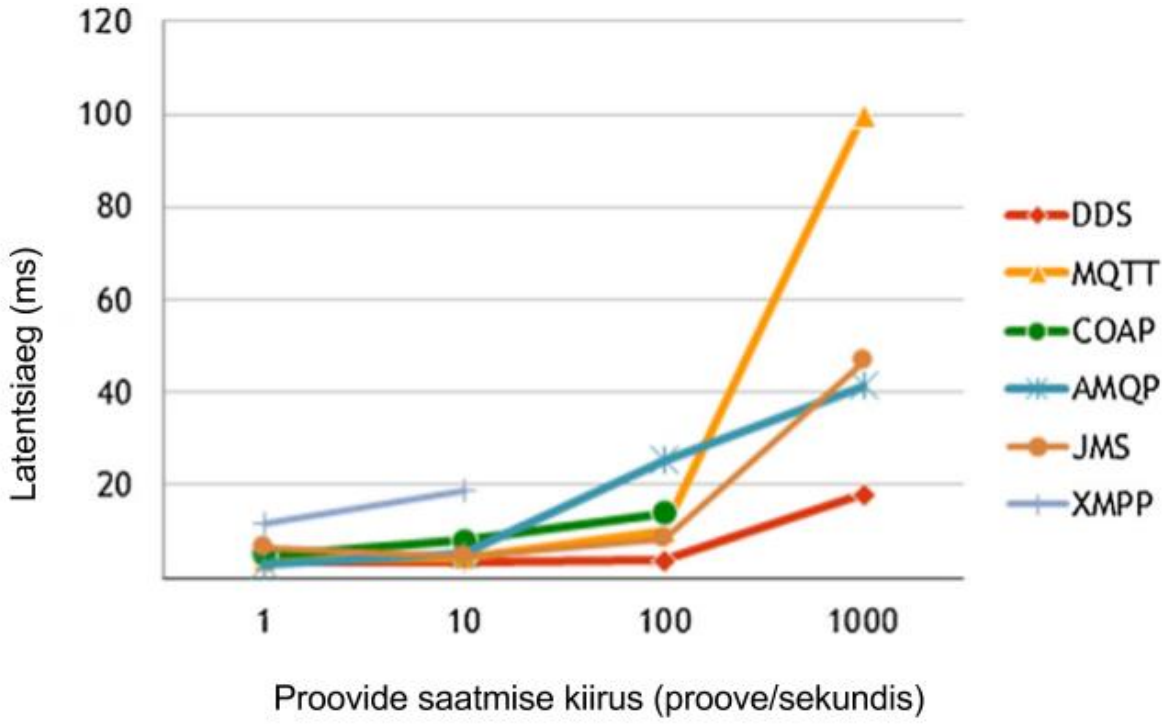
Antud jõudlustestide tulemusi vaadeldes võrreldakse omavahel ainult protokolle millest on eelnevalt bakalaureusetöös juttu tulnud.

Protsessori kasutust mõõdeti protsentides ja kõige rohkem protsessorit kasutav protokoll, 1000 sõnumi sekundis vastu võtmisel, oli DDS ja kõige vähem AMQP (Joonis 1).



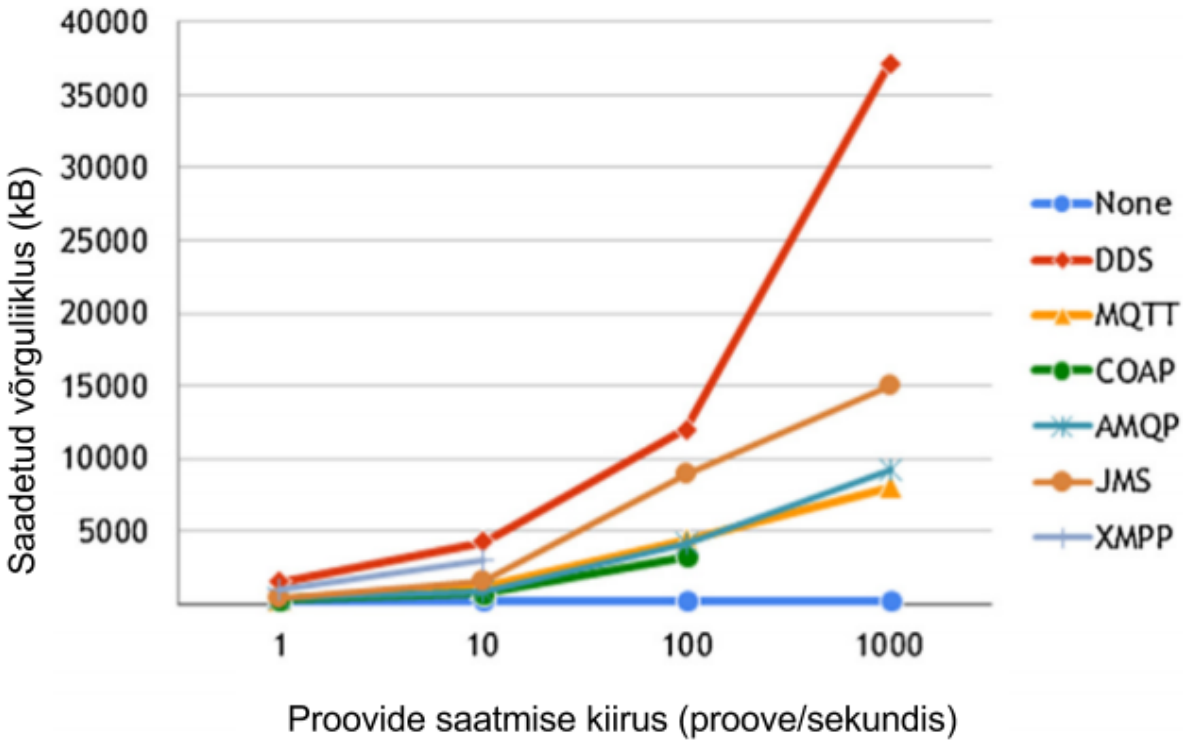
Joonis 1. Tellijate protsessori kasutus (Barroso, Valderrama, Roa, Tosin, & Ruiz, 2016)

Tellijate latentsiaegu (ingl *latency*) võrreldes, 1000 sõnumi sekundis vastu võtmisel, saavutas kõige parema tulemuse DDS protokoll ja kõige suurema latentsiajaga oli MQTT (Joonis 2).



Joonis 2. Tellijate latentsiaeg 1000 sõnumi vastu võtmisel (Barroso, Valderrama, Roa, Tosin, & Ruiz, 2016)

Tellijate ja Avaldajate poolt saadeti kõige rohkem võrguliiklust (ingl *network traffic*) DDS protokolli puhul ja kõige vähem MQTT protokolli kasutades (Joonis 3).



Joonis 3. Avaldajate ja tellijate poolt saadetud võrguliiklus (Barroso, Valderrama, Roa, Tosin, & Ruiz, 2016)

1.4.2. Protokollide tugevused ja nõrkused

Eelpool mainitud uuringu (Barroso, Valderrama, Roa, Tosin, & Ruiz, 2016) tulemused on kontsentreeritud kujul ära toodud järgnevas tabelis (Tabel 3).

Protokoll	Tugevused	Nõrkused
DDS	• Kõrge töökindlus. • Rikkalik valik QoS tasemeid. • Suur skaleeruvus. • Sobilik rangetele reaalajarakendustele. • Vigade tolereerimine. • Koostalitlusvõime. • Puuduvad Maaklerid, kes võiksid olla andmete saatmisel kitsaskohtad • Automaatne ressursside avastamine. • Andme struktuur ja teema on defineeritud kasutaja poolt. • Avaldaja ja tellija eraldatud üksteisest.	• Originaalselt arendati ainult isoleeritud kohtvõrkudes kasutamiseks. • Suur mälukasutus. • Piirangutega seadmetel kehv valik avatud lähtekoodiga teeke. • Arendus ja konfigureerimis keerukus võrreldes teiste protokollidega.

