KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Justė Plytnikaitė

AUTONOMINIO ROBOTO SAUGAUS KOMUNIKAVIMO METODO SUDARYMAS IR TYRIMAS

Baigiamasis magistro darbas

Vadovas
Prof. Algimantas Venčkauskas

KAUNAS, 2018

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

AUTONOMINIO ROBOTO SAUGAUS KOMUNIKAVIMO METODO SUDARYMAS IR TYRIMAS

Baigiamasis magistro darbas
Informacijos ir informacinių technologijų sauga (kodas 621E10003)

Vadovas
Prof. Algimantas Venčkauskas
2018-05-23

Recenzentas
Doc. Tomas Adomkus
2018-05-23

Projektą atliko
Justė Plytnikaitė
2018-05-23

KAUNAS, 2018

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Justė Plytnikaitė

(Studento vardas, pavardė)

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

(Studijų programos pavadinimas, kodas)

„Autonominio roboto saugaus komunikavimo metodo sudarymas ir tyrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 18 m. gegužės 23 d.

Kaunas

Patvirtinu, kad mano **Justės Plytnikaitės** baigiamasis projektas tema „Autonominio roboto saugaus komunikavimo metodo sudarymas ir tyrimas“ yra parašytas visiškai savarankiškai, o visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Plytnikaitė, J. „Autonominio roboto saugaus komunikavimo metodo sudarymas ir tyrimas“. Magistro baigiamasis projektas / vadovas prof. Algimantas Venčkauskas; Kauno technologijos universitetas, Informatikos fakultetas, Kompiuterių katedra.

Kaunas, 2018. 60 p.

SANTRAUKA

Autonominiai robotai pritaikomi įvairiausiose srityse – pradedant nuo namų ūkio, baigiant karo pramone. Jau netolimoje ateityje, autonominiai robotai neišvengiamai taps vis išmanesni ir atsiras beveik kiekvieno žmogaus namuose. Todėl verta susirūpinti saugaus komunikavimo metodo naudojimu, kuris padėtų išvengti savanaudiškų pašalinių asmenų ir kenksmingų atakų, kurių metu įmanomas slaptų, asmeninių duomenų nutekėjimas. Tačiau kuriant saugų komunikavimo metodą autonominiam robotui būtina atsižvelgti į jo ribotus išteklius – tiek elektros energijos sunaudojimą, tiek ne itin galingus skaičiavimo operacijų pajėgumus. Šiam tikslui pasiekti galima naudoti supaprastintus ir sudėtingesnius komunikacijų protokolus, tačiau būtina įdiegti papildomą saugą.

Dėl šios priežasties, baigiamojo darbo tikslas yra patobulinti esamus autonominio roboto komunikacijos metodus saugos srityje, taip pasiūlant veiksmingą komunikacijos apsaugos metodą. Sukurtas metodas privalo tenkinti roboto ribotų išteklių keliamus reikalavimus: būti pakankamai saugus ir greitas, tačiau nenaudoti per daug skaičiavimo išteklių bei elektros energijos. Literatūros analizėje apžvelgiamos ir ištiriamos esamos robotų saugos problemos ir pažeidžiamumai, palyginami naudojami komunikacijos metodai. Baigiamojo darbo metodo prototipe realizuojamas saugus komunikavimas naudojant dviejų rūšių protokolus – HTTP ir MQTT.

Galutiniai baigiamojo darbo rezultatai pateikiami eksperimentiniame tyrime, kur palyginamos protokolų saugos lygmenų modifikacijos, skaičiavimo išteklių naudojimas, greitaveika, atsirandantis papildomas duomenų srautas ir suvartojamas elektros energijos kiekis.

Raktažodžiai: *autonominių robotų sauga, komunikacijos metodai, ribotų išteklių sistemos, supaprastinti protokolai, Arduino platforma, ESP8266*

Plytnikaitė, J. *Development and Research on Secure Autonomous Robot Communication Method: Master's* thesis / supervisor prof. Algimantas Venčkauskas. Department of Computer Science, Faculty of Informatics, Kaunas University of Technology.

Key words: *autonomous robots security, communication methods, limited resources systems, lightweight protocols, Arduino platform, ESP8266*

Kaunas, 2018. 60 p.

SUMMARY

Autonomous robots are used in a wide range of areas, ranging from households to military industry. Soon autonomous robots will inevitably become more intelligent and will be used in almost every person's home. Therefore, it is time to think about using secure communication method that would help to avoid selfish strangers and malicious attacks that could allow the leak of sensitive personal data. However, when developing a secure communication method for an autonomous robot, it is necessary to consider its limited resources - both power consumption and low power computing capabilities. Both simplified and more sophisticated communication protocols can be used to achieve this, but additional security needs to be implemented.

For this reason, the aim of the final thesis is to improve the existing methods of autonomous robotic communication in the field of safety, thus offering an effective method of communication protection. The developed method must meet the requirements of the robot's limited resources: to be safe and fast enough, but not to use too much computing resources and electricity. The literature analysis reviews and examines current robotic security issues and vulnerabilities, compares used communication methods. Secure communication in the method prototype is implemented using two types of protocols - HTTP and MQTT.

The results are presented in the experimental study, which compares the security levels modifications of the protocols, use of calculation resources, speed, additional data overhead and the amount of electricity consumed.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

3DES – trigubas informacijos užšifravimo algoritmas (simetrinio rakto blokinis šifras) (angl. *Triple Data Encryption Algorithm*)

6LoWPANs – interneto protokolo versija Nr. 6, veikianti negalinguose bevieliuose asmenininiuose tinkluose (angl. *IPv6 over Low-Power Wireless Personal Area Networks*)

AES – pažangus šifravimo standartas (šifravimo algoritmas) (angl. *Advanced Encryption Standard*)

CA – sertifikatų valdžia (išduodanti sertifikatus) (angl. *Certificate Authority*)

CoAP – suvaržytų aplikacijų protokolas (angl. *Constrained Applications Protocol*)

CPU – centrinis procesorius (angl. *Central Processing Unit*)

DNS – sričių vardų sistema (angl. *Domain Name System*)

DoS – atsisakymo aptarnauti ataka (angl. *Denial of Service*)

DTLS - datagramų transporto sluoksnio apsauga (angl. *Datagram Transport Layer Security*)

HTTP – hipertekstų persiuntimo protokolas (angl. *Hypertext Transfer Protocol*)

ID – identifikatorius (angl. *Identifier*)

IoT – „daiktų internetas“ (angl. *Internet of Things*)

IPv6 – interneto protokolo versija Nr. 6 (angl. *Internet Protocol version 6*)

M2M – veikimo principas: prietaisas prietaisui (angl. *machine-to-machine*)

MD5 – žinučių santrauka Nr. 5 (maišos funkcija)

MQTT – žinučių eilės sudarymo telemetrijos transporto protokolas (angl. *Message Queuing Telemetry Transport Protocol*)

PSK – iš anksto pasidalintas raktas (angl. *Pre-Shared Key*)

QoS – paslaugų kokybės lygis (angl. *Quality of Service*)

RAM – laisvosios prieigos atmintis (angl. *Random Access Memory*)

RC4 – Rivest šifras Nr. 4 (srauto šifras) (angl. *Rivest Cipher 4*)

RSA – Rivest – Shamir – Adleman viešojo rakto kriptosistema

SHA1 – saugus maišos algoritmas Nr. 1 (maišos funkcija) (angl. *Secure Hash Algorithm 1*)

SOC – „sistema ant lusto“ (angl. *System-on-a-Chip*)

SRAM – statinė laisvosios prieigos atmintis (angl. *Static Random-Access Memory*)

SSL / TLS – saugiųjų jungimų lygmens protokolas / transporto lygmens protokolas, skirtas saugumui užtikrinti TCP / IP tinkluose (angl. *Secure Socket Layer / Transport Layer Security*)

TCP / IP – transporto valdymo protokolas / interneto protokolas, t.y. protokolų rinkinys, aprašantis duomenų persiuntimą tarp įvairių tipų kompiuterių ir operacinių sistemų (angl. *Transport Control Protocol / Internet Protocol*)

TLV – rūšis, ilgis, vertė (angl. *Type-Length-Value*)

TTL – veikimo principas: tranzistorius tranzistoriui logika (angl. *Transistor–transistor Logic*)

UART – universalus asinchroninis imtuvas – siųstuvas (angl. *Universal Asynchronous Receiver-Transmitter*)

UDP – vartotojo datagramų protokolas (angl. *User Datagram Protocol*)

UML – vieninga modeliavimo kalba, skirta diagramoms (angl. *Unified Modeling Language*)

URIs – universalieji išteklių identifikatoriai (angl. *Uniform Resource Identifiers*)

URL – universalusis adresas (angl. Universal *Resource Locator*)

USB – universali nuoseklioji kompiuterio magistralė (angl. *Universal Serial Bus*)

VPN – virtualusis privatusis tinklas (angl. *Virtual Private Network*)

WiFi – belaidis tinklas

ĮVADAS

Robotai yra pasmerkti susidurti su panašiomis problemomis į tas, kurioms į akis žiūrėjo kompiuterių pramonė prieš 30 metų pradėjus plisti internetui. Staiga sistemos, apie kurių saugumą dar net nebuvo spėta gerai pagalvoti, tampa pažeidžiamos. Industriniai robotai, fiziškai apsaugoti sienų ir kibernetiškai izoliuoti, sėkmingai dirbo jau kelis dešimtmečius. Bet šiandien jiems apsaugos nebegalima užtikrinti: gamybos įrengimai sujungiami į paskirstyto valdymo tinklą tam, kad padidinti produktyvumą ir sumažinti valdymo išlaidas, ir tokiu būdu robotai pradeda kentėti nuo tokių pačių problemų kaip ir bet kokios kitos kibernetinės – fizinės sistemos [1].

Robotų panaudojimo sritis yra be galo plati ir kasdien vis didėja. Pradžioje, robotikai dar tik besiskinant kelią, visas dėmesys buvo skirtas industrinių robotų kūrimui, kurie pagelbėdavo automatizuoti gamybos proceso dalis. Tik neseniai robotai buvo pradėti naudoti tokiose sferose kaip karinė ir medicinos pramonė, save vairuojantys automobiliai, grindų valymas, vejos pjovimas ir net realistiškų gyvūnų kūrimas, kurie pasitarnauja žmonių potrauminėje reabilitacijoje [2].

Atsižvelgiant į tai, kad netolimoje ateityje robotai neišvengiamai bus naudojami kasdieniniame mūsų gyvenime, tampa privaloma pradėti rūpintis metodais, kurie mums padėtų išvengti nepageidaujamų atakų ir savanaudiškų, kenksmingų išnaudojimų robotų komunikacijų programinėje įrangoje. Deja, istorija linkusi pasikartoti – kaip ir kompiuterių amžiaus pradžioje, robotika vystosi „laimingo naivumo burbule“, kur saugumo taisyklės prieš išorines atakas nepritaikomos, tariant, kad visi robotiką išmanantys žmonės turi tik gerus ketinimus. Nors galbūt tai ir buvo tiesa praeityje, masiškas robotų naudojimo neišvengiamumas padidins atakų tikimybę. Šis faktas itin aktualus gynyboje, medicinoje ir kitose kritiškai svarbiose srityse, kurios susiję su žmogumi ir galimybe jį rimtai sužaloti, taip pat ir kėsinimusi į privatumą. Dėl šių priežasčių mokslininkai ir pramonė turėtų padidinti pastangas sprendžiant kibernetinio saugumo problemas ir sekti tinkamais pavyzdžiais kuriant ir platinant robotų programinę įrangą. Nepaisant kitų problemų šaltinių, tokių kaip programinės įrangos klaidos ar pažeidžiamumai (buferio perpildymas, komandų įterpimas ir t.t.), manoma, kad komunikacijos robotikoje šiuo metu yra viena iš silpniausių vietų [3].

Iki tol, kol galėsime tokią techniką naudoti kovos lauke, dar net nepriėjus prie visų galimybių įgyvendinimo, teks išspręsti daugybę techninių problemų, pvz. kaip patikimumas, komunikacijos ir suderinimas, naudojimo doktrinos ir, tikriausiai pati svarbiausia ir daugiausiai lemianti – saugumas [4]. Taigi šio darbo tikslas ir yra apžvelgti potencialias saugumo ir privatumo grėsmes, identifikuoti su jomis susijusius iššūkius ir pasiūlyti sprendimų, kaip būtų galima apeiti šiuos kylančius sunkumus. Norima pabrėžti, kad šis metas yra pats idealiausias laikas vykdyti tokius tyrimus – kol robotikos sritis yra palyginus jauna, ir tol, kol robotai, turintys rimtų ir esminių saugumo defektų nepasirodė kiekvienoje parduotuvėje.

Darbo tikslas – pasiūlyti veiksmingą autonominio roboto komunikacijos apsaugos metodą.

Darbo uždaviniai:

- atlikti analizę ir ištirti esamas robotų sistemų saugos problemas ir pažeidžiamumus;
- ištirti ir palyginti pramonėje naudojamus komunikacijos metodus;
- sukurti veiksmingą roboto komunikacijos saugos metodą;
- įvertinti sukurtą metodą ir palyginti gautus rezultatus.