MQTT	• Lihtne. • Kergekaaluline: Madal protsessori- ja mälukasutus. • Lühike latentsiaeg väikeste andmehulkade puhul. • WebSocketite tugi • Laiem hulk implementatsioone manusseadmetele ja mobiilsetele platvormidele. • Avaldaja ja tellija eraldatud üksteisest. • Aktiivne arendajate kommuun.	• Pikk latentsiaeg suure proovisageduse puhul. • Ei paku automaatset seadmete avastamist. • Andmetüüp on liiga lihtne. • Ei paku piisavalt turvalisust protokolli tasemel. • Ei sobi reaalajarakendustele kus proovisagedus on suur.
CoAP	• Lihtne. • Põhineb robustsel HTTP protokollil. • Kergekaaluline: madal protsessori- ja mälukasutus. • Madal ribalaiuse tarbimine. • Ressursside avastamine. • IP multisaade (ingl *Multicasting*). • Klient/Server mudeli tugi.	• Pikk latentsiaeg. • Ei sobi reaalaja rakendustele. • SSL/TLS ei ole saadaval. • Puudub tugi tööriistadele ja teekidele. • Andmetüüp on liiga lihtne. • Kehv QoS tugi.
AMQP	• Väga laiendatud. • Kõrge koostalitlusvõime erinevate edasimüüjate vahel. • Disainitud toetama kriitilisi rakendusi finantssektoris.	• Koostalitlusvõime pole alati tagatud. • Ei sobi reaalajarakenduste loomiseks. • Ei paku automaatset ressursside avastamist. • Piirangutega seadmetele pole piisavalt avatud lähtekoodiga teeke.
XMPP	• Lihtne. • Ühtne ühendus erinevate kasutajate ja serverite vahel. • Laialdaselt toetatud erinevate firmade ja avatud lähtekoodiga projektide poolt. • Laiendatav ja paindlik erinevate rakenduste tüüpide tarbeks. • On defineeritud protokolli laiend seadmete avastamiseks.	• Kõrge protsessori kasutus. • Suur ribalaiuse tarbmine. • Kehv veakindlus. • Ei sobi reaalaajarakenduste loomiseks. • Andmetüüp on liialt lihtne. • Puudub QoS tugi.

Tabel 3. Iga protokolli tugevused ja nõrkused (Barroso, Valderrama, Roa, Tosin, & Ruiz, 2016)

2. CoAP vs MQTT

Kõike eelnevalt bakalaureuse töös käsitletut analüüsides jäi autori hinnangul kodu automatiseerimise tarbeks sobilikuks valikuks veel 2 protokolli: CoAP ja MQTT. Otsustamisel vaadeldi protokollide paigaldamise lihtsust, avatud lähtekoodiga implementatsioonide olemasolu ja sobilikkust piirangutega seadmetel kasutamiseks. Järgnevas peatükis võrreldaksegi lähemalt kahte sobilikku asjade interneti protokolli, et selgitada välja protokoll millega bakalaureusetöö kolmandas peatükis luua näidisrakendus. Järgnevalt vaadeldakse protokollide puhul populaarsemate programmeerimiskeelte implementatsioonide olemasolu. Samuti on otsitud reaalseid kasutuskohti, kus kumbki protokoll on kasutust leidnud. Antud peatükis selgitatakse ka mõningaid protokollidega seotud tööpõhimõtteid.

2.1. TIOBE Populaarsemad programmeerimiskeeled

TIOBE indeks on populaarsemate programmeerimiskeelte indeks mis arvutab programmeerimiskeelte populaarsust kasutades suuremaid otsingumootoreid nagu näiteks Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube ja Baidu (TIOBE software BV, 2016).

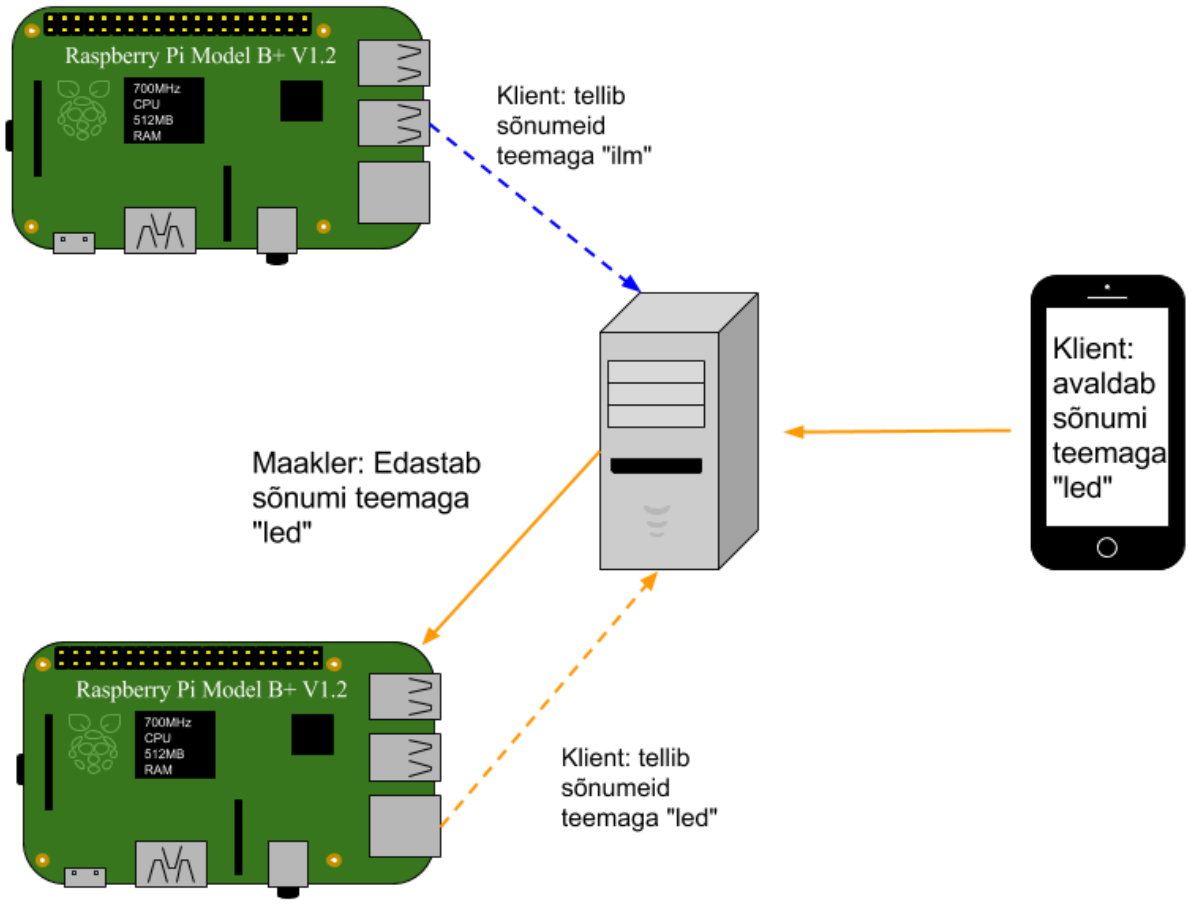
Autor pidas vajalikuks üles loedelda esimesed 5 populaarsemat programmeerimiskeelt, et võrrelda CoAP ja MQTT protokollide erinevate implementatsioonide olemasolu (Tabel 4).

Programmeerimiskeel	**Positsioon tabelis**
Java	1
C	2
C++	3
C#	4
Python	5

Tabel 4. Populaarsemad programmeerimiskeeled seisuga Aprill 2016 (TIOBE software BV, 2016)

2.2. Avalda/Telli Muster

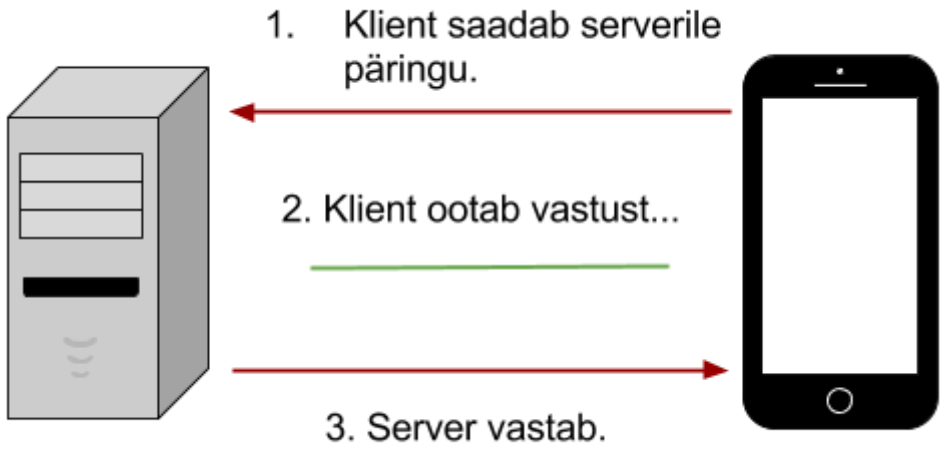
Avalda/Telli muster (ingl *Publish/Subscribe pattern*) pakub alternatiivset lahendust tavapärasele klient-server lahendusele, kus klient suhtleb otse lõppsõlmega (ingl *endpoint*). Avalda/Telli lahendus eraldab üksteisest kliendid, kasutades vahelülina maaklerit (ingl *broker*), kes edastab klientide saadetud sõnumid kõikidele huvilistele (Joonis 4). See tähendab, et sõnumi saatja klient ja sõnumi saaja klient ei tea üksteise olemasolust mitte midagi, kogu suhtlust käib läbi maakleri. Maakler sorteerib kõik sissetulevad sõnumid ja jagab need laiali vastavalt tellimiseelistustele (HiveMQ Team, 2015). Antud suhtlusmudelit kasutab MQTT. Antud mudeli eeliseks on see, et klientide võrku juurde lisamine on väga lihtne, kuna mitte ükski klient ei sõltu teisest kliendist: mooduleid võib muretult juurde lisada või ära võtta. Avalda/Telli mustri puudus on aga see, et maaklerist võib saada suuremate lahenduste puhul nii-öelda pudelikael, mis suudab vaid nii palju informatsiooni korraga laiali jagada kui võimas on ta riistvara.



Joonis 4. Avalda/Telli muster

2.3. Päring/Vastus muster

Päring/Vastus (ingl *Request/Reply*) muster on üks kõige lihtsamaid mooduseid veebiserveriga suhtlemiseks (Joonis 5). Klient küsib serveri käest mõnda ressurssi ja jääb ootama kuni server talle vastab (wcftutorial.net, 2014). Antud suhtlusmudelit kasutab ka CoAP protokoll. Antud mudeli eeliseks on see, et ühendust serveriga ei hoita pidevalt lahti ja sel viisil säästetakse piirangutega seadmetel väärtuslikku elektrienergiat.



Joonis 5. Päring/Vastus mudel

2.4. CoAP protokolli implementatsioonid

Järgnevalt vaadeldakse mõningaid CoAP protokolli implementatsioone populaarsemates programmeerimiskeeltes. Üheks kriteeriumiks mille põhjal antud nimekiri koostati oli see, et need oleksid aktiivselt arenduses – viimane uuendus teegile on toimunud viimase aasta jooksul.

2.4.1. Java

CoAP protokolli kõige märkimisväärsem implementatsioon on Java programmeerimiskeeles loodud Californium (Eclipse Foundation, 2015). Californium (edaspidi Cf) on mõeldud kasutamiseks piiramatta võrkudes näiteks taustaserverite (ingl *Backend server*) infrastruktuurides (vaheserverid, pilveteenused) ja vähepiiratud seadmetel mis on näiteks võimelised jooksutama mõnda linuxi distributsiooni (nutikad tehase- ja koduseadmed, mobiilvõrgulüüsid). Cf implementatsioon programmeeriti vastavalt IETF'i CoAP standardile ja hiljuti kirjutati nullist ümber, peale täieliku vajaliku kogemuse omandamist. Cf on nüüdseks keskendunud skaleeruvatele teenustele suuremahulistes asjade interneti rakendustes.

2.4.2. C

Libcoap (Bergmann, 2015) on C keele implementatsioon kergekaalulisest protokollist seadmetele mille arvutusjõudlus, RF raadius, mälu, ribalaius või netipaketi suurused on piiratud. Tegemist on ka avatud lähtekoodiga tarkvaraga.

2.4.3. C++

CantCoap (Mills, 2015) on CoAP protokolli implementatsioon mis keskendub lihtsusele. Antud teek pakub ainult protokolli andmeühiku (ingl *Protocol Data Unit*) loomist ja hävitamist. On eeldatud, et kasutaja ise tegeleb sõnumite taasedastamise, aegumise ja sõnumite identifikaatorite võrdlemise ja sobitamisega. Mis raamistiku loojate sõnul ei tohiks olla nii tülikas kui kõlab ja peaks piiratud ressurssidega seadmetel olema loogiline.

2.4.4. C#

CoAP.NET (SmeshLink Technology Co., 2015) on CoAP protokolli implementatsioon programmeerimiskeeles C# mis pakub tuge .NET rakendustele ja põhineb suuresti Californiumi implementatsioonil. Toetatud võimalused on:

- CoAP Implementatsioon vastavalt RFC 7252 väljapakutud standardidokumendile.
- Suurte failide saatmist blokkideks jaotatuna.
- Ressursside vaatlemist (ingl *resource monitoring*)

2.4.5. Python

Aiocoap (Wasilak & Amsüss, 2014) on CoAP protokolli implementeeriv Python 3.4 kasutav teek. Python 3.4’st kasutab antud teek asyncio moodulit. Aiocoap baseerub txThings implementatsioonil mis toetab vanemaid Pythoni versioone. Kui ei ole võimalik Python 3 kasutada, siis tasub mõelda txThings raamistiku kasutamisele.

2.5. MQTT Klient

Tulenevalt MQTT protokolli eripäradest tuleb MQTT puhul vaadelda kahte tüüpi implementatsioonide olemasolu, klient ja maakler. Sarnaselt CoAP implementatsioonidele jälgiti seda, et teek oleks aktiivselt arenduses.

MQTT Kliendi implementatsioonidest pakub kõige mitmekülgsemat lahendust Eclipse Paho Projekt (Eclipse Foundation, kuupäev puudub). Eclipse paho projekt pakub Kliendi implementatsioone paljudes erinevates programmeerimiskeeltes, sealhulgas Java, C, C++, C# ja Python.

2.5.1. Java

Eclipse paho pakub Java platvormile kahte kliendi implementatsiooni Android Service ja J2SE.

Android Service (Eclipse Foundation, kuupäev puudub) kujutab endast liidest Paho Java MQTT teeki kasutavale rakendusele mis võimaldab Androidi seadmel saata- ja vastu võtta MQTT sõnumeid isegi siis kui rakendus ise ei ole käivitatud, kasutades androidi taustal jooksvat teenust. Paho Android Service pakub asünkroonset rakenduse programmeerimisliidest (ingl *Application Programming Interface*).

J2SE (Eclipse Foundation, kuupäev puudub) Kujutab endast standardse Java plavormi implementatsiooni MQTT protokollist mis on mõeldud rakenduste arendamiseks mis jooksevad JVM'il (ingl *Java Virtual Machine*) või mõnel muul Javaga ühenduval platvormil nagu näiteks Android.

2.5.2. C/C++

Eclipse paho projektis on samuti kaks MQTT C keele implementatsiooni (Eclipse Foundation, kuupäev puudub) C Posix ja Windows süsteemidele ja C/C++ manussüsteemidele (ingl *embedded systems*). Lisaks MQTT-SN implementatsiooni C ja C++ keeles manussüsteemidele.

2.5.3. C#

M2Mqtt (Patierno, M2Mqtt, kuupäev puudub) on MQTT klient mis on saadaval kõikidele .Net platvormidele ja WinRT platvormidele.

2.5.4. Python

Paho Python klient (Eclipse Foundation, kuupäev puudub) pakub kliendi klassi mis toetab MQTT v3.1 ja v3.1.1 Python 2.7 või 3.x implementatsioone. Samuti pakub Pythoni klient abifunktsioone mis peaks sõnumite avaldamise MQTT serverile väga arusaadavaks protsessiks.

2.6. MQTT maakler

MQTT maaklerite puhul on implementatsioonid programmeerimiskeeleti rohkem killustunud kui klientide puhul.

2.6.1. Java

Moquette (Selva, 2016) on Java programmeerimiskeeles kirjutatud kergekaaluline implementatsioon MQTT protokolli maaklerist. Bakalaureuse töö kirjutamise ajal aktiivselt arenduses ja kasutatud ka Freedomotic kodu automatiseerimisraamistikus.