1. AUTONOMINIŲ ROBOTŲ SAUGUMO ANALIZĖ

1.1. Autonominių robotų saugumo problema

Mobiliaisiais, galinčiais judėti robotais vadiname jungtines mechanines sistemas, kurios sugeba keliauti autonomiškai ar bent jau pusiau autonomiškai ir turi galią daryti įtaką artimai aplinkai. Kai kurie iš tokių robotų taip pat turi belaidžio komunikavimo funkciją, kuri iškart priverčia susimąstyti apie bendradarbiaujančių robotų idėją. Mobiliuosius robotus galime rasti vis didėjančiame sektorių kiekyje, pavyzdžiui, gamyboje, bet taip pat ir aptarnavimo sferoje, sveikatos sektoriuje, apskritai bet kokioje žmogaus profesijoje, kur būtina vykdyti pavojingas užduotis. Japonija ir Vokietija yra labiausiai tobulinant civilinius mobiliuosius robotus pažengusios šalys, o Jungtinės Valstijos ir Izraelis dominuoja karinių robotų rinkoje. Prancūzijos ekonomikos ministerija prognozuoja, kad robotai gali atstovauti 50 milijardų JAV dolerių rinką iki 2020 metų pabaigos [6].

Pačios populiariausios atakos, su kuriomis susiduria kompiuterių naudotojai, yra, pavyzdžiui, atsisakymo aptarnauti atakos DoS (angl. *Denial of Service*), slaptas pasiklausymas ir duomenų rinkimas, bandymas gauti priėjimą prie vietų, kurioms vartotojas neturi leidimo, bandymas skirti sau daugiau prieigos teisių ar saugomų duomenų paskelbimas. Prie šių visų problemų robotai prideda dar ir vieną papildomą veiksnį – fizinę sąveiką. Tuo tarpu, kol kompiuterio ar serverio užgrobimas gali baigtis informacijos praradimu (priskaičiavus visas su tuo susijusias išlaidas), roboto užvaldymas gali sukelti grėsmę bet kam, kas tuo metu bus šalia. Kai robotai vis labiau jungiami į komunikacijų tinklus, atrodo, tinkama robotams valdyti dar kartą panaudoti tuos pačius metodus, skirtus saityno programoms. Tačiau mokslininkai laikosi nuomonės, kad yra daug skirtumų tarp paprastų kompiuterių, kurie komunikuoja tinklo pagalba, ir robotų, darančių tą patį. Mohanarajah et al. (2015) pateikia skirtumus tarp saityno ir robotų mobiliųjų programų: „Saityno programos paprastai yra pavieniai, be būsenos procesai, kurie naudoja užklausos – atsakymo modelį bendraujant su vartotoju. Tuo tarpu robotų programos turi būsenas ir daugiaprogramį režimą bei reikalauja dvikryptės komunikacijos su vartotoju. Šie esminiai skirtumai gali nuvesti link skirtingų kompromisų ir dizaino pasirinkimų, ir gali baigtis skirtingais programinės įrangos sprendimais saityno ir robotų mobiliosioms programoms [7].“ Prie šių skirtumų galime pridėti ir realaus laiko suvaržymus, kurie charakterizuoja robotikos programas.

Robotikos pramonė klesti dėl daugybės mokslininkų ir pramonės pastangų, sutelktų į robotų integravimą į ne tik profesionalų naudojimą gamyboje, bet ir kasdieniniam naudojimui namuose. Potenciali nauda matoma aiškiai. Robotai galėtų padėti atliekant namų ir ūkio ruošos darbus, pasiūlyti draugiją, pagelbėti su sveikatos problemomis ir senjorų priežiūra. Tačiau, kiek yra žinoma, šiuo metu žvelgiant į robotų saugumo ir privatumo problemas, matoma didelė neužpildyta erdvė su neišspręstomis problemomis. Poreikis spręsti visas šias problemas yra aiškus – ateities robotai gali su

savimi atnešti dar daugiau sudėtingesnių saugumo ir privatumo grėsmių bet kuriems jų naudotojams – tiek savininkams, tiek žmonėms, gyvenantiems aplink. Sprendžiant daugybę atvejų gali būti net neaišku, kaip apskritai nugalėti šias saugumo ir privatumo grėsmes [8].

1.2. Autonominių robotų apžvalga

Apžvelgiama keletas robotikos sričių, kuriose saugumas ir privatumas yra itin aktualus, taip pat pateikiamos pavojingos situacijos, galinčios įvykti laiku nepasirūpinus saugumu.

1.2.1. Gynyba ir erdvė

Kariuomenės sritis turėtų puikiai išmanyti ir žinoti apie tinkamiausius kibernetinio saugumo metodus robotams. Nepilotuojami orlaiviai, dažniau vadinami tiesiog dronais, skirti aplinkos stebėjimui ir kovinėms misijoms. Savaime suprantama, kad bet kokios su šiais orlaiviais vykdomos komunikacijos turi būti užšifruotos, bet realybėje viskas vyksta kitaip. Pavyzdžiui, 2012 metais atliktas tyrimas [9] parodė, kad tik 30 – 50 procentų visų JAV „*Predators*“ ir „*Reapers“* (tai du populiariausi JAV kariuomenės dronų tipai) duomenis siuntė pilnai užšifruotus [3].

Galimos situacijos:

- Neautorizuotas objektas slapta stebi ir surinkinėja stebėjimo duomenis iš drono, pasinaudodamas neužšifruotu ryšiu perima drono valdymą ir sudaužo jį tankiai apgyvendintoje vietovėje.
- Neautorizuotas objektas perima roboto, dirbančio Tarptautinėje Kosminėje Stotyje, valdymą ir sugadina vykstantį eksperimentą.

1.2.2. Namų ūkio robotai

Ši rinka auga tiek moksliniuose tyrimuose, tiek komercijoje parduodant robotus. Planuojama, kad robotai bus naudojami kaip papildoma pagalba namuose. Pavyzdžiui, vienas iš tokių projektų yra „*Care-O-bot“* [10], namų robotas – asistentas. Vienoje iš galimų versijų robotas turi mikrofoną, kamerą ir 3D jutiklius. Toks daviklių rinkinys gali rinkti milžinišką kiekį informacijos, kuri privalo būti apsaugota [8]. Manoma, kad aptarnaujantys robotai ateityje gali rinkti žmogaus sveikatos duomenis, o įstatymai reikalauja, kad tokie duomenys būtų tvarkomi itin atidžiai [3].

Galima situacija:

- Neautorizuotas objektas perima namų ūkio roboto valdymą, siunčiasi gyvai transliuojamus vaizdus, tokiu būdu nutekindamas privačius duomenis.

1.2.3. Katastrofų robotai

Nuo 2011 metų Fukušimos Daiichi atominės nelaimės, robotikos bendruomenė padidino pastangas kurti ir apmokyti robotus katastrofų scenarijams. Viena iš užduočių, su kuria tikėtina, kad robotams teks susidurti katastrofos akivaizdoje, yra susijusi su priėjimu prie pavojingų sistemų ir jų

taisymu bei atjungimu. Dėl galimo pavojaus, kuris kyla tokiose situacijose [5], negalima suteikti galimybės modifikuoti robotus išorinės atakos metu [3].

Galima situacija:

- Neautorizuotas objektas perima roboto valdymą, kurio užduotis atjungti atominį reaktorių, kuriam gresia radioaktyvių medžiagų nutekėjimas, ir sutrukdo atjungimo operaciją.

1.3. Kompiuterių ir robotų saugos palyginimas

Sunku kalbėti apie autonominius robotus taip pat kaip kompiuterių saugumo tyrėjai paprastai kalba apie kompiuterius. Autonominis robotas neturi savyje informacijos, kuri būtų prieinama tam tikriems autorizuotiems „vartotojams“. Tiesa tokia, kad iš tiesų autonominis robotas net neturi „vartotojų“ įprastai suprantama prasme. Apie autonominį robotą geriausia galvoti kaip apie įrenginį, kuris geba reguliuoti savo elgesį. Robotas susideda iš jutiklių, vykdytojų (valdomų galūnių) ir vidinių duomenų apdorojimo išteklių. Bet nėra (arba bent jau neturėtų būti) tiesioginio priėjimo prie vidinių duomenų apdorojimo išteklių. Roboto *raison d'être* (pran. *egzistavimo prasmė*) yra atskleisti teisingą elgesį pagal esamus jutiklių įvesties duomenis, misiją ir visų jo sąveikų su aplinka istoriją. Pagrindinis priešų atakų prieš robotą tikslas visada bus neleisti robotui elgtis „tinkamai“, apribojant valdomų galūnių veiksmingumo galimybes arba kažkokiu būdu priverčiant robotą elgtis „neteisingai“ [4].

Ateityje planuojama robotų programavimą supaprastinti iki mobiliųjų telefonų lygio, palaikyti kelių robotų programas vienu metu, ir nuo pat pradžių teikiant tokį funkcionalumą užtikrinti, kad bus naudojami atitinkami saugumo mechanizmai. Nors daugybę tų pačių saugumo technologijų, kurias naudojame tradiciškesniuose duomenų apdorojimo įrenginiuose, bus galima pritaikyti ir robotams (pvz., mažos priėjimo privilegijos prie programinės įrangos ir modulinė įranga), robotai nuo pat pamatų skiriasi nuo tradicinių kompiuterinių sistemų įdomiais ir neįprastais atžvilgiais [2].

Kadangi robotikos platforma yra sukurta naudojant tradicines kompiuterines sistemas, savaime aišku, kad jos bus irgi pažeidžiamos tose pačiose saugumo spragose [8]. Tačiau tuo pačiu robotika ir pateikia iš esmės besiskiriančią platformą nuo tradicinių kompiuterinių sistemų, taip su savimi atsinešdama ir naujų saugumo spragų rinkinį. Toliau apžvelgiama kuo robotai yra panašūs į tradicines kompiuterines platformas, ir kuo skiriasi, ir kokie iššūkiai gali kilti iš tų skirtumų [2].

1.3.1. Panašumai

Daugybė robotų platformų savo komponentams naudoja plačiai paplitusią techninę ir programinę įrangą. Pavyzdžiui, šiame straipsnyje [2], pavyzdinis robotas naudoja tinklinuką (angl. *netbook*) su Linux programine įranga kaip pagrindinį valdymo įrenginį. Tam, kad robotas būtų saugus, valdymo įrenginys taip pat turi būti saugus. Sukompromitavus roboto operacinę sistemą per branduolio lygio nuotolinio kodo vykdymo įsilaužimo programą, robotas taip pat atsidurtų pavojuje. Dalis robotų

sistemų komponentų yra roboto išorėje: nuotolinis serveris, skirtas laikyti įvykių žurnalus ar vykdyti brangias skaičiavimo operacijas, ar netgi teikti komandas robotui. Visi šie komponentai komunikuoja vienas su kitu naudojantis tradiciniais tinklais, pvz. belaidžiu ryšiu. Kuriant saugias roboto programas, būtina pagalvoti ir apie šias nuotolines sistemas bei tinklus.

1.3.2. Skirtumai

Robotai nuo tradicinių kompiuterių sistemų skiriasi dviem svarbiomis ypatybėmis. Pirma, robotai gali judėti autonomiškai, t.y. savarankiškai, ir jie turi manipuliatorius (pvz., rankas), kurios gali daryti įtaką fiziniam pasauliui. Antra, daugumoje savarankiškai „mąstančių“ robotų pamatiniai algoritmai iš esmės yra paremti tikimybėmis. Šie skirtumai pateikia galybę iššūkių visiems saugių robotų sistemų kūrėjams.

Robotų mobilumas ir paleidimas. Robotai geba tiesiogiai bendrauti su aplinkiniu pasauliu tokiame lygyje, kokiame tradicinėms kompiuterinėms sistemoms neįmanoma. Tai suteikia ne tik šansą jų saugumo priemones pastatyti į pavojų, bet ir platesnes galimybes kenkėjui, jeigu tik jo ataka pavyksta. Nors ir ne robotų sistemos gali turėti galimybę daryti kažkokį poveikį aplinkai, jis paprastai būna labai ribotas. Jeigu robotui suteiktume tokį manipuliatorių kaip pvz., ranka, jam taptų įmanoma atlikti beveik viską, ką gali atlikti žmogus. Robotai netgi sugeba patys save modifikuoti – pavaros pagalba pakoreguoti vieną iš savo kamerų, prijungti tyčia atjungtą jutiklį, galbūt netgi tyčia sugadinti savo paties saugumo įrangą. Daugumoje situacijų yra priimta, kad jeigu turima fizinė prieiga prie kompiuterio, tuo pačiu gaunama prieiga prie jame esančių duomenų. Tuomet robotas potencialiai turi priėjimą prie pats savęs.