2.6.2. C/C++

Mosquitto (Eclipse Foundation, 2016) on C keeles programmeeritud kergekaaluline MQTT Maakleri implementatsioon. Maakler on kirjutatud C keeles, et võimaldada jooksutamist masinatel millel pole isegi võimekust jooksutada JVM-i. Kuna sarnaselt sensoritele ja aktuaatoritele, võivad manussüsteemid kes sõnumeid vahendavad olla väikesed ja piiratud jõudlusega. Lisaks sellele et Mosquitto suudab vastu võtta sõnumeid MQTT kliendi instantsidelt on Mosquittol olemas sild (ingl *bridge*) mis lubab sellel ühendada end teiste MQTT serveritega, sealhulgas Mosquitto instantsidega. See lubab ehitada MQTT serverite võrgustikke kes toimetavad sõnumeid ühest sõlmest teise, olenevalt sellest kuidas sillad on konfigureeritud.

2.6.3. C#

GnatMQ (Patierno, GnatMQ, 2015) on projekt mis loodi et arendada MQTT maakler mis põhineks .Net raamistikul. Projekt on loodud et kaasata ja õpetada inimesi loomaks kasulikke resursse MQTT protokollile, kooskõlas M2Mqtt kliendi projektiga, mida sai eelpool mainitud.

2.6.4. Python

Python implementatsioonidest on saadaval HBMQTT (Beer Factory, 2016). Avatud lähtekoodiga MQTT kliendi ja maakleri implementatsioon. Rajatud asyncio teegile, pakub HBMQTT lihtsasti arusaadavat rakenduste programmeerimisliidest, mis muudab asünkroonselt töötavate programmide kirjutamise lihtsaks. Lisaks sellele implementeerib HBMQTT kogu MQTT 3.1.1 protokolli standardi.

2.7. MQTT produktsioonis

Üks tuntumaid ja laialt kasutuses olevatest MQTT implementatsioonidest pärineb Facebookilt. Facebook kasutab MQTT protokolli alates 2011 aasta 9 augustist. Sel päeval avalikustas Facebook enda Facebook Messenger rakenduse, mis võimaldab inimestel saata sõnumeid üks-ühele või grupivestluses sõpradele. Paar nädalat enne uue Messengeri avalikustamist oli inseneridel mitmeid probleeme. Üks nendest oli sõnumite saatmisel suur latentsisaeg. Meetod mida kasutati oli küll töökindel aga aeglane ja selle parendamine oli limiteeritud võimalustega. Lõpuks ehitasid insernerid uue mehanismi mis hoidis pidevat ühendust Facebooki serveritega. Et teha seda effektiivselt, ilma akusid üleliia kulutamata, võeti kasutusele MQTT protokoll millega oli varasemalt kokkupuututud Beluga platvormi arendamisel. Hoides pidevat MQTT ühendust Facebooki serveritega ja suunates need läbi suhtluskonveieri (ingl *chat pipeline*), saavutati latentsiaeg, telefonist-telefoni sõnumi saatmisel, paarsada millisekundit mis on tunduvalt parem kui varasemalt saavutatud paar sekundit (Zhang, 2011). Facebook oli sunnitud implementeerima operatsioonisüsteemipõhise tarkvara, mis suhtleks eksisteerivate lõppsõlmedega. Facebook Messengeri disainimisel üheks võtmekohaks oli kasutajaliides: soov taaskasutada nii palju koodi kui võimalik ja nii palju kui võimalik luua üldisel viisil kasutajaliides mida saaks kasutada nii Windowsi operatisoonisüsteemil kui ka Mac'i operatsioonisüsteemidel. Seetõttu otsustati valida HTML põhiliseks informatsiooni kuvatehnoloogiaks ja kasutada JavaScripti rakenduse loogika loomiseks. Operatsioonisüsteemipõhise rakenduse näol on tegemist kergekaalulise veebibrauseri mootoriga, mis hangib kasutajaliidese ja loogika serveri käest rakenduse käivitamisel (Raji, 2012).

2.8. CoAP produktsioonis

Bakalaureusetöö kirjutamise ajal ei eksisteeri ühtegi mainimisväärset reaalset CoAP kasutusjuhtumit. Uurides ka ühte suuremat ja korralikku implementatsiooni, Californium, ei leidu kusagil viidet mõnele kuulsale rakendusele mida isiklikult keegi oleks kasutanud.

Antud fakti võib lugeda ka CoAP'i miinuseks. CoAP'i puhul üks suuremaid murekohti ongi kasutajaskonna väiksus. Autori arvates on selle üheks põhjuseks suurema eestvedaja puudumine., võrreldes näiteks MQTT protokolliga, mille Facebook on teinud asjade interneti ringis kuulsaks, rajades sellele enda Messenger rakenduse sõnumite saatmise loogika.Teiseks põhjuseks on autori arvates fakt, et CoAP protokoll on liiga lihtne ja sellele pole veel igapäeva elus kasutuskohta leitud. Autor usub et see kasutuskoht leitakse mõne manussüsteemi (ingl *embedded system)* näol lähima viie aasta jooksul.

3. Rakenduse arendus

Käesolevas bakalaureusetöös on erinevate rakenduste ehitamiseks kasutusel Rasperry Pi model B+ (edaspidi Raspberry) millel jookseb operatsioonisüsteem Raspian GNU/Linux 7 (wheezy) (edaspidi Raspian). Autor valis rakenduste ehitamiseks Raspberry platvormi, kuna see on piisavalt võimekas et jooksutada erinevaid protokolle, aga samas omades piiranguid mis tihtipeale piirangutega seadmetel on. Samuti on kasutusel Wi-Pi lisamoodul, et ühendada Raspberryt kohalikku traadita võrku (ingl *Wireless Local Area Network*). Protokollidest osutus valituks MQTT protokoll just tema implementeerimise lihtsuse ja algajasõbralikkuse tõttu. Samuti sai otsustavaks asjaolu, et võrreldes CoAP'iga on MQTT'ga seotud kommuun oluliselt suurem ja MQTT on leidnud kasutuspinda reaalse probleemi lahendamiseks.

3.1. Esmase rakenduse arendus

Antud rakenduse loomisjuhend sai läbi katsetatud ka Asjade Interneti kursuse raames.Esmase rakenduse tarbeks on vaja installida nii maaklerit jooksutavale arvutile (edaspidi maakler) kui ka Raspberryle Mosquitto klient. Täielikud paigaldamisjuhised erinevatele platvormidele on saadaval Mosquitto ametlikul kodulehelt (Mosquitto.org, kuupäev puudub) ja midagi keerulist seal ei ole. Soovitav on kasutada puhast Raspiani installatsiooni, vastasel korral võib tekkida konflike ja vigu. Mosquitto, nagu eelpool mainitud, on kergekaaluline avatud lähtekoodiga MQTT implementatsioon, mis sisaldab endas nii avaldajat kui ka tellijat.

3.1.1. Tere maailm

Kui Mosquitto on edukalt paigaldatud nii Raspberryle kui ka arvutile mis hakkab käituma kui maakler, siis oleks mõistlik katsetada kas kõik toimib "Tere maailm" näitel. Antud näite käivitamiseks on vajalik et mõlemad seadmed oleksid ühendatud samasse kohtvõrku.

Maakleris tuleb käivitada käsurea abil Mosquitto maakler.

```
mosquitto -v
```

Antud käsklus käivitab arvutis maakleri, kes hakkab sõnumeid vahendama spetsiifilise teema tellijatele. Tasub tähele panna, et sõltuvalt operatsioonisüsteemist, tuleb ka viidata kaustale kuhu Mosquitto on installitud, vastasel juhul kõlab antud käsklusele vastuseks command not found või mõni muu lause mis viitab sellele, et antud käsklust ei eksisteeri.

Lisaks eelmisele käsurea aknale tuleb avada veel üks käsurea aken ja jooksutada käsklus:

```
mosquitto_sub -h maakleri_ip -i testSub -t debug
```

Näiteks autori kohalik IP on 172.20.10.2 siis:

```
mosquitto_sub -h 172.20.10.2 -i testSub -t debug
```

Antud käsklus käivitab Mosquitto tellija kliendi, id-ga "testSub", tellima uudiseid teemaga debug maakleri käest kelle IP on 172.20.10.2

- -h on host parameeter, mis tähistab maakleri IP aadressit.
- -i on identifikaator, millega antud seadet teistest võrguseadmetest eristada.
- -t määrab ära teema, millised sõnumid antud tellijat huvitavad.