Turėdamas tokias mobilumo galimybės, robotas gali pats save pasistatyti į pageidaujamą vietą. Jeigu dirbama prie savo roboto saugioje aplinkoje, pvz., dirbtuvėse, robotas tikriausiai gali pats save persikelti į nesaugią aplinką, pvz., vaikų žaidimų kambarį ar net lauką. Robotui įmanoma tiesiog kažkur pačiam „nuvažiuoti“ ir pasiklysti. Jeigu robotas turi mikrofoną, jis gali save pasistatyti į tokią vietą, kad galėtų klausytis net tuomet, kai būsite įsitikinęs, jog robotą palikote jau toliau nei fizinėse įmanomo klausymosi ribose. Kenkėjas netgi galėtų pasivogti robotą (ir tuo pačiu „susirinkti“ visą informaciją, kurią įmanoma pasiekti roboto pagalba) tiesiog nuotoliniu būdu liepdamas jam išvažiuoti iš namų.

Tikimybių algoritmai. Pastarojo dešimtmečio moksliniai tyrimai parodė, kaip robotų kūrėjai gali pasinaudoti tikimybių teorijomis ir metodais tam, kad sukurtų itin tvirtus mobilius robotus. Mokslininkai sukūrė matematinius pagrindus ir algoritmus roboto judėjimo argumentams ir lokalizacijai nežinomose aplinkose, naudojantis dar ir prastais jutiklio duomenimis [11]. Jie pritaikė šiuos metodus robotams, kurie veda turus muziejuose [12] ir kuriant save vairuojančius automobilius [13].

Tačiau tikimybėmis paremta šių sistemų prigimtis sukelia problemų saugoje ir iškelia naujų klausimų. Kadangi robotas niekada nebūna tvirtai įsitikinęs su kuo bendrauja ar kur šiuo metu yra, priimant sprendimus, susijusius su sauga, būtina atsižvelgti į tikimybėmis paremtą prigimtį. Be to, šios sistemos yra patikimos ir stiprios nežinomoje ir triukšmingoje aplinkoje, bet dauguma jų taip ir nebuvo išbandytos naudojant priešišką modelį, kur kenkėjas bando apsimesti teisėtu vartotoju [2].

1.4. Piktavališkų atakų prieš robotus grėsmės

Nors kompiuteriams sukurtos saugumo idėjos ir mechanizmai gana gerai pritaikomi kompiuterių tinklams, robotai mums pateikia naują ir rimtesnį iššūkį. Roboto sistemos elgesys labiau painus tuo, kad jį nulemia daug daugiau kintamųjų. Išoriniam stebėtojui, net jeigu jis yra roboto kūrėjas, kalbant apie roboto misiją ir aplinką ne visada įmanoma būti užtikrintam, kad bus galima suprasti ir paaiškinti kiekvieną roboto elgsenos elementą. Iš tikrųjų, pristačius visus mechanizmus, kurie reikalingi palaikyti roboto saugumą, šis sudėtingas jo elgesio paaiškinimas gali pasirodyti dar painesnis – tuo įsitikinama paskaičius toliau. Taigi gali būti sudėtinga pastebėti robote kenkėjo ataką, ir dar sudėtingiau įvertinti tos atakos sėkmę [4].

Autonominių robotų tiesioginės ir netiesioginės grėsmės. Kadangi roboto elgesys iš esmės apskaičiuojamas sistemos apdorojimo ištekliuose akimirksniu, pasinaudojant esamos atminties būklės informacija, įvesties duomenimis iš jutiklių ir komunikacijų ištekliais, naudinga atskirti dvi skirtingas grėsmių rūšis: tiesioginė grėsmė – kai kenkėjo veiksmai siekia tiesioginio ir greito efekto, ir netiesioginė grėsmė – kai iš anksto kenkėjo numatyti ir atlikti veiksmai pakeičia roboto vidinę būseną, kas gali baigtis „neteisingu“ roboto elgesiu. Pavyzdžiui, dvi tiesioginės grėsmės, kurias kenkėjas galėtų panaudoti prieš ginkluotą, kovojimui skirtą robotą, galėtų būti: (1) klaidingų komandų siuntimas jam ir (2) savo paties galių paslėpimas tam, kad robotas patikėtų, jog kenkėjas negrėsmingas ir „draugiškas“. Pirmuoju atveju netiesiogine grėsme taps roboto aukšto lygio tikslų struktūros keitimas, o antruoju – bandymas iškraipyti roboto pasaulio suvokimo modelį. Tiesioginė grėsmė yra artimai susijusi su mechanizmu, pagal kurį robotas ir kenkėjas bendrauja tarpusavyje, ir menkiausia tiesioginė grėsmė gali sukelti didelį kiekį netiesioginių grėsmių [4].

*Tiesioginės grėsmės tarpininkaujant **jutikliui**:*

1. **aptikimo vengimas**: kenkėjas bando manipuliuoti jutiklio įvesties duomenimis tam, kad jutiklis nepastebėtų kenkėjo egzistavimo;
2. **trukdymas**: kenkėjas bando manipuliuoti jutiklio įvesties duomenimis tam, kad jutikliui nepavyktų perduoti informacijos apie kenkėjo ketinimus. Sistema pastebi, kad jutiklis neperduoda pageidaujamos informacijos, bet nebūtinai priskiria tai prie kenkėjo veiksmų. Jutiklis pradeda veikti normaliu rėžimu vos trukdymas pasibaigia;

3. **apgavystė**: kenkėjas bando manipuliuoti jutiklio įvesties duomenimis tam, kad jis pateiktų tikėtiną, bet neteisingą informaciją. Sistema nepastebi, kad teikiama informacija yra neteisinga ir priimdama sprendimus vis tiek ja remiasi;
4. **atjungimas**: kenkėjas bando manipuliuoti jutiklio įvesties duomenimis tam, kad suprastintų jutiklio veikimą iki tol, kol jis nebegalės tinkamai funkcionuoti, ir net pasibaigus atakai nebegrįžtų į teisingą rėžimą;
5. **grįžtamasis ryšys**: kenkėjas bando aptikti ir suprasti, kokiu elgesiu ir ženklais jutiklis gali išsiduoti, pavyzdžiui, kad kenkėjo buvimą pavyko pastebėti;
6. **charakterizavimas**: kenkėjas bando sužinoti jutiklio pajėgumo ribas tam, kad galėtų nuspėti ar tam tikrose situacijose jutiklis jį užfiksuos. Tokiu būdu kenkėjui galima pasirūpinti atsakomosiomis priemonėmis.

*Tiesioginės grėsmės tarpininkaujant **vykdytojams (valdomoms galūnėms)**:*

1. **atjungimas**: kenkėjas atlieka veiksmą, kuris sumažina arba sustabdo vykdytojų (valdomų galūnių) veikimą, ir jie tampa neveiksnūs tol, kol bus pataisyti;
2. **paralyžiavimas**: kenkėjas atlieka veiksmą, kuris sumažina arba sustabdo vykdytojų (valdomų galūnių) veikimą, bet jie tampa vėl veiksnūs tuomet, kai kenkėjas baigia savo veiklą;
3. **charakterizavimas**: kenkėjas bando sužinoti vykdytojų (valdomų galūnių) pajėgumo ribas tam, kad galėtų nuspėti kokios reakcijos iš vykdytojų (valdomų galūnių) galima tikėtis tam tikrose situacijose. Tokiu būdu kenkėjui galima pasirūpinti atsakomosiomis priemonėmis.

*Tiesioginės grėsmės tarpininkaujant **komunikacijoms**:*

1. **aptikimas**: kenkėjas bando aptikti vykstantį komunikavimą (perdavimą arba priėmimą);
2. **trukdymas**: kenkėjas bando neleisti sėkmingai įvykti komunikavimui. Sistema pastebi, kad komunikavimo posistemė neperduoda ar nepriima pageidaujamos informacijos, bet nebūtinai priskiria tai prie kenkėjo veiksmų. Sėkmingai veikti komunikacijų posistemei pavyksta vos pasibaigus trukdymui;
3. **apgavystė**: kenkėjas siunčia duomenis, kuriuos robotas priimdamas tiki, jog jie buvo perduoti „draugiško“ transliuotojo. Patys duomenys gali būti pilnai sukurti kenkėjo, pasisavinti iš „draugiško“ transliuotojo ir šiek tiek pakeisti, arba palikti originalūs. Kenkėjas kartais net gali nežinoti kas yra užšifruota žinutėje;
4. **atjungimas**: kenkėjas bando sugadinti komunikavimo posistemę taip, kad ji nebegalėtų tinkamai funkcionuoti, ir net pasibaigus atakai nebegrįžtų į teisingą rėžimą;
5. **charakterizavimas**: kenkėjas bando sužinoti komunikacijų posistemės pajėgumą tam, kad pasirūpintų atsakomosiomis priemonėmis;

6. **slaptas klausymasis**: kenkėjas bando perimti komunikacijų valdymą ir mėgina iššifruoti;
7. **srauto analizė**: kenkėjas bando perprasti komunikavimo srauto šabloną (nebūtinai bandydamas iššifruoti komunikacijų turinį) ir tokiu būdu gauti daugiau informacijos apie roboto galimybes ir misiją.

*Tiesioginės grėsmės tarpininkaujant **apdorojimo elementams**:*

Kadangi fizinis įsiskverbimas į prietaisą yra neleidžiamas ir nepasiekiamas, autonominio roboto apdorojimo elementai neturi tiesioginio kontakto su išorinėmis jėgomis savo veikimo metu; visas bendravimas vyksta tarpininkaujant jutikliams, valdomoms galūnėms ir komunikacijų elementams, kaip buvo apžvelgta aukščiau. Tiesioginė ataka įmanoma tik dar prieš prasidedant roboto naudojimui, tačiau tik per pvz., Trojos arkliuko implantavimą į programinę įrangą. Kadangi autonominio roboto elgesys gali būti sudėtingesnis nei „tradicinių“ sistemų (jo vidinės būsenos būtų mažiau suprantamos išoriniam stebėtojui), atsargiai implantuotas Trojos arkliukas būtų aptinkamas tik vykstant labai kruopščiai „apkrėsto“ įrankio daugiasluoksnei analizei [4].

Perimtos grėsmės. Jei roboto apdorojimo elementas yra sumodeliuotas kaip variklis, kuris veikia pagal (1) išorinę aplinką ir (2) iš misijos kilusios tikslo struktūros esminį atvaizdavimą, tuomet sistemos būklė iš esmės yra išreikšta šių dviejų duomenų struktūrų. Atakos, besikėsinančios į šių struktūrų saugumą ir vientisumą, gali būti tokios:

1. kenkėjas tyčia siekia pakeisti roboto tikslo struktūrą;
2. kenkėjas tyčia siekia pakeisti roboto aplinkos modelį;
3. kenkėjas siekia sužinoti roboto aplinkos modelio struktūrą;
4. kenkėjas siekia sužinoti roboto tikslo struktūrą.

Tiesioginių grėsmių mechanizmų keliai, kurie buvo apžvelgti aukščiau, gali būti naudojami kenkėjo tam, kad pasiektų bet kuriuos iš šių netiesioginių tikslų.

1.5. Protokolų, naudojamų robotuose ir „daiktų internete“, apžvalga

Saugumas turėtų būti neatsiejama daiktų interneto (angl. *Internet of Things, IoT*) dalis. Tiek skaitmeniniame, tiek globaliame pasaulyje saugumo reikšmė pastebima kiekvieną dieną – nesvarbu ar bandoma pervesti pinigų draugui, apsipirkinėjama internetinėje parduotuvėje ar bandoma prisijungti prie debesyse (angl. *cloud*) saugomų asmeninių dokumentų. Daiktų interneto tikslas ir yra taip sujungti kiekvieną daiktą į tinklą, kad naudojimasis jais taptų produktyvesnis, taip suteikiant daugiau komforto ar pagerinant kasdieninį gyvenimą tiek darbe, tiek namuose. Bet vos pabandžius sujungti tokius objektus kaip automobiliai, namai ir namų apyvokos daiktai, susiduriama su galimybe, kad nutekės daug mūsų asmeninės informacijos. Pavyzdžiui, jeigu namuose turimas vaizdo stebėjimo robotas bus naudojamas išvykus atostogauti, o jis veikdamas kiekvieną kartą pasiųs pranešimą, jeigu namuose kažkas vaikšto – viskas skamba puikiai, tačiau ne pati geriausia idėja būtų ta informacija pasidalinti su

ilgapirščiais. Tokia informacija akivaizdžiai nėra skirta visuomenei, todėl turėtų būtų apsaugota informacijos saugos pamatais – konfidencialumu, vientisumu ir prieinamumu.

Saugumo iššūkiai daiktų internete. Nekyla jokios abejonės, kad saugumas šioje sferoje būtinas, tačiau norint tai įgyvendinti daiktų internete, susiduriama su sunkumais. Saugumas ir taip ne retai lieka kompromisu tarp visiškos saugos ir patogaus, greito naudojimosi, daiktų internete tas kompromisas įgauna dar didesnę reikšmę. Daiktų interneto prietaisai labai dažnai yra suvaržyti ribotų skaičiavimo išteklių, atminties kiekio ir energijos trūkumo. Taigi naudotis pačiais saugiausiais kriptografiniais algoritmais tampa neįmanoma, nes jie tiesiog reikalauja daugiau išteklių negu visi namuose esantys daiktų interneto įtaisai turi kartu sudėjus.