Järgmisena tuleb käivitada Rasperryl käsklus:

```
mosquitto_pub -h maakleri_ip -i rasperryPi -t debug -m 'Tere maailm'
```

Sarnaselt eelnevale tuleb asendada sõna maakleri_ip arvuti IP'ga millel jookseb maakler. Parameetritena antakse antud käsklusena juurde -m, mis tähistab saadetavat sõnumit (ingl *message*). Kui kõik on õigesti konfigureeritud peaks tellijale saabuma sõnum "Tere maailm"

3.1.2. LED Tulukese kontrollimine

Järgmiseks luuakse /home/pi/ kausta uus kaust nimega led:

```
mkdir led
```

Järgmiseks luuakse led kausta sisse kaust nimega python

```
mkdir /home/pi/led/python
```

Kausta python sisse luuakse üks Pythoni scripti, kasutadades Raspiani tekstiredaktorit nano. Tähtis on et antud käsklus jooksutataks juurõigustega, vastasel korral pärast pikka trükkimist ei ole võimalik antud faili salvestada.

```
sudo nano /home/pi/led/python/script.py
```

Järgmiseks on tarvis installida pip paketihaldur.

```
sudo apt-get update
```

```
sudo apt-get install python-pip
```

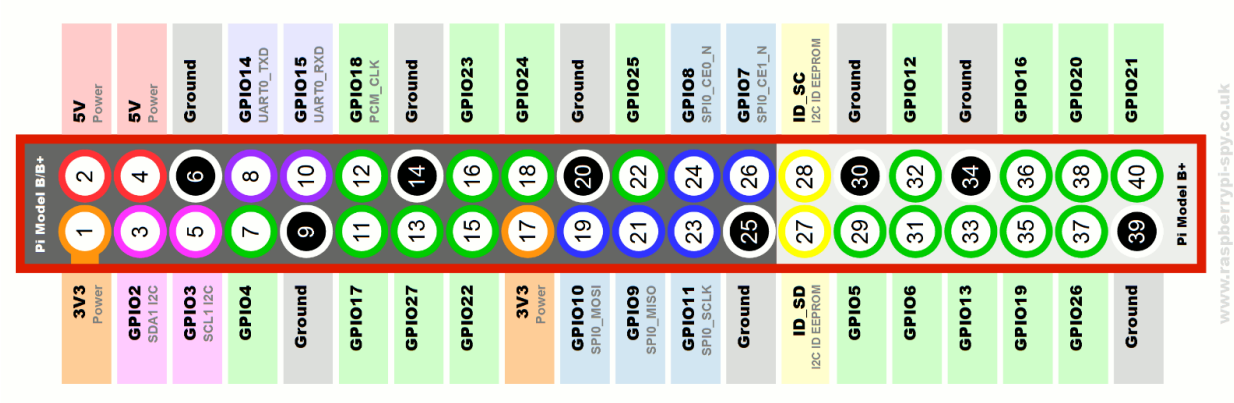
Kui see on edukalt paigaldatud tuleb, kasutades pip paketihaldurit, installida MQTT Pythoni klient Eclipse paho.

```
sudo pip install paho-mqtt
```

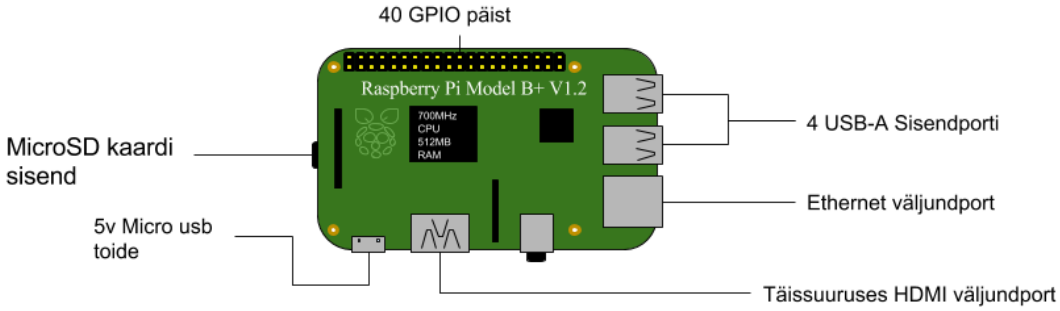
Ära tuleb uuendada ka RPi.GPIO moodul. See on vaikimisi juba paigaldatud Raspiani operatsioonisüsteemile, aga tasub veenduda et tegemist on kõige uuema versiooniga sellest.Veenduda tuleks selles, et see on uusim versioon käivitades käskluse:

```
sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

Järgnevalt on tarvis ühendada Raspberry külge üks LED tuluke. Allpool on pilt (Joonis 6) mis aitab aru saada mida mõni viik(ingl *pin*) Raspberry Pi+ mudelil teeb (Joonis 7).



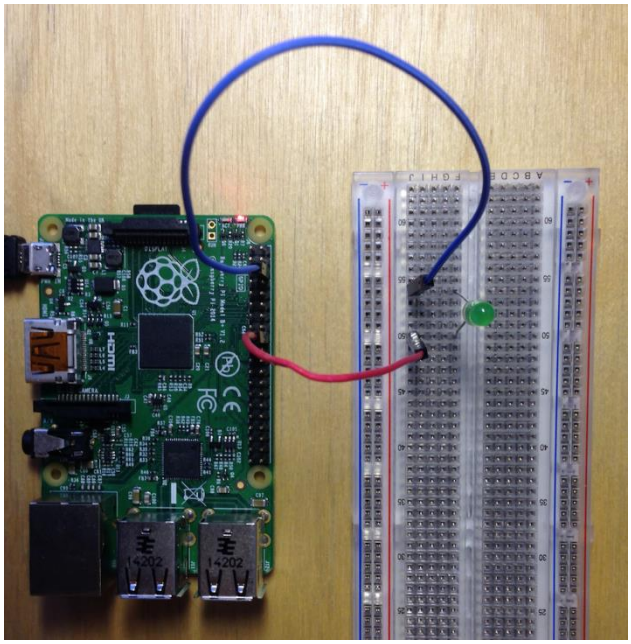
Joonis 6. Raspberry Pi Model B+ viikide kirjeldus (element14, kuupäev puudub)



Joonis 7 Raspberry Pi Model B+ võrdluseks

Järgnevalt seatakse üles Rasperry LED näite jaoks. Selleks ühendatakse üks juhe viik number 18 küljest makettlaua külge ja teiseks ühendatakse viik number 6 küljest juhe samuti makettlaua

külge (Joonis 8). Antud Raspberry puhul on viik number 18 GPIO24 (ingl *General Purpose input/output pin*) viik ja viik number 6 on maandusviik (ingl *Ground pin*).



Joonis 8. Raspberry ühendamine Led tulukesega

Järgnevalt kirjutatakse script.py faili programmijupp (Koodinäide 1), mis registreerib teemade „led“ ja „getStatus“ kuulajaks. Sõnumi „led“ saabumisel lülitab koodijupp LED tulukese kas siis sisse või välja ja sõnumi „getStatus“ saabumisel saadab ta huvitujatele antud LED tulukese staatuse, kas siis põleb või mitte.

```
import paho.mqtt.client as mqtt

import RPi.GPIO as GPIO

greenLedPin = 24

GPIO.setmode(GPIO.BCM)

GPIO.setup(greenLedPin, GPIO.OUT)

client = None

def green():

     if GPIO.input(greenLedPin):

          GPIO.output(greenLedPin, 0)
```

```
        else:
            GPIO.output(greenLedPin, 1)
def getStatus():
        client.publish("status", GPIO.input(greenLedPin))
def on_connect(client, userdata, flags, rc):
        print("Connected with result code: " + str(rc))
        client.subscribe("led")
        client.subscribe("getStatus")
def on_message(client, userdata, msg):
        if msg.topic == "led":
            green()
        else if (msg.topic == "getStatus"):
            getStatus()
        print(msg.topic + ": " + str(msg.payload))
try:
        client = mqtt.Client()
        client.on_connect = on_connect
        client.on_message = on_message
        #Järgnevas reas asendada sõna Maakleri_IP päris Maakleri Ip-
ga
        client.connect("Maakleri_IP", 1883, 33)
        client.loop_forever()
except KeyboardInterrupt:
```

```
    GPIO.cleanup()
```

Koodinäide 1. Led tulukese kontrollimine üle võrgu

Nüüd on võimalik jooksutada lihtsat LED tulukese tööle- ja kinni lülitamise näidet.Teises arvutis, mis paikneb rasperry pi-ga samas kohtvõrgus, käivitatakse maakleri instants.