Dažniausiai pasitaikančios saugos spragos daiktų internete. Šioje dalyje apžvelgiamos saugumo grėsmės ir spragos daiktų internete tam, kad būtų aiškiau į ką reikėtų atžvelgti dėmesį bandant padidinti protokolo saugumą. Lentelėje pateiktos septynios galimos saugumo grėsmes, kurios suskirstytos pagal sluoksnius [30]:

1.1 lentelė. Saugumo grėsmių analizė pagal OSI modelio sluoksnius

Sluoksniai	Galimos saugumo grėsmės
Fizinis įrenginys	Saugumo parametrų išgavimas
Taikymo (programų)	Programinės įrangos ataka atnaujinimo metu
Transporto	Neteisėtas informacijos perėmimas
	Susidūrimo viduryje ataka
Tinklo	Neteisėtas informacijos perėmimas
	Susidūrimo viduryje ataka
	Atsisakymo aptarnauti ataka
	Maršrutizavimo ataka
Fizinis	Atsisakymo aptarnauti ataka

Saugumo grėsmes galima apytiksliai suskirstyti į dvi kategorijas: fizinė (vietinė) ataka ir ne fizinė (nuotolinė) ataka. Fizinės atakos įvyksta kai užpuolėjas bando fiziškai įsivežti, priartėti prie neapsaugoto daikto ir sukompromituoti jį skirtingais metodais [31]:

Programinės aparatinės įrangos ataka atnaujinimo metu. Praėjus tam tikram įrenginio veikimo laikui, daikto programinė įranga gali būti atnaujinama, siekiant geresnio veikimo ar naujų funkcijų įtvirtinimo. Atnaujinimo metu, programišius gali pasinaudoti galimybe pakeisti jį kenksminga programine įranga. Jeigu tokia ataka pavyksta, daikto veikimas gali būti įtakojamas ir išnaudojamas negeriems tikslams.

Saugumo parametrų išgavimas. Daiktai, kurie naudojami namų aplinkoje, yra paprastai niekaip fiziškai neapsaugoti, todėl juos užpuolėjas lengvai gali pagrobti. Fiziškai savo rankose turėdamas prietaisą, programišius gali pabandyti išgauti tam tikrą saugumo informaciją, tarkim, slaptažodžius ar šifro kodus. Vos tik nors vienas slaptažodis sukompromituojamas, visas tinklas taip pat gali atsidurti pavojuje.

Atsisakymo aptarnauti ataka. Atsisakymo aptarnauti ataką lengvai galima sukelti fiziškai užkemšant komunikacijos kanalus.

Visi aukščiau aprašyti atakos metodai priklausomi nuo fiziškai šalia esančio programišiaus. Todėl visos saugos priemonės, kurių privaloma imtis norint apsaugoti įrenginius ir daiktų internetą nuo tokių atakų, apima platesnę sritį negu vien interneto protokolo saugumas. Ne fizinės (nuotolinės) atakos apima tokius atvejus [30, 31]:

Neteisėtas informacijos perėmimas (angl. *Eavesdropping*). Įvedus prietaisą į tinklą, jis gali tapti prastai apsaugotas nuo neteisėto informacijos parėmimo, kitaip vadinamo pasiklausymu – jeigu saugumo parametrai ar raktų apsikeitimas vyksta be jokios apsaugos, programišiui gali tapti lengvai įmanoma nugvelbti visą informaciją, keliaujančią tarp komunikuojančių pusių. Bet net ir apsaugotas tinklas gali būti nepajėgus apsisaugoti nuo pasiklausymo – pavyzdžiui, jeigu sesijos raktai būtų sukompromituoti, kas gali atsitikti po ilgesnio laiko naudojantis tuo pačiu raktu jo neatnaujinant.

Susidūrimo viduryje (angl. *Man-in-the-Middle*) ataka. Susidūrimo viduryje ataka tampa įmanoma bendravimo fazėje, pavyzdžiui kai atvirai apsikeičiama raktais, o raktų apsikeitimo algoritmo saugumas priklauso nuo to, ar jokia trečioji šalis negali neteisėtai šios informacijos gyvai klausytis ir perimti. Vėliau šios atakos įmanomos tik jeigu naudojami anoniminiai susijungimai.

Maršrutizavimo (angl. *Routing*) ataka. Šios rūšies atakas galima išskirti į dvi rūšis: a) Smegduobės (angl. *sinkhole*) ataka, kur įsilaužėlis apsimeta turintis aukštos kokybės maršrutą, kas jam leidžia daryti ką tik nori su praeinančiais paketais; b) Persiuntimas atsirenkant, kur įsilaužėlis gali išmesti arba persiųsti tik atsirinktus paketus.

Atsisakymo aptarnauti ataka. Programišius gali nesustodamas siųsti užklausas, prašydamas įvykdyti specifines užduotis, ir tokiu būdu išnaudoti resursus. Tai ypatingai pavojinga daiktų internete, kadangi daiktų interneto įrenginiai paprastai turi itin ribotą atmintį ir skaičiavimo galimybes.

1.5.1. CoAP protokolas

Suvaržytų aplikacijų protokolas (angl. *Constrained Application Protocol, CoAP*) [14] yra specializuotas tinklo perdavimo protokolas, skirtas išteklių suvaržytiems, prie tinklo prijungtiems įrenginiams, taip pat negalingiems bei apribotiems tinklams. Tinklo mazgais dažnai tampa 8-bitų mikrovaldikliai su ribota darbine atmintimi, procesoriumi ir energijos ištekliais, o tuo tarpu tokie apriboti tinklai kaip 6LoWPANs (interneto protokolo versija Nr. 6, veikianti negalinguose bevieliuose asmenininiuose tinkluose (angl. *IPv6 over Low-Power Wireless Personal Area Networks*)) dažnai

pasižymi dideliu paketų klaidų rodikliu ir tipiniu nedideliu pralaidumu. Šis protokolas yra sukurtas veikimo principui prietaisas prietaisui (angl. *machine-to-machine, M2M*) naudojimui, pavyzdžiui, išmaniajam elektros energijos valdymui namuose ar pastatų automatizavimui.

CoAP suteikia galimybę naudotis užklausos – atsakymo bendravimo modeliu tarp programos galutinių taškų (prietaisų), palaiko įterptinę naujų prietaisų ir išteklių paiešką, apima pagrindinius tinklo konceptus, tokius kaip universalius resursų identifikatorius (angl. *Uniform Resource Identifiers, URIs*) ir interneto medijų rūšis. CoAP yra sukurtas taip, kad būtų galima lengvai kurti sąsają su hipertekstų persiuntimo protokolu (angl. *Hypertext Transfer Protocol, HTTP*) paprastam integravimuisi į tinklą tais atvejais, jeigu susiduriama su specialiais reikalavimais kaip duomenų perdavimas keliems įrenginiams vienu metu (angl. *multicast*) ar neišvengiamu paprastumo reikalavimu suvaržytose aplinkose.

Vienas iš pagrindinių tikslų kuriant CoAP protokolą buvo kiek įmanoma sumažinti perduodamos žinutės pridėtinę informaciją (angl. *message overhead*) tam, kad pavyktų daug resursų kainuojantis IPv6 paketų fragmentavimas, tad tokiu būdu tausojami tokie apriboti tinklai kaip 6LoWPAN. CoAP naudoja dviejų rūšių žinučių tipus – užklausą ir atsakymą, kuriuos paremtos paprastu, dvejetainiu, fiksuoto dydžio antraštės formatu, po kurio galimai seka toks pasirinkimo formatas kaip Rūšis – Ilgis – Vertė (angl. *Type-Length-Value, TLV*) ir siunčiama žinutė (angl. *payload*). Siunčiamos žinutės ilgis priklauso nuo datagramos ilgio, ir kuomet naudojamasi vartotojų datagramų protokolu (angl. *User Datagram Protocol, UDP*), ji privalo tilpti į vieną datagramą [15].

CoAP pasižymi tokiomis pagrindinėmis savybėmis:

- tai yra suvaržyto tinklo protokolas, kuris atitinka M2M reikalavimus;
- naudojamas kartu su UDP, su laisvai pasirenkamu patikimumu palaikant duomenų perdavimą vienam ar keliems įrenginiams vienu metu;
- nedidelė antraštės pridėtinė informacija;
- galimybė užsisakyti pranešimus per paskelbimo – užsakymo mechanizmą;
- nesudėtingos įgaliotojo serverio (angl. *proxy*) ir podėliavimo (angl. *caching*) galimybės;
- galimybė naudoti kartu datagramų transporto sluoksnio apsauga (angl. *Datagram Transport Layer Security, DTLS*).

1.5.2. MQTT protokolas

MQTT – žinučių eilės sudarymo telemetrijos transporto (angl. *Message Queuing Telemetry Transport Protocol*) [16] palengvintas, supaprastintas žinučių perdavimo protokolas, kuris suteikia galimybę išplatinti informaciją net ribotų išteklių bei tinklų pajėgumų suvaržytiems klientams. Šis protokolas naudoja užsiprenumeravimo – leidimo komunikacijų metodą ir tampa vis dažniau naudojamas M2M komunikavimui, tuo pat metu greitai populiarėjantis ir „daiktų internete“. MQTT

suteikia galimybę net ir labiausiai suvaržytų išteklių prietaisams dalintis informacija pasirinktąja tema, siunčiant žinutes per serverį, kuris veikia kaip MQTT žinučių brokeris. Brokeris persiunčia informaciją tiems klientams, kurie yra užsiprenumeravę būtent tą temą. Norint paaiškint kuo paprasčiau, reikėtų įsivaizduoti temą kaip hierarchinę failų sistemą, kur klientas gali užsiprenumeruoti tam tikrą hierarchijos lygį arba panaudoti specialius simbolius, jeigu pageidautų užsiprenumeruoti kelis lygius.

MQTT protokolas yra vienas iš geriausių pasirinkimų, jeigu tenka dirbti nepatikimame bevieliame tinkle arba jeigu leidžiamas persiunčiamų duomenų kiekis yra itin ribotas. Net jeigu nutrūktų ryšys tarp užsiprenumeravusio kliento ir brokerio, brokeris išsaugos žinutes ir išsiųs jas prenumeratoriui, vos tik jam pavyks prisijungti prie ryšio. Jeigu ryšys nutrūktų tarp žinutes publikuojančio kliento, brokeris gali nutraukti komunikavimą, tuo pačiu prenumeratoriams išsiųsdamas „paskutinės valios“ (angl. *last will*) išsaugotą žinutę su instrukcijomis iš publikuojančio kliento [15].

MQTT sesiją galima padalinti į keturias fazes: susijungimas, autentifikavimas, bendravimas ir nutraukimas. Klientas pradeda sukurdamas TCP / IP (transporto valdymo protokolas / interneto protokolas, t.y. protokolų rinkinys, aprašantis duomenų persiuntimą tarp įvairių tipų kompiuterių ir operacinių sistemų (angl. *Transport Control Protocol / Internet Protocol*)) susijungimą su brokeriu, naudojant standartinį prievadą, jeigu nenustatyta kitaip. Standartiniai prievadai yra: 1883 prievadas nešifruotai komunikacijai, ir 8883 prievadas šifruotam bendravimui, naudojant SSL / TLS (saugiųjų jungimų lygmens protokolas / transporto lygmens protokolas, skirtas saugumui užtikrinti TCP / IP tinkluose (angl. *Secure Socket Layer / Transport Layer Security*)). Vykstant SSL / TLS rankos paspaudimui, autentiškumo patikrinimui paprastai naudojamas serverio sertifikatas. Kadangi MQTT protokolas taikosi prie prietaisų, kurie turi itin suvaržytus išteklius, SSL / TLS apsauga ne visuomet yra pageidautina ar net įmanoma. Tokiais atvejais autentifikavimui naudojamas tiesiog vartotojo vardas su slaptažodžiu, kurie išsiunčiami nešifruotu tekstu ir matomi visiems, paslapčia stebintiems susijungimo / susijungimo patvirtinimo (angl. *connect / connack*) paketų apsikeitimą. Kai kurie brokeriai, ypač sukurti viešose interneto platformose, priima tiesiog visus anoniminius klientus – tuomet vartotojo vardas su slaptažodžiu programiniame kode paliekami tušti.

MQTT protokolas gali pasiūlyti tris paslaugų tiekimo lygius QoS (angl. *Quality of Service*), nuo kurių priklausys, kaip bus elgiamasi su siunčiamais ir gaunamais duomenimis. Nors aukštesni paslaugų tiekimo lygiai yra patikimesni, bet jie reikalauja didesnio naudojamo duomenų srauto, tad pasirinkimą atlieka prenumeratoriai pagal savo pageidavimus.

Pats paprasčiausias paslaugų tiekimo lygis yra nepatvirtinamas. Šis paslaugų tiekimo lygis naudoja paskelbimo (angl. *publish*) paketų seką – žinučių siuntėjas pateikia pranešimą brokeriui vieną kartą, tuomet brokeris vieną kartą perduoda tą pranešimą visiems prenumeratoriams. Naudojant šį lygį, nėra jokio būdo užtikrinti, kad pranešimas pasiekė brokerį sėkmingai, o ir brokeris tos žinutės

neišsaugo. Dėl to šis paslaugų tiekimo lygis dar vadinamas „daugiausiai vieną kartą“ (angl. *QoS0, at most once*) arba „paleidi ir pamiršti“.