```
mosquitto -v
```

Maakleril avatakse ka teine käsurea instants ja selle kaudu saadetakse sõnumi Raspberryle.

```
mosquitto_pub -h maakleri_ip -i rasperryPi -t LED -m 'test'
```

Tulemuseks peaks olema see, et Raspberry saab sõnumi kätte ja LED tuluke süttib sõnumi saabumisel põlema. Kui saata veel üks sõnum, analoogselt eelmisele näitele, siis LED tuluke kustub.

3.2. LED tulukese juhtimine veebibrauserist

Et suhelda eelneva näitega veebilehe kaudu kasutades WebSocketeid, laaditakse maaklerile alla paho JavaScripti klient (Eclipse Foundationd, 2016). Maakleri tarbeks luuakse kaust http_docs. Kaust tasub luua kergesti ligipääsetavasse kohta, kuna hiljem on tarvis sellele viidata veebibrauseri kontekstist. Kausta http_docs luuakse fail index.html ja veel üks kaust js, kuhu paigutatakse erinevad JavaScripti failid.

Index.html faili näol on tegemist JavaScripti põhise kliendiga (Lisa 1) mis võimaldab teistel seadmetel, mis kasutavad igapäevaseks internetisuhtluseks HTTP protokolli, ilma suurema vaevata suhelda Raspberry Pi'ga üle võrgu.

Kui veebileht on loodud, tuleb ka Mosquitto maakler vastavalt ära konfigureerida, kuna Mosquitto maakler vaikesättena ei aksepteeri sõnumeid WebSocketitelt. Selleks tuleb muuta mosquitto.conf faili. Autori maakleris asub see fail kaustas /usr/local/etc/mosquitto/mosquitto.conf .

Sinna faili lisatakse read, mis selgitavad Mosquitto maaklerile missugused pordid avada sissetulevate sõnumite kuulamiseks ja missugustelt protokollidelt aksepteeritakse sõnumeid (Koodinäide 2).

```
#Vaikimisi käivitatava sõnumite kuulaja seadmed
```

```
#IP tuleb samuti asendada maakleri IP-ga

bind_address 172.20.10.2

port 1883

#Antud read defineerivad ära WebSocketitega seonduva loogika

#WebSocketite saadetud sõnumeid kuulatakse pordil 8000

listener 8000

#Märgitakse ära et tegemist on WebSockets transpordi protokolliga

protocol websockets

#kaust millele pääseb ligi veebibrauserist ja kus paikneb index.html ja js kaust

http_dir /Users/toomashaide/Documents/bakalaureusetoo/mqttjs/

#Antud read on logi kuvamiseks

log_type error

log_type warning

log_type notice

log_type information
```

Koodinäide 2. Mosquitto.conf faili sisu

Kui kõik on õigesti üles seatud, siis on võimalik saata sõnumeid Raspberryle ka veebiliidese kaudu. Veebiliidesele saab aga ligi mõne moodsa veebibrauseriga, sisestades aadressi kujul: maakleri_ip:8000. Näiteks Koodinäide 2 puhul oleks see aadress 172.20.10.2:8000. Antud juhul on veebilehel ainult üks nupp (Joonis 9), millega saab LED tulukest sisse ja välja lülitada ning mis vastavalt LED tulukese seisundile muudab värvi.

Joonis 9. LED tulukese sisse- ja välja lülitamisrakenduse kasutajaliides

3.3. Jututoa rakenduse loomine

Facebook kasutab MQTT protokolli kasutajate vaheliste vestlussõnumite saatmiseks. Järgnevalt vaadeldakse kuidas oleks võimalik sarnane lahendus luua. Facebooki inseneride poolt avaldatud ehituslogist võib lugeda (Raji, 2012), et rakenduse arendamisel oli tähtis võimalikult palju koodi taaskasutada. Sel põhjusel arendatakse ka antud juhul kasutajaliides kasutades HTML'i.

Luuakse uus CSS-i fail, layout.css mis kirjeldab kuidas antud rakendus välja hakkab nägema (Koodinäide 3).

```
* {
  margin: 0;
}
html, body {
  height: 100%;
}
.wrapper {
  min-height: 100%;
  margin: 0 auto -100px;
}
footer, .push {
  height: 100px;
```