Antrasis paslaugų tiekimo lygis yra patvirtinamas. Šis paslaugų tiekimo lygis naudoja paskelbimo / paskelbimo patvirtinimo (angl. *publish / puback*) paketų seką tarp žinučių siuntėjo ir brokerio, taip pat ir tarp brokerio ir prenumeratorių. Patvirtinimo paketas užtikrina, kad žinutės turinys buvo gautas, o jeigu patvirtinimas nebuvo gautas per tam tikrą laiką, pakartojimo mechanizmas vis siųs žinutę. Dėl to gali pasitaikyti, kad ta pati žinutė bus išsiųsta keletą kartų. Dėl to šis paslaugų lygis dar vadinamas „bent vieną kartą“ (angl. *QoS1, at least once*).

Trečiasis, sudėtingiausias paslaugų tiekimo lygis yra garantuotas. Šis paslaugų tiekimo lygis naudoja dvi paketų poras. Pirmoji paketų pora yra paskelbimo / paskelbimo gavimo (angl. *publish / pubrec*), o antroji pora yra paskelbimo perdavimo / paskelbimo pabaigos (angl. *pubrel / pubcomp*). Šių dviejų paketų pora užtvirtinama, kad nepriklausomai nuo pakartojimų skaičiaus, žinutė bus pristatyta tik vieną kartą. Dėl to šis paslaugų lygis dar vadinamas „tiksliai vieną kartą“ (angl. *QoS2, exactly once*).

MQTT protokolo sauga. MQTT protokole saugumas padalintas keliuose sluoksniuose, iš kurių kiekvienas padeda išvengti skirtingų rūšių atakų. Kadangi šio protokolo tikslas yra parūpinti palengvintą (angl. *lightweight*) ir nesudėtingą komunikacijų protokolą daiktų internetui, todėl ir saugumo mechanizmų paminėta nėra daug. Toliau apžvelgiami siūlomi saugumo sprendimai kiekvienam sluoksniui:

1. tinklo sluoksnis (angl. *Network Level*). Vienas iš būdų kaip užtikrinti saugų ir patikimą ryšį tarp kliento ir brokerio (serverio), yra kaip pagrindą naudoti fiziškai kenkėjams neprieinamą tinklą arba virtualų privatų tinklą (angl. *Virtual Private Network, VPN*). Tai tiktų šliuzų aplikacijoms (angl. *Gateway Applications*), kur vienoje pusėje šliuzas prijungtas prie prietaisų, o kitoje pusėje prie brokerio per VPN;
2. transporto sluoksnis (angl. *Transport Level*). Šiame sluoksnyje tikslas yra užtikrinti konfidencialumą, tad daugumoje atvejų bandoma naudoti TLS / SSL transportavimo šifravimui. Žinoma, tokiu būdu gauname užtikrintą būdą, kad niekas negalės skaityti perduodamų žinučių bei autentifikuoti abi puses, naudojant kliento autentifikavimą sertifikatų pagalba, tačiau kadangi daiktų interneto įrenginiai dažnai turi ribotus išteklius, tai tampa neįmanoma;
3. Taikymo (programų) sluoksnis (angl. *Application Level*). Transporto sluoksnyje buvo galima užtikrinti, kad komunikacija yra šifruota, o tapatybė autentifikuojama – MQTT protokolas parūpina kliento identifikatorių ir vartotojo vardo / slaptažodžio kredencialus, kurie irgi gali būti naudojami autentifikuoti prietaisus taikymo sluoksnyje. Dar viena galimybė yra naudoti tikrosios perduodamos žinutės (angl. *payload*) šifravimą taikymo

sluoksnyje tam, kad būtų užtikrinta parduodamos informacijos sauga net ir nenaudojant visaverčio transporto šifravimo.

MQTT naudojasi TCP kaip transporto protokolu, tad pagal nutylėjimą ryšys nesinaudoja šifruotomis komunikacijomis. Norint užšifruoti visą MQTT komunikaciją, daugelis MQTT brokerių leidžia naudoti TLS šalia paprasto TCP. Tad jeigu daiktų internete naudojame „MQTT CONNECT“ paketo „vartotojo vardas“ ir „slaptažodis“ laukelius autentifikavimui ir autorizavimui, patartina naudoti TLS. 8883 prievadas tapo standartu norint užtikrinti saugų MQTT ryšį.

Žinoma, pats didžiausias trūkumas naudojant TLS kartu su MQTT yra išteklių kaina, kalbant apie centrinį procesorių (angl. *Central Processing Unit, CPU*) ir komunikacijų apkrovą. Nors papildoma CPU apkrova brokeriui nereikšminga, tai tampa didele problema ribotų išteklių prietaisams, kurie nesukurti apdoroti intensyvias skaičiavimo užduotis. Tokiais atvejais galima panaudoti tokius metodus kaip sesijos atgavimas (angl. *Session Resumption*), kurie gali drastiškai pagerinti TLS darbą. Sesijos atgavimas – tai metodas, leidžiantis pakartotinai panaudoti jau aptartą TLS sesiją po pakartotino prisijungimo prie serverio, tokiu būdu klientui ir serveriui nebebūtina pakartoti pilno TLS rankos paspaudimo. Verta paminėti, kad ne visos TLS bibliotekos realizavusios sesijos atgavimą. Sesiją galima grąžinti dviem būdais:

- sesijos ID: Serveris išsaugo slaptą būseną kartu su sesijos ID. Kai klientas pakartotinai prisijungia, jis vėl pateikia sesijos ID ir sesija gali būti pratęsta;
- sesijos bilietas (angl. *Session Tickets*): Serverio slapta būsena perduodama klientui, kuri būna užšifruota su slaptu raktu, kurį žino tik serveris. Klientas, bandydamas pakartotinai prisijungti, nusiunčia šį bilietą atgal serveriui. Jeigu serveris gali iššifruoti bilieto turinį, sesija pratęsiama tokios būsenos, kokia nurodyta slaptame biliete.

Su didele komunikacijų apkrova galime susidurti TLS rankos paspaudime (angl. *TLS Handshake*), jeigu MQTT kliento susijungimai planuojami būti tik trumpalaikiai. Nors kokio tinklo pralaidumo (angl. *bandwidth*) TLS rankos paspaudimui prireiks priklauso nuo daug faktorių, kai kurie bandymai parodė [17], kad sukuriant naują TLS susijungimą gali prireikti kelių kilobaitų pralaidumo. Kadangi naudojant TLS kiekvienas paketas užšifruojamas, ryšiu keliaujantys paketai irgi turės papildomą apkrovą lyginant su nešifruotais paketais.

Jeigu TLS įgyvendinti kartu su MQTT visiškai neįmanoma, galima bandyti naudoti tikrosios perduodamos žinutės (angl. *payload*) šifravimą leidžiamoms (angl. *publish*) žinutėms, taip pat naudoti maišą ar šifruoti slaptažodį kliento prisijungimo (angl. *connect*) žinutėje, taip pat išbandyti TLS – PSK (iš anksto pasidalintų raktų (angl. *Pre-Shared Key*)) šifrų komplektus (angl. *Cipher Suites*), kadangi juos naudojant galima išvengti daug CPU resursų naudojančių viešojo rakto operacijų.

1.5.3. HTTP protokolas

HTTP, šiuo metu yra vienas iš populiariausių internete naudojamų taikomųjų programų lygmens protokolų. HTTP yra asimetrinis užklausos – atsakymo bei kliento – serverio architektūros protokolas. Jo veikimo principas paremtas kliento užklausos siuntimu serveriui, kuris pagal užklausą suformuluoja ir pateikia atsakymą. HTTP protokolas yra neįsimenantis – kitaip tariant, dabartinė užklausa nežino nieko apie praėjusias užklausas. HTTP palaiko skirtingus duomenų tipus, o tai leidžia sukurti sistemas, nepaisant perduodamų duomenų tipo. Pačios populiariausios užklausos yra keturių rūšių:

- gavimo užklausa (angl. *get*). Naudojant šio tipo užklausą, ištraukiami duomenys iš jau egzistuojančių išteklių. URL (universalusis adresas (angl. Universal *Resource Locator*)) saugoma visa reikalinga informacija, kuria, norėdamas atsakyti į užklausą, pasinaudoja serveris;
- sukūrimo užklausa (angl. *post*). Paprastai šio tipo užklausa yra papildyta informacija, kuri aprašo naujai sukuriamus duomenų išteklius;
- atnaujinimo užklausa (angl. *put*). Šio tipo užklausa atnaujina jau egzistuojančių išteklių parametrus ar informaciją;
- ištrynimo užklausa (angl. *delete*). Naudojant šio tipo užklausą, galima ištrinti egzistuojančius informacijos išteklius.

Nors beveik absoliuti dauguma programuotojų HTTP protokolą bent jau žino, jeigu ne gerai išmano, jis nėra pats populiariausias protokolas tarp „daiktų interneto“ projektų. Nors prietaisai gali siųsti savo duomenis kaip HTTP užklausą, ir iš serverio laukti gavimo patvirtinimo, šis komunikavimo metodas turi tam tikrų apribojimų:

- HTTP yra sinchroninis protokolas – klientas laukia serverio atsakymo, tad daiktų internete ar namų automatikoje tai gali sukelti problemas dėl didelio kiekio negalingų prietaisų, kurie siųstų nors ir nedideles, bet dažnai pasikartojančias žinutes;
- HTTP yra vienpusis protokolas. Klientas privalo inicijuoti komunikavimą, o daiktų internete prietaisai ar jutikliai įprastai atlieka kliento vaidmenį, kas reikštų, jog jie nebegali pasyviai iš tinklo gauti komandas.
- HTTP yra 1 – 1 protokolas. Klientas duoda užklausą, o serveris atsako, tad dėl to tampa sudėtinga (ir resursų prasme brangu) tą patį pranešimą išsiųsti visiems tinklo prietaisams, kas daiktų internete yra itin dažnai pasitaikanti situacija.
- Galiausiai, HTTP protokolas, palyginus jį su tokiais palengvintais protokolais kaip MQTT ar CoAP, atrodo sudėtingas ir netgi griozdiškas dėl antraščių ir taisyklių – žinoma, jis kur kas labiau modifikuojamas ir pritaikomas įvairiems projektas, tačiau gali tapti neįgyvendinamu itin suvaržytose aplinkose.

1.5.4. SSL / TLS įgyvendinimas mikrovaldikliuose

Teorinis SSL / TLS pritaikymas 8-bitų sistemoms. Timothy Stapko teigia [18], kad SSL / TLS įgyvendinimas įmanomas ne tik 32-bitų sistemose – su trupučiu kūrybiškumo ir atidaus programavimo jis puikiai naudojamas ir 8-bitų mikroprocesoriuose. Sukurti SSL / TLS pritaikymą mažesnėms, įterptinėms sistemoms yra didelis iššūkis, kadangi nėra jokio „įterptinio“ SSL / TLS protokolo. Kalbant apie procesoriaus ciklus ir atminties sunaudojimą, šifravimo operacijos „kainuoja“ labai daug. Šis protokolas buvo sukurtas galingiems mechanizmams, kurie neturi išteklių apribojimų. Tačiau perspektyvus SSL / TLS įgyvendinimas, itin atidžiai realizuotas, iš tiesų gali pridėti tik 50 KB dydžio kodo pėdsaką programoje, ir užimti mažiau nei 20 KB vietos duomenų saugojimo vietoje. Daugumai įterptinių sistemų SSL / TLS reikės tik serverio pusėje, kadangi jos bus prieinamos naudojantis interneto naršykle (klientu). Tai leidžia atsikratyti kliento pusėje esančiu SSL / TLS kodu, įskaitant ir intensyviai resursus naudojantį sertifikato – autentifikavimo kodą, bei šakninius sertifikatus, kuriuos išduoda sertifikuota valdžia. Kai kiekvienas sertifikatas gali užimti nuo 1 iki 2 KB, tai tampa nuolatiniu ir akivaizdžiai pastebimu vietos sutaupymu. Taip pat galima pasinaudoti struktūriniais panašumais tarp 5 SSL versijos ir TLS – parašant protokolo kodą tokį, kuris veiktų su abiem protokolais. Beveik 90% viso kodo gali dalintis abu protokolai [18].

Palaikomo algoritmo pasirinkimas taip pat turi labai didelę įtaką kodo dydžiui. AES (pažangus šifravimo standartas (šifravimo algoritmas) (angl. *Advanced Encryption Standard*)) ir 3DES (trigubas informacijos užšifravimo algoritmas (simetrinio rakto blokinis šifras) (angl. *Triple Data Encryption Algorithm*)) yra sudėtingi šifravimo algoritmai, palyginus su RC4 (Rivest šifras Nr. 4 (srauto šifras) (angl. *Rivest Cipher 4*)), kuris labai paprastas ir reikalauja tik keleto eilučių kodo. Neatsitiktinai RC4 pademonstruoja ir geresnes charakteristikas nei AES ar DES [19]. TLS specifikacijos (RFC 2246) reikalauja 3DES tam, kad būtų laikomasi reikalavimų, bet praktikoje dauguma interneto naršyklių ir SSL / TLS įgyvendinimų palaiko RC4, todėl vis šiek tiek nukrypstant nuo specifikacijų ir palaikant tik RC4, galima išsaugoti keletą kilobaitų vietos be praktiškai jokio blogo poveikio [18]. Tačiau šis variantas tinka tik „mėgėjiškoms“ robotų sistemoms, ir visiškai netiks kuriant šimtaprocentinio saugumo reikalaujančias sistemas, pvz., susijusias karine ar medicinos technika, nes yra įrodyta, jog RC4 vis dėlto įveikiamas [20].

Skaitmeniniai sertifikatai yra būtini SSL / TLS komunikacijoms, kadangi juose yra tiek viešasis raktas, tiek autentifikavimo informacija. Deja, visa ši informacija paverčia sertifikatą pakankamai dideliu – kiekvieną nuo 1 iki 2 KB. SSL / TLS protokolas apibrėžia, kad bet kuris SSL / TLS klientas ar serveris turėtų sugebėti palaikyti keletą sertifikatų. Tačiau praktikoje komunikacijai dažniausiai tereikia vieno sertifikato. Sumažinant sertifikatų saugojimo leidimą iki vieno sertifikato vienam prietaisui, galime sumažinti duomenų užimamą vietą [18].

Duomenų užimamos vietos minimizavimas. Geriausias būdas sumažinti užimamą vietą yra atidžiai sudaryti žinučių apsaugojimo algoritmus ir minimizuoti buferio dydžius. Reikia sutalpinti visą žinutę bandant ją apdoroti ar patvirtinti prieš perduodant ją TCP susijungimui ar programai. Tai reiškia, kad įeinančios žinutės buferis turi būti mažiausiai 16 KB dydžio (didžiausio SSL įrašo dydžio, pagal SSL specifikacijas) tam, kad būtų galima įsitikinti, jog bus gauta visa žinutė. Tačiau išeinančios žinutės buferio dydis gali būti ženkliai sumažintas, kadangi išsiunčiamo įrašo dydis yra kontroliuojamas. Jis turi būti bent 2 KB ir savyje turėti sertifikato žinutę, kuri yra tarp 1 ir 2 KB dydžio. Minimalus buferis paaukoja šiek tiek įvykdymo greičio, kadangi mažesni įrašai gali būti mažiau efektyvūs našume, tačiau taip RAM sutaupymas pasirodo itin žymus. Buferio dydis lengvai parametrizuojamas, taip suteikiant programuotojui (ir netgi vartotojui) daugiau galios reguliuoti atminties panaudojimą derinant sistemos įgyvendinimą [18].

Pirminis rankos paspaudimas reikalauja daugiau buferio vietos negu pati sesija (jam reikia gauti sesijos raktus ir atlikti viešojo rakto operaciją), tad galima laikinai paskirti tam šiek tiek atminties. Tai efektyviai įmanoma atlikti įdiegiant paprastą rinkinį į statinę atmintį, iš kurios galima paskirti vietos buferiui.

Tinkamai užšifruojant ir iššifruojant žinutes, galima naudoti tik vieną buferį kiekvienai įvesčiai ir išvesčiai, vietoje bandymo naudoti atskirtus buferius užšifruotoms ir iššifruotoms žinutėms. Tai yra įmanoma, nes visi SSL / TLS palaikomi kriptografiniai algoritmai vienu metu veikia tik viename fiksuoto dydžio duomenų bloke, ir išvesties duomenys yra tokio pat dydžio kaip įvesties. Todėl galima daryti išvadą, kad užšifruoti ir iššifruoti duomenys yra tokio pat dydžio naudojant bet kokį algoritmą, todėl neteks susidurti su buferio perpildymu [18].

Veikimo efektyvumas. Kriptografija neabejotinai godi kalbant apie resursų naudojimą, bet SSL / TLS buvo sukurtas pasiekti maksimalų saugumą, o ne optimaliausią veikimą ir resursų panaudojimą. Bet egzistuoja keli būdai privesti SSL / TLS veikti greičiau ir sklandžiau naudojant ne tokius galingus procesorius. Kol kas „brangiausiai kainuojanti“ resursų atžvilgiu operacija bet kurioje SSL / TLS sesijoje yra viešojo rakto operacija. Tokie viešojo rakto algoritmai kaip RSA (Rivest – Shamir – Adleman) reikalauja tūkstančių operacijų su dideliais skaičiais (kiekvienas nuo 512 iki 2048 bitų). Be efektyvaus programavimo, vienintelis kitas būdas pagreitinti šias operacijas yra naudojant tik kažkokios rūšies techninės įrangos pagalbą. Kai kurios kompanijos kuria įrangą, skirtą būtent viešojo rakto šifravimui, pavyzdžiui kaip „Atmel“ [21], kuri suteikia įrangą specialiai RSA kriptografijos akceleracijai.

Dar vienas būdas yra bandyti naudoti centrinį procesorių pagal specialias instrukcijas, kurias sekant galima pagreitinti RSA algoritmą. Daugeriopo tikslumo daugybos instrukcijos gali pagreitinti RSA iki 10 kartų. Alfred Menezes savo knygoje [22] pateikia pavyzdžių kaip efektyviai vykdyti RSA.

Su tokiomis instrukcijomis, pirminio rankos paspaudimo laikas lengvai gali būti sumažintas nuo 30 sekundžių iki mažiau nei 3 sekundžių naudojant 44 MHz procesorių.

Sesijos sąlygų aptarimas iš naujo. Viena iš pačių svarbiausių sudėtingesnių SSL / TLS savybių, skirtų optimizuoti įterptinį SSL / TLS yra sesijos sąlygų aptarimas iš naujo. Sesijos susikūrimo metu, klientas ir serveris susitaria dėl sesijos ID, kuris yra didelis sveikasis skaičius, naudojamas unikaliam esamos sesijos identifikavimui. Tiek serveris, tiek klientas išsaugo sesijos raktus bei kitą sesijos informaciją laikinojoje atmintyje, ir, pasibaigus sesijai, kurį laiką tą informaciją saugo. Ši informacija gali būti panaudota pakartotinai pradėti anksčiau užbaigtą sesiją, taip išvengiant daug resursų naudojančios viešojo rakto operacijos [18].

Skirtingai nuo viešojo rakto kriptografijos, maišos algoritmai leidžiasi nesudėtingam programiniam optimizavimui. Tokie algoritmai kaip MD5 (žinučių santrauka Nr. 5 (maišos funkcija)) ir SHA1 (saugus maišos algoritmas Nr. 1 (maišos funkcija) (angl. *Secure Hash Algorithm 1*)) gali būti nesunkiai visiškai parašomi asembleryje. Jie palyginus greitesni už kitas kriptografines operacijas, todėl retai taps bet kokia efektyvaus veikimo silpnąja vieta [18].

1.6. MQTT ir HTTP protokolų funkcionalumo palyginimas

Didžiausias skirtumas tarp šių protokolų yra tai, kad MQTT priskiriamas prie supaprastintų ir palengvintų komunikavimo protokolų, kai HTTP laikomas sudėtingu ir iš pirmo žvilgsnio ne pačiu geriausiu protokolo pasirinkimu daiktų interneto prietaisams. Tačiau taip griežtai skirstyti negalima, nes priklausomai nuo prietaiso naudojimo situacijos bei pačios sistemos dizaino gali labiau tikti ir sudėtingesnis protokolas. Kartais labai patogu pasinaudoti jau sukurtomis, darbui paruoštomis aukštesnio lygio programų sąsajomis, kurias siūlo egzistuojančios MQTT protokolo bibliotekos, tačiau kartais naudingiau kodą kurti pačiam programuotojui nuo pat pradžių, naudojant žemesnio lygio programų sąsajas ar protokolus, pavyzdžiui, HTTP. Pasirinkimas priklausys nuo naudojimo situacijos – jeigu projektui įgyvendinti reikalinga dvipusė komunikacija, ar patys duomenys turi būti surenkami iš daugybės prietaisų ir persiunčiami dideliam kiekiui prenumeratorių, tokiu atveju naudingiau rinktis MQTT. Jeigu reikalingas paprasčiausias AJAX stiliaus komunikavimas – užklausos ir atsakymo komunikavimo metodas, tuomet kam apsisunkinti projektą su MQTT brokerių įdiegimu, kai tokiam komunikavimui užteks keleto eilučių kodo ir HTTP protokolo.

MQTT protokolas pasižymi kur kas mažesniu antraštės dydžiu, o pats mažiausias galimas duomenų paketo dydis siekia vos 2 baitus. Tačiau HTTP protokolas leidžia naudoti ilgas antraštes ir siųsti itin didelius pranešimus. HTTP yra senas, patikimas protokolas, su beveik neribotomis galimybėmis plėstis. Tačiau jeigu nereikalingas itin sudėtingas sprendimas, MQTT gali tikri labiau. Toliau pateikiamas išsamus HTTP ir MQTT protokolų palyginimas.

Kada verta rinktis MQTT protokolą?

- kai siunčiamus duomenis privaloma gauti realiuoju laiku, ir bet koks vėlavimas yra nepageidautinas;
- kai komunikacijoje dalyvauja didelis kiekis duomenis renkančių prietaisų, kurių informaciją pageidauja gauti didelis kiekis klientų;
- Kai pageidaujama turėti galimybę dvipusiam komunikavimui, t.y. iš prietaiso ne tik gauti pranešimus, bet ir siųsti duomenis į jį.
- kai įmanoma ir naudinga palaikyti ilgai gyvuojantį TCP susijungimą su brokeriu, nes duomenų pokyčiai turi būti perduodami realiu laiku;
- kai naudojamas prietaisas turi itin ribotus energijos išteklius, pavyzdžiui, energiją gauna naudodamas baterijas, tad labai svarbu kiek įmanoma labiau taupyti elektros energiją, kad baterijos nereikėtų dažnai keisti;
- jeigu persiunčiamų duomenų kiekis nėra labai didelis, o duomenys persiunčiami artimoje aplinkoje – t. y. duomenimis dalintis nereikia su skirtinguose pasaulio kraštuose esančiais klientais. Kitokiu atveju verta naudoti debesijos MQTT brokerius, bet juos sunkiau modifikuoti ir įdiegti TLS saugą;
- kai duomenis siunčiantys ar gaunantys klientai nėra labai patikimi ir gali bet kada išsijungti – MQTT protokolas suteikia galimybę kaupti brokeryje žinutes, kurios bus pristatytos vos sugrįžus klientui, taip pat ir „paskutinės valios“ nurodymus, kad iš ryšio zonos dingus duomenis siunčiančiam klientui, brokeris turėtų tolimesnius nurodymus kaip elgtis.

Kada verta rinktis HTTP protokolą?

- kai persiunčiamų duomenų kiekis yra labai didelis, o duomenis į tą patį serverį siunčia prietaisai iš viso pasaulio;
- kadangi HTTP protokolas paprastai veikia tik tarp dviejų bendraujančių subjektų (kliento ir serverio), jis labiau tinkamas reguliariam būsenos raportavimui, o ne įvykių realiam laike fiksavimui;
- HTTP protokolas yra tapęs pagal nutylėjimą numatytuoju protokolu tinklo komunikacijoms. Dėl šios priežasties kuriant bet kokį projektą galima būti įsitikinus, kad tiek kliento, tiek serverio pusė tikrai palaikys šį protokolą, o iškilus problemoms bus galima kreiptis į didelę interneto bendruomenę, kurioje tikrai bus suteikta pagalba;
- kadangi HTTP protokolas daug kur palaikomas, tai užtikrina paprastesnį suderinamumą su ugniasienėmis, o tai tampa itin svarbu, jeigu projektas įgyvendinamas aplinkoje, kurios programos kūrėjas nebekontroliuos;

- verta pagalvoti, ar tikrai reikia jaudintis dėl tinklo pralaidumo ir didesnio papildomo duomenų kiekio, naudojant HTTP protokolą – šiuolaikinių technologijų pagalba komunikacijų pralaidumas vis gerėja, o šiais laikais ne tik interneto, bet ir mobiliojo ryšio tiekėjai siūlo neribotą persiunčiamų duomenų kiekį už fiksuotą sumą.

1.7. Analizės išvados

1. Egzistuoja didelė galimybių rinka robotų kibernetinio saugumo tyrinėjimo srityje. Jeigu žiūrėsime iš saugumo pusės, dauguma robotų yra net nepasiruošę būti pritaikomi kasdieniame gyvenime. Programinė įranga nepasiruošusi apsiginti nuo atakų, nes dauguma komunikacijų įprastai yra neužšifruotos [3].