```
}
```

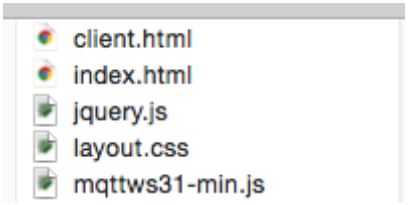
Koodinäide 3. Jututoa rakenduse stiilifail

Antud stiilifail laetakse üles kusagile veebiserverisse ja sellele hakkavad viitama jututoa rakenduse platvormi põhised kliendid.

Järgmisena muudetakse senist index.html faili, selliselt et kasutaja käest jututoa käivitamisel küsitaks kasutajanime ja jututoa IP'd millega ta soovib ühineda (Lisa 2).

Järgmisena luuakse fail nimega chat.html (Lisa 3). Antud faili sisaldab jututoa loogikat ja põhineb suuresti eelmisel LED tulukese kontorollimise näitel. Antud juhul kasutatakse jQuery teeki, et HTML faili sisu muuta selliselt, et sõnumi saabumisel kleebitakse see juurde chatbox div elemendile.

Pärast eelpoolmainitud muudatuste jõustumist on http_docs kaustas 5 erinevat faili (Joonis 10). Üks on siis index.html (Lisa 2) kuhu kasutaja maandub jututoa lehele navigeerides. Üks kirjeldab kliendi loogikat (Lisa 3). Üks kirjeldab väljanägemist (Koodinäide 3). Teised kaks on välised teegid mida kasutatakse sõnumite saatmise loogikas.



Joonis 10. http_docs kausta sisu pärast jututoa loomist

Pärast seda on võimalik käivitada Mosquitto maakler koos WebSocketite toega. Seejärel on võimalik üks klient käivitada näiteks lauaarvutist ja teine Raspberry peal. Antud jututuba pole küll ilusti kujundatud, aga teoorias on seda kõike väga lihtne muuta (Joonis 11). Et antud jututuba avalikuks teha, nii et sellele pääseks ligi ka välisvõrkudest tuleb konfigureerida ruuteri seadmeid, mis on antud bakalaureusetöö ulatusest väljas.

Joonis 11. Kasutusnäide jututoast

Edasiarendus antud jututoa rakendusest oleks kasutada Raspberry kliendi loomiseks mõnda kergekaalulist JavaScripti toega veebibrauserimootorit, mida on võimalik jooksutada ka piirangutega seadmetel, et ehitada omarakendus (ingl *Native application*). Üks võimalus on kasutada näiteks WCGBrowser raamistikku, mis põhineb PyQt4 raamistikul. Tänu sellele oleks võimalik Python'i koodist käivitada JavaScripti funktsioone ja vastupidi. Samuti säästetakse energiat kuna sõnumite saatmisel ei kasutata WebSocketeid.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua alguspunkt, millest lähtuda piirangutega seadmete juures asjade interneti lahenduse loomisel. Käesoleva töö tulemuste põhjal on võimalik teha asjade interneti rakenduste arendamisel esimene samm ja valida just selline protokoll, mis on kooskõlas rakenduse vajadustega.

Soositud protokolliks piirangutega seadmetele asjade internet lahenduse loomisel osutus MQTT. Selleks võrreldi omavahel kuute protokolli ja otsus langetati nende tugevuste ja nõrkuste põhjal. Bakalaureuse töö käigus loodi kolm näidisrakendust kasutades MQTT protokolli, et pakkuda rakenduse loomiseks alguspunkt ja näidata erinevaid viise, kuidas on võimalik üle võrgu mõne seadmega suhelda. Näidisrakenduse loomisjuhendit katsetati ka asjade interneti kursuse raames.

Kolmanda näitena loodi algeline jututuba lähtudes Facebooki Messenger rakenduse disainist. Kolmanda näite edasiarendusena pakub autor välja võimalust arendada platvormipõhine mobiilirakendus lähtudes autoripoolsetest näpunäidetest.

Bakalaureusetöö kirjutamise käigus omandas autor suurel hulgal uusi teadmisi asjade internetist, selle kasutuskohtadest ja sellega seotud tehnoloogiatest. Autori jaoks aga kõige tähtsamad teadmised omandas autor teadusliku uurimustöö kirjutamise kohta.

Kasutatud allikad

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE COMMUNICATION SURVEYS & TUTORIALS, 17*(4), lk 2353.

AMQP Working Group. (kuupäev puudub). *Products and Success Stories.* Retrieved Aprill 15, 2016, from amqp.org: https://www.amqp.org/about/examples

Banks, A., & Gupta, R. (10. Detsember 2015. a.). *MQTT Version 3.1.1 Plus Errata 01.* Kasutamise kuupäev: 24. Märts 2016. a., allikas oasis-open.org: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

Barroso, A. T., Valderrama, M. A., Roa, L. M., Tosin, J. R., & Ruiz, F. O. (Juuni 2016. a.). A Machine-to-Machine protocol benchmark for eHealth applications – Use case: Respiratory rehabilitation. *Computer Methods and Programs in Biomedicine, 129*, lk 1-11.

Beer Factory. (2016). *hbmqtt*. Kasutamise kuupäev: 20. Aprill 2016. a., allikas Github: https://github.com/beerfactory/hbmqtt

Bergmann, O. (2015). *libcoap*. Retrieved Aprill 1, 2016, from Github: https://github.com/obgm/libcoap

Bormann, C., Ersue, M., & Keranen, A. (Mai 2014. a.). *RFC 7228 - Terminology for Constrained-Node Networks.* Kasutamise kuupäev: 2. Aprill 2016. a., allikas Internet Engineering Task Force (IETF): https://tools.ietf.org/html/rfc7228

Cohn, R. (kuupäev puudub). *A Comparison of AMQP and MQTT.* Kasutamise kuupäev: 12. Aprill 2016. a., allikas oasis-open.org: https://lists.oasis-open.org/archives/amqp/201202/msg00086/StormMQ_WhitePaper_-_A_Comparison_of_AMQP_and_MQTT.pdf

Drager, D. (4. September 2015. a.). *MQTT vs Websockets vs HTTP/2: The Best IoT Messaging Protocol?* Kasutamise kuupäev: 16. Märts 2016. a., allikas SystemBash: https://systembash.com/mqtt-vs-websockets-vs-http2-the-best-iot-messaging-protocol/

Eclipse Foundation. (2015, November 4). *Californium (Cf) CoAP Framework*. Retrieved Märts 19, 2016, from Eclipse: https://projects.eclipse.org/projects/technology.californium

Eclipse Foundation. (14. Veebruar 2016. a.). *Mosquitto*. Kasutamise kuupäev: 23. Aprill 2016. a., allikas eclipse.org: http://projects.eclipse.org/projects/technology.mosquitto

Eclipse Foundation. (kuupäev puudub). *Android Service*. Kasutamise kuupäev: 24. Aprill 2016. a., allikas eclipse.org: http://www.eclipse.org/paho/clients/android/

Eclipse Foundation. (kuupäev puudub). *Java Client*. Kasutamise kuupäev: 23. Aprill 2016. a., allikas eclipse.org: http://www.eclipse.org/paho/clients/java/

Eclipse Foundation. (kuupäev puudub). *MQTT C Client for Posix and Windows*. Kasutamise kuupäev: 23. Aprill 2016. a., allikas eclipse.org: http://www.eclipse.org/paho/clients/c/

Eclipse Foundation. (kuupäev puudub). *paho*. Kasutamise kuupäev: 29. Veebruar 2016. a., allikas Eclipse: http://www.eclipse.org/paho/

Eclipse Foundation. (kuupäev puudub). *Python Client*. Kasutamise kuupäev: 23. Aprill 2016. a., allikas eclipse.org: http://www.eclipse.org/paho/clients/python/

Eclipse Foundationd. (2016). *Paho Downloads.* Retrieved Aprill 24, 2016, from eclipse.org: https://projects.eclipse.org/projects/technology.paho/downloads

element14. (kuupäev puudub). *Raspberry-Pi-GPIO-Layout-Model-B-Plus*. Retrieved Aprill 16, 2016, from element14: https://www.element14.com/community/servlet/JiveServlet/downloadImage/38-21309-230019/1600-533/Raspberry-Pi-GPIO-Layout-Model-B-Plus-rotated-2700x900.png?01AD=3GtuHQ34uIl_OcbM02DJicsi-SfsWCyDfZxYH4BpyuOu25dw48GMdIQ&01RI=E9AB43C28A1EEA3&01NA=

Evans, D. (Aprill 2011. a.). Kasutamise kuupäev: 1. Aprill 2016. a., allikas http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

Foster, A. (Juuni 2015. a.). *Messaging Technologies for the Industrial Internet and the Internet of Things Whitepaper*. Kasutamise kuupäev: Aprill 2016. a., allikas PrismTech: http://www.prismtech.com/sites/default/files/documents/Messaging-Whitepaper-040615_1.pdf

HiveMQ Team. (2015). *MQTT Essentials Part 2: Publish & Subscribe*. Retrieved Aprill 4, 2016, from HiveMQ: http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe

Mills, A. (2015). *CantCoap*. Retrieved Aprill 7, 2016, from Github: https://github.com/staropram/cantcoap

Mosquitto.org. (kuupäev puudub). *Downloads*. Kasutamise kuupäev: 13. Aprill 2016. a., allikas Mosquitto.org: http://mosquitto.org/download/

OASIS - Organization for the Advancement of Structured Information Standards. (29. Oktoober 2012. a.). *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0*. Kasutamise kuupäev: 20. Aprill 2016. a., allikas oasis-open.org: http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf

OMG - Object Management Group. (Aprill 2015. a.). *Data Distribution Service (DDS)*. Kasutamise kuupäev: 3. Aprill 2016. a., allikas Data Distribution Service™, V1.4: http://www.omg.org/spec/DDS/1.4/PDF/

Patierno, P. (2015). *GnatMQ*. Kasutamise kuupäev: 20. Aprill 2016. a., allikas Github: https://github.com/ppatierno/gnatmq

Patierno, P. (kuupäev puudub). *M2Mqtt*. Retrieved Aprill 21, 2016, from M2Mqtt & GnatMQ: https://m2mqtt.wordpress.com/

Raji, V. (8. Märts 2012. a.). *Under the Hood: Building Facebook Messenger for Windows*. Kasutamise kuupäev: 19. Aprill 2016. a., allikas facebook: https://www.facebook.com/notes/facebook-engineering/under-the-hood-building-facebook-messenger-for-windows/10150600868658920/

RTI International. (kuupäev puudub). *RTI Use Case - Integrate Medical Devices*. Kasutamise kuupäev: 29. Märts 2016. a., allikas RTI International: https://www.rti.com/resources/usecases/medical-devices.html

Selva, A. (2016). *Moquette*. Kasutamise kuupäev: 23. Aprill 2016. a., allikas Github: https://github.com/andsel/moquette

Shelby, Z., Hartke, K., & Bormann, C. (Juuni 2014. a.). *The Constrained Application Protocol (CoAP)*. Kasutamise kuupäev: 19. Märts 2016. a., allikas Internet Engineering Task Force (IETF): https://tools.ietf.org/html/rfc7252

SmeshLink Technology Co. (2015). *Welcome to CoAP.NET*. Retrieved Märts 4, 2016, from Smeshlink: http://open.smeshlink.com/CoAP.NET/

Zhang, L. (12. August 2011. a.). *Building Facebook Messenger*. Kasutamise kuupäev: 24. Aprill 2016. a., allikas facebook: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920

TIOBE software BV. (Aprill 2016. a.). *TIOBE Index for April 2016*. Kasutamise kuupäev: 8. Aprill 2016. a., allikas TIOBE - the software quality company: http://www.tiobe.com/tiobe_index?page=index

Undertow. (27. Aprill 2015. a.). *An in depth overview of HTTP/2*. Kasutamise kuupäev: 4. Aprill 2016. a., allikas http://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html

Waher, P. (2015, November 12). *XEP-xxxx: Quality of Service*. Retrieved Aprill 1, 2016, from xmpp.org: http://xmpp.org/extensions/inbox/qos.html

Wasilak, M., & Amsüss, C. (2014). *aiocoap*. Kasutamise kuupäev: 26. Märts 2016. a., allikas Github: https://github.com/chrysn/aiocoap

wcftutorial.net. (2014). *Request/Reply*. Retrieved Märts 29, 2016, from WCF Tutorial.net: http://wcftutorial.net/request-reply.aspx

XMPP. (kuupäev puudub). *An Overview of XMPP*. Kasutamise kuupäev: 29. Märts 2016. a., allikas xmpp.org: https://xmpp.org/about/technology-overview.html

XMPP. (kuupäev puudub). *History of XMPP*. Kasutamise kuupäev: 29. Märts 2016. a., allikas xmpp.org: http://xmpp.org/about/history.html

Summary

Title: Internet of Things Protocols for Constrained Devices

The purpose of this Bachelor Thesis was to provide a starting point for building an internet of things solution for constrained devices, by choosing the right application layer protocol. The results of this thesis can be used to make the initial decision of which protocols suit the applications needs.

The aims of the current research paper were to:

- Figure out which protocols are relevant for building Internet of Things solutions.
- Figure out which protocols should be used on constrained devices.
- Figure out which protocols should be used for the purpose of home automation.

A favored protocol, for building internet of things solutions on constrained devices, was determined by analysing the strengths and weaknesses of 6 application layer protocols. In the end MQTT emerged as the favored protocol by the author.Three examples were developed using MQTT for the purpose of providing a starting point for building applications on constrained devices. The examples showcase the different possibilities of how one can communicate with a device over a network. The final example recreated a basic chat client that acts also as a starting point for developing a native chat application, for example on a smartphone.

The writing of this thesis gave the author a good insight in to the relevant Internet of Things protocols and their use cases. Most importantly, it gave the author an experience of writing a scientific thesis.

Lisad

Lisa 1. JavaScripti kliendi kood Raspberryga suhtlemiseks