2. Saugių robotų kūrimas yra sudėtinga užduotis dėl keleto priežasčių: aplinkoje, kurioje bus ne vienas robotas, susidurs su padidėjusiais saugumo pavojais, kadangi netgi jeigu robotas saugus izoliuotoje aplinkoje, jis gali tapti lengvai pažeidžiamas dalyvaudamas jungtinėse atakose. Tipinė aplinka yra dinamiška, kurioje daugybė gyvų būtybių, įskaitant ir nepatyrusius vartotojus, vaikus, pagyvenusius žmones ir gyvūnus; sunku laikyti sistemą saugia be jokio standartizuoto atramos taško [8].

3. Autonominiai robotai atstovauja naują prietaisų klasę, kurios saugumo reikalavimai atrodo besiskiriantys nuo įprastinių kompiuterių ir kompiuterių tinklų sistemų [4].

4. Prieš bandant sukurti tikrai veiksmingą komunikacijos saugos metodą autonominiam robotui, būtina apgalvoti kaip vyks identifikavimas bei autentifikavimas, atsižvelgiant į esamą situaciją, pvz., ar robotas bus naudojamas karinei pramonei, o gal namų ūkiui, parinkti tinkamą sprendimą duomenų šifravimo klausimu bei raktų valdymu. Ar įmanoma ir verta bandyti įgyvendinti SSL, jeigu ne – pasitenkinti paprastesniu saugumo užtikrinimu.

5. Apžvelgus saugos pasiūlymus ir jau įgyvendintus specifinius sprendimus, lengva įžvelgti komunikacijų saugos reikšmę, taip pat ir tampa savaime suprantama, jog kuo geresnį saugos sprendimą bus bandoma pritaikyti, tuo labiau nukentės ištekliai ir bus sunaudojama daugiau energijos, kad yra itin aktualu tokiuose prietaisuose kaip robotai.

2. AUTONOMINIO ROBOTO SAUGAUS KOMUNIKAVIMO METODAS

Pirmojoje darbo dalyje (analizėje) buvo atidžiai ir išsamiai išnagrinėtos autonominių robotų rūšys, apžvelgta, kuo skiriasi kompiuterio ir roboto sauga, išsamiai aprašytos galimos grėsmės nepasirūpinus tinkamu autonominių robotų komunikacijų saugumu, taip pat pasiūlyta specifinių, jau įgyvendintų saugos sprendimų, taip pat apibrėžta tų sprendimų nauda tolimesniam darbui.

Pagrindinė užduotis, kurią reikia išspręsti šiame darbe – sukurti saugų autonominio roboto komunikavimo sprendimą, kurį įgyvendinus nebūtų įmanoma taip lengvai nutekinti jautrius duomenis ir juos klastoti. Norint užtikrinti saugumą, pasirinktame mikrovaldiklyje bus įgyvendintas komunikavimas su SSL kriptografiniu protokolu, užtikrinančiu saugią komunikaciją ir duomenų perdavimą internetu.

2.1. Siekiami rezultatai

2.1.1. Saugumo užtikrinimas

Saugumą galima padidinti įterpiant tokius kriptografinius metodus kaip blokiniai šifrai, maišos funkcijos, parašo algoritmai, taip pat nekriptografinius metodus, kurie naudotų autorizaciją ir kitus saugumo politikos reikalavimų aspektus. Tačiau visi išvardinti įprasti saugumo užtikrinimo metodai gali būti lengvai pritaikomi dideliems, galingiems tinklams, o dėmesys turėtų būti sutelktas į suvaržytus tinklus, dėl to būtina atsižvelgti į papildomas pasekmes, kurios atsiranda suvaržytose, ribotų išteklių aplinkose. Be to, reikėtų kiek įmanoma sumažinti žinučių kiekį bei dydį tam, kad sumažėtų reikalavimai darbinei atminčiai bei būtų optimizuotas papildomas duomenų srautas, tačiau tuo pačiu išlaikant aukštus saugumo standartus. Žinoma, bandant supaprastinti saugumo protokolą dėl energijos naudojimo sumažinimo, privaloma tikėtis saugumo kokybės suprastėjimo, todėl reikėtų surasti patenkinamą pusiausvyrą abiejose pusėse.

Saugumo užtikrinimui reikalingi trys pagrindiniai elementai – vientisumas, autentifikavimas ir konfidencialumas, o TLS gali įgyvendinti juos visus. Kitaip nei tinklo sluoksnio saugumo protokolai, TLS užtikrina saugą nuo komunikacijos pradžios iki galo. Vadinasi, naudojant kuriamą autonominio roboto saugaus komunikavimo metodą, įsilaužėliui bus nelengva pasiekti per tinklo įrenginį keliaujančią informaciją.

2.1.2. Taupus elektros energijos vartojimas

Robotą dažniausiai sudaro penki pagrindiniai komponentai: energijos šaltinis, motoras, jutikliai, mikrovaldiklis bei integruotas kompiuteris. Nors dabar, vis labiau populiarėjant atsinaujinančiai energetikai, kai kuriems robotams įdiegiamos saulės energiją naudojančios baterijos, bet pats populiariausias energijos šaltinis išlieka paprastos, pakartotinai įkraunamos baterijos. Kadangi robotai su laiku tampa vis sudėtingesni, visos valdymo sistemos, jutikliai, komunikacijos ir skaičiavimų

operacijos naudoja vis daugiau ir daugiau elektros energijos. Mikrovaldiklis periodiškai siunčia komandas į motorus ir jutiklius, ištraukia iš jų duomenis ir komunikuoja su integruotu kompiuteriu. Kadangi mikrovaldiklio užduotys įprastai būna fiksuoto dydžio ir su reguliariais pakartojimais, galima nesudėtingai išmatuoti jo sunaudojamą elektros energiją. Atliekant bandymus ir išmatuojant elektros energijos suvartojimą galima pastebėti, kad mikrovaldiklio suvartojamos elektros energijos kiekis žymiai kinta priklausomai nuo dirbančios programos sudėtingumo [23]. Vadinasi, kuriamas autonominio roboto saugaus komunikavimo metodas pasižymės nedideliu elektros suvartojimu.

2.1.3. Riboti skaičiavimo ištekliai

Paprastai, jeigu robotas yra modernesnis ir turi keletą jutiklių, atlieka nuolatinius, sudėtingus skaičiavimus, jam neužtenka tik mikrovaldiklio – tokiu atveju mikrovaldiklis užsiima tik žemo lygio užduotimis, pavyzdžiui, tiesioginiu motorų valdymu ar duomenų ištraukimu iš jutiklių. Tuo pat metu jis suteikia programavimo sąsają integruotam kompiuteriui, kuris, turėdamas kur kas galingesnius skaičiavimo pajėgumus, užsiima aukšto lygio užduotimis, pavyzdžiui, judesių valdymu ir veiklos koordinacija. Tačiau šiuo atveju kuriamas saugaus komunikavimo metodas skirtas nesudėtingam autonominiam robotui, kuris visiems skaičiavimams ir valdymui naudoja tik mikrovaldiklį be papildomo, galingesnio integruoto kompiuterio, todėl jo skaičiavimo ištekliai yra riboti. Vadinasi, kuriamas autonominio roboto saugaus komunikavimo metodas pasižymės nedideliu skaičiavimo išteklių naudojimu.

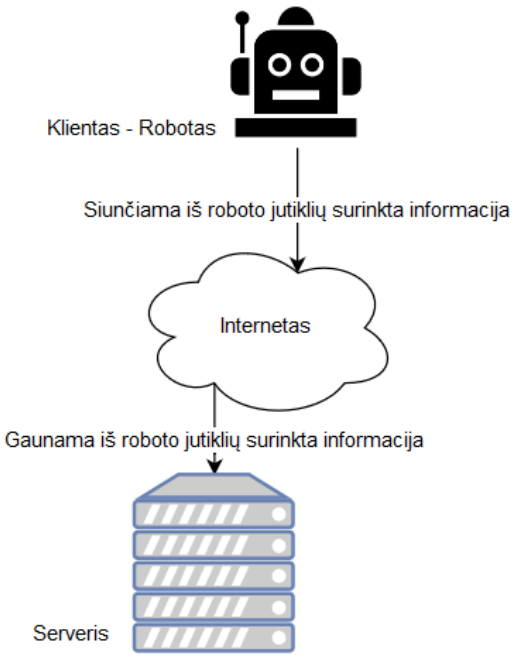
2.1.4. Ribota prieiga prie tinklo resursų

Ypač dažnai su tinklu susijusiomis problemomis susiduriama, jeigu tinkle informacija dalinasi daugiau negu pora robotų arba duomenis surenkantis serveris yra nutolęs. Robotų komunikacijai įgyvendinti naudojamos trumpalaikės, pasikartojančios užklausos, kurios turi būti įvykdytos per tam tikrą, griežtai nustatytą laiko tarpą. Tam, kad duomenų apsikeitimas įvyktų bendradarbiaujant realiu laiku, robotai ir serveris turi palaikyti nuolatinę komunikaciją, o informacijai perduoti skirtas tinklo srautas turi būti pakankamai didelis. Nors šiuo atveju kuriamas saugaus komunikavimo metodas bus išbandomas neapkrautame namų tinkle, kuriame dalyvauja tik vienas robotas ir vienas serveris, tačiau metodas vis tiek pasižymės nedideliu papildomu duomenų srautu ir bus tinkamas ribotos prieigos tinklams.

2.2. Planuojama serverio – kliento architektūra

Šiame autonominio roboto saugaus komunikavimo metode visa roboto, kuris atlieka kliento vaidmenį, surinkta informacija siunčiama į vietinį serverį naudojant WiFi (*žr*. 2.1 pav.). Pavyzdžiui, autonominis namų aplinkos stebėjimo robotas su judesio jutikliais siunčia informaciją į serverį kiekvieną kartą kažkam sujudėjus – aktualu, kai išvykstama atostogų ir namai paliekami be priežiūros.

Paprastai informacija siunčiama TCP ar UDP transporto protokolais, priklausomai nuo to, koks taikomųjų programų lygmens protokolas naudojamas – HTTP, MQTT ar, pavyzdžiui, CoAP. Šiame darbe bus išbandomi du protokolai – HTTP ir MQTT. O šio kuriamo metodo tikslas tarp taikomųjų programų lygmens ir transporto lygmens įterpti TLS apsaugą, kad taptų neįmanomas keliaujančios informacijos nutekėjimas.



2.1 pav. Planuojamos serverio – kliento architektūros funkcinė schema

2.3. Protokolų saugos lygmenų modifikacijos

Prietaiso identifikavimo naudojimas. Tiek HTTP, tiek MQTT galima naudoti kliento autentifikaciją su prisijungimo vardu bei slaptažodžiu – taip galima sustabdyti pašalinių asmenų žinučių siuntimą į serverį. Pavyzdžiui, MQTT protokole tai atlikti labai lengva, nes tereikia *mosquitto.conf* faile aktyvuoti slaptažodžių panaudojimą ir sukurti *passwords.txt* failą, kuriame įrašomi visų vartotojų, kuriems leista prisijungti, duomenys. Tačiau vartotojo vardo ir slaptažodžio naudojimas praranda prasmę, jeigu jie prisijungiant siunčiami neužšifruoti ir matomi bet kuriam besiklausančiam – kyla susidūrimo viduryje (angl. *Man-in-the-Middle*) atakos pavojus. Todėl vis tiek reikalingas transporto lygio šifravimas TLS.