```
<!doctype HTML>
<html>
<head>
<title>Led kontroller</title>
<script type="text/javascript" src="js/mqttws31.js"></script>
<script type="text/javascript" src="js/jquery.js"></script>
<style type="text/css">
  button {
    display: block;
    margin: 5px 5px 5px 5px;
    width: 160px;
    height: 45px;
    font-size: 24pt;
    font-weight: bold;
    color: white;
  }

  #LOW {
    background-color: Black;
  }
```

```
  #HIGH {
    background-color: Blue;
  }


</style>
</head>
<body>
<script>


//Genereerib UUID
var uuid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
    var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
    return v.toString(16);
});


// Luuakse kliendi instants
client = new Paho.MQTT.Client("172.20.10.2", 8000, uuid);


// Määratakse ära callback funktsioonid
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
```

```
// Luuakse kliendi ühendus
client.connect({onSuccess:onConnect});

// tagasihelistus funktsioon mis käivitatakse kliendi ühendamisel
function onConnect() {
  // Kui ühendus loodud, saadetakse staatuse päring Raspberry-le
  console.log("onConnect");
  client.subscribe("status");

  message = new Paho.MQTT.Message("What's your status?");
  message.destinationName = "getStatus";
  client.send(message);
}
//Funktsioon mis paneb led tulukese põlema
function turnOnLed(){
  message = new Paho.MQTT.Message("Let there (not) be light");
  message.destinationName = "led";
  client.send(message);
}
```

```
// tagasihelistusfunktsioon mida kutsutakse välja kui klient kaotab ühenduse
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0) {

console.log("onConnectionLost:"+responseObject.errorMessage);
  }
}

// tagasihelistusfunktsioon mis kutsutakse välja uue sõnumi saabumisel
function onMessageArrived(message) {
  console.log("onMessageArrived:"+message.payloadString);

//Loogika mis muudab vastavalt tagastatud led väärtusele nupu värvi
  if (message.payloadString == 1) {
    $('#LOW').attr('id', 'HIGH');
  } else if (message.payloadString == 0) {
    $('#HIGH').attr('id', 'LOW');
  };
}
  </script>
<button id="LOW" onClick="turnOnLed()">Led</button>
```

```
</body>

</html>
```

Lisa 2. Kasutaja autentimise lehekülg

```
<!DOCTYPE html>

<html>

  <head>

    <title>Chatbox</title>

    <meta name="description" content="Simpe MQTT chatroom.">

    <link rel="stylesheet" type="text/css" media="screen" href="layout.css">

    <script language="JavaScript" type="text/javascript" src="jquery.js"></script>

    <script type="text/javascript">

      $(document).ready(function() {

        $('#submit').click(function() {

        var username = $('#username').val();

        var ip = $('#ip').val();

        window.location = 'client.html?username=' + username + '&ip=' + ip;

        });

      });

    </script>

  </head>
```

```
  <body>
      Username:<br>
      <input id="username" type="text" name="username"> <br>
      Ip:<br>
      <input id="ip" type="text" name="ip"><br>
      <button id="submit">Start</button>
   </body>
</html>
```

Lisa 3. Jututoa kliendi loogika

```
<!DOCTYPE html>
<html>
  <head>
    <title>Chatbox</title>
    <meta name="description" content="Simpe MQTT chatroom.">
    <link rel="stylesheet" type="text/css" media="screen" href="layout.css">
    <script language="JavaScript" type="text/javascript" src="jquery.js"></script>
    <script language="JavaScript" type="text/javascript" src="mqttws31-min.js"></script>
    <script type="text/javascript">
```

```
    var getUrlParameter = function getUrlParameter(sParam) {
      var sPageURL = decodeURIComponent(window.location.search.substring(1)),
          sURLVariables = sPageURL.split('&'),
          sParameterName,
          i;

      for (i = 0; i < sURLVariables.length; i++) {
          sParameterName = sURLVariables[i].split('=');

          if (sParameterName[0] === sParam) {
              return sParameterName[1] === undefined ? true : sParameterName[1];
          }
      }
    };

    var client;
    var username = getUrlParameter('username');
    var ip = getUrlParameter('ip');

    //Genereerib UUID
    var uuid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
```

```
      var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
      return v.toString(16);
    });

      $(document).ready(function() {
        if (username && ip) {
          client = new Paho.MQTT.Client(ip, 8000, uuid);
          client.onConnectionLost = onConnectionLost;
          client.onMessageArrived = onMessageArrived;
          client.connect({onSuccess:onConnect});
        } else {
          window.location = "index.html";
        };

        $('#sendMessageButton').click(function() {
          var text = $('#messageBody').val();
          message = new Paho.MQTT.Message(username + ": " + text);
          message.destinationName = "chatbox";
          client.send(message);
        });
```

```
      });


      function addContent(message){
        $("#chatbox").append(message + "<br>");
      }


// tagasihelistus funktsioon mis käivitatakse kliendi ühendamisel
function onConnect() {
  // Kui ühendus loodud, saadetakse staatuse päring Raspberry-le
  console.log("onConnect");
  client.subscribe("chatboxNotifications");
  client.subscribe("chatbox");


  var notification = "<b>" + username + " just joined the chatbox!" + "</b>";
  message = new Paho.MQTT.Message(notification);
  message.destinationName = "chatboxNotifications";
  client.send(message);
}


// tagasihelistusfunktsioon mida kutsutakse välja kui klient kaotab ühenduse
function onConnectionLost(responseObject) {
```

```
  if (responseObject.errorCode !== 0) {

console.log("onConnectionLost:"+responseObject.errorMessage);
  }
}

// tagasihelistusfunktsioon mis kutsutakse välja uue sõnumi saabumisel
function onMessageArrived(message) {
  console.log("onMessageArrived:"+message.payloadString);
  addContent(message.payloadString);
}
    </script>
  </head>
  <body>
    <div class="wrapper">
      <header>
        <h1>Welcome to using our new edition of chatbox!</h1><br>
      </header>
     <article id="chatbox">
        <h2>----------------------------------------------------------</h2>
     </article>
```

```
      <div class="push"></div>
    </div>
   <footer>
      <textarea id="messageBody" rows="4" cols="50" placeholder="Enter your message..."></textarea>
     <button type="button" id="sendMessageButton">Send message</button>
    </footer>
   </body>
</html>
```