SSL / TLS palaikymas. Tiek HTTP, tiek MQTT palaiko SSL / TLS saugą. Kadangi prieš pasiekiant tikslą, TCP paketai keliauja per daugybę ryšio infrastruktūros komponentų, kiekvienas infrastruktūros dalyvis gali paprasčiausiai perskaityti siunčiamą informaciją arba net ją pakeisti. Todėl užtikrinti saugiam komunikacijos kanalui būtina naudoti TLS, kur jokia trečioji šalis negalės prisiliesti prie siunčiamų duomenų su blogais ketinimais – aišku, tik tokiu atveju, jeigu bus naudojami saugūs šifro komplektai (angl. *cipher suites*) bei saugi TLS versija. Jeigu dėl ribotų išteklių viešojo rakto

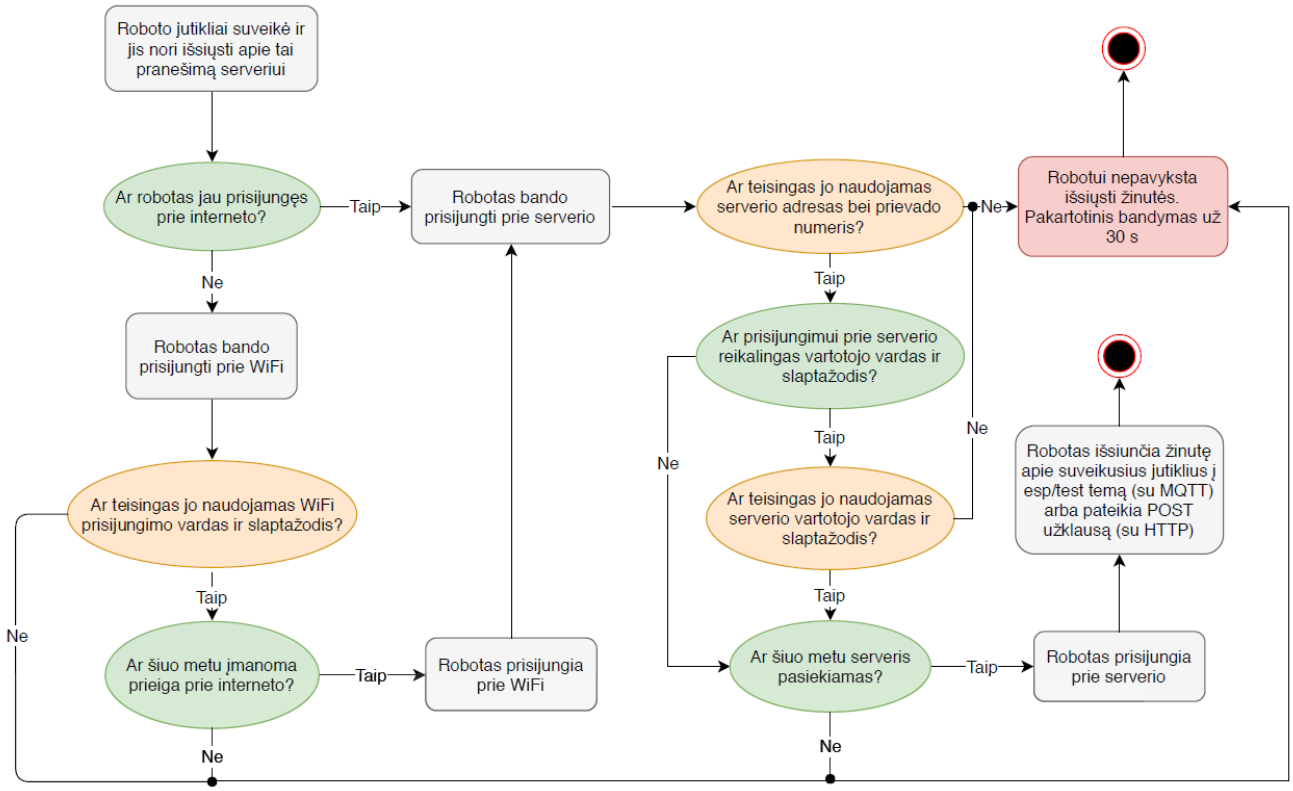
operacijos neįmanomos, verta išbandyti TLS – PSK šifro komplektus, tačiau prieš tai reikėtų įsitikinti, ar pasirinkta TLS biblioteka juos palaiko, nes TLS – PSK nėra itin populiarus.

Pranešimų pristatymų lygiai. Priešingai nei HTTP, MQTT yra lankstesnis ir gali pasiūlyti tris skirtingus lygius pranešimų pristatymo kokybei ir saugai – kas ypač aktualu suvaržytuose tinkluose bei įrenginiuose. QoS0 lygį rekomenduojama naudoti, kai ryšys tarp siuntėjo ir gavėjo yra visiškai arba beveik stabilus, kai nėra gyvybiškai svarbu gauti kiekvieną iš roboto išsiųstą žinutę, taip pat kai brokeriui nebūtina kaupti nepristatytų žinučių, nes klientas – gavėjas visą laiką yra aktyvus. QoS1 lygį rekomenduojama naudoti, kai privaloma gauti kiekvieną žinutę, o tinklo apkrova ir energijos suvaržymai yra nesvarbūs – su žinučių pasikartojimais bus sėkmingai susitvarkoma. Šis lygis populiariausias, kadangi suteikia garantiją, jog žinutė bus pristatyta bent vieną kartą, o pridėtinės informacijos yra ne tiek ir daug. QoS2 lygį rekomenduojama naudoti, kai programos ar įtaiso veikimas priklauso nuo lygiai vieną kartą gaunamos žinutės – pavyzdžiui, jeigu dėl dublikatų sutriktų prietaiso veikimas ar nukentėtų žinučių prenumeratoriai. Tačiau būtina turėti omenyje, kad padidės pridėtinės informacijos kiekis bei žinutės išsiuntimas užtruks ilgiau.

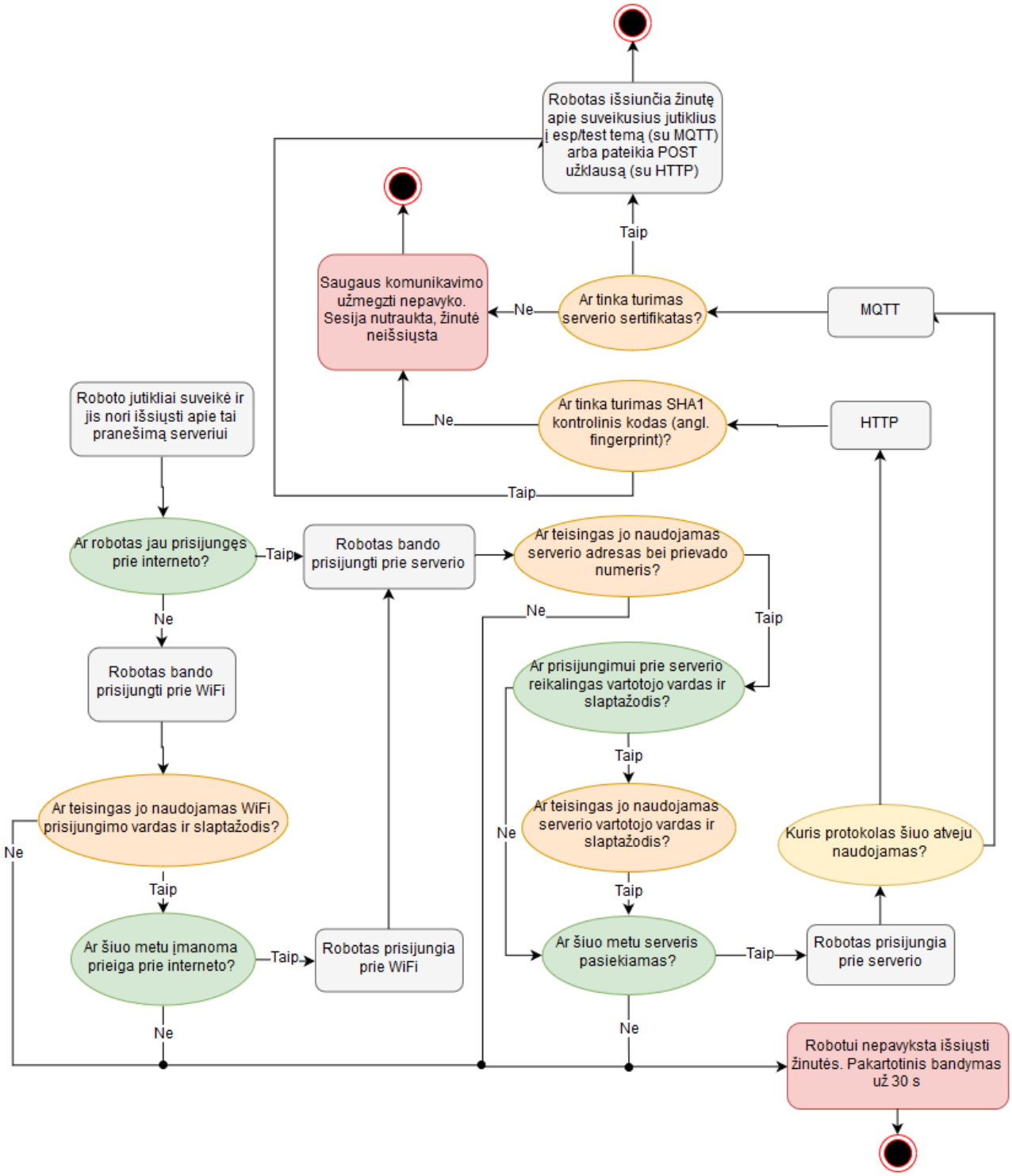
2.4. Komunikavimo duomenų srautai

Įgyvendinamo roboto komunikavimo su serveriu scenarijaus UML diagramos pateikiamos 2.2 pav., 2.4 pav. ir 2.3 pav. Jeigu komunikacija yra be TLS apsaugos (*žr.* 2.2 pav. arba 2.4 pav.), o robotas nori išsiųsti pranešimą apie suveikusius jutiklius serveriui, pirmiausiai jis bando prisijungti prie interneto ryšio. Tuomet tikrinama, ar teisingas roboto naudojamas WiFi prisijungimo vardas ir slaptažodis. Jeigu tai duomenys teisingi ir tuo metu įmanoma prieiga prie interneto, bandoma prisijungti prie serverio. Tokiu atveju tikrinama, ar teisingas roboto naudojamas serverio adresas bei prievado numeris (MQTT protokolo atveju – ar teisingi brokerio duomenys). Jeigu prisijungimui prie serverio ar brokerio reikalingas vartotojo vardas ir slaptažodis, jie pateikiami. Jeigu duomenys teisingi ir serveris / brokeris šiuo metu yra pasiekiamas, robotas sėkmingai prisijungia prie serverio, ir atitinkamai nuo to, koks protokolas naudojamas, išsiunčiama žinutė į tam tikrą temą (naudojant MQTT protokolą) arba pateikiama POST užklausa (naudojant HTTP protokolą). Jeigu kuriame nors žingsnyje nepavyksta prisijungti prie serverio, interneto, ar nepavyksta autentifikacija, žinutė neišsiunčiama, o pakartotinis bandymas vyksta už 30 sekundžių. Jeigu komunikacija vyksta su TLS apsauga (žr. 2.4 pav. arba 2.3 pav.), o robotas nori išsiųsti pranešimą apie suveikusius jutiklius serveriui, pirmiausiai jis bando prisijungti prie interneto ryšio. Tuomet tikrinama, ar teisingas roboto naudojamas WiFi prisijungimo vardas ir slaptažodis. Jeigu tai duomenys teisingi ir tuo metu įmanoma prieiga prie interneto, bandoma prisijungti prie serverio. Tokiu atveju tikrinama, ar teisingas roboto naudojamas serverio adresas bei prievado numeris (MQTT protokolo atveju – ar teisingi brokerio duomenys). Jeigu prisijungimui prie serverio ar brokerio reikalingas vartotojo vardas ir slaptažodis, jie pateikiami. Jeigu

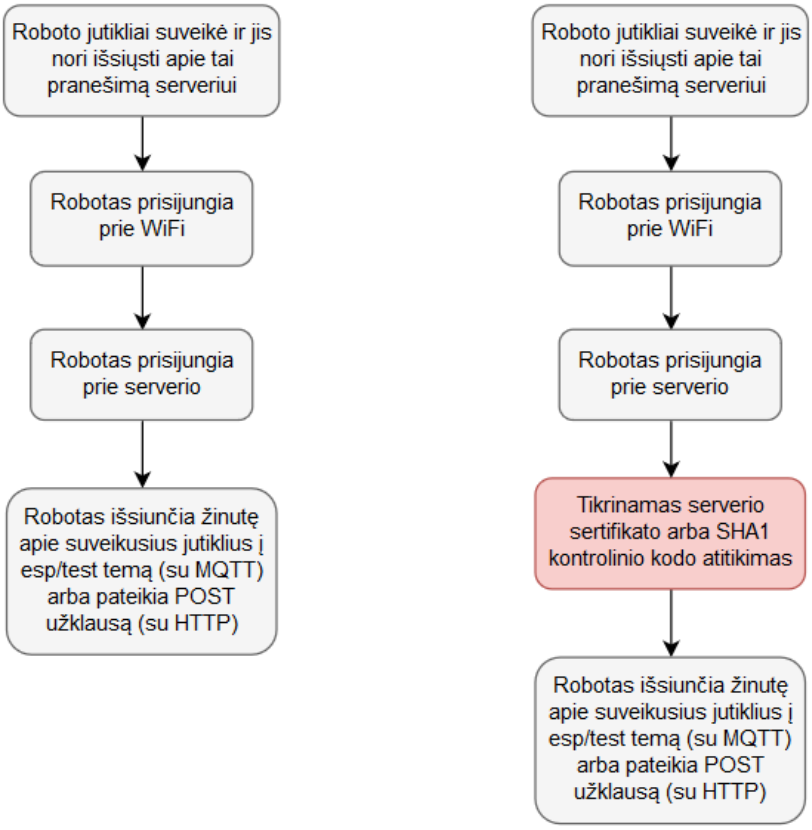
duomenys teisingi ir serveris / brokeris šiuo metu yra pasiekiamas, robotas sėkmingai prisijungia prie serverio, tačiau prieš siunčiant žinutę įsitikinama, ar komunikacija yra tikrai saugi – atitinkamai nuo to, koks protokolas naudojamas, tikrinamas serverio protokolo atitikimas (naudojant MQTT protokolą) arba SHA1 kontrolinis kodas (naudojant HTTP protokolą), ir tik tuomet išsiunčiama žinutė į tam tikrą temą arba pateikiama POST užklausa. Jeigu kuriame nors žingsnyje nepavyksta prisijungti prie serverio, interneto, ar nepavyksta autentifikacija, žinutė neišsiunčiama, o pakartotinis bandymas vyksta už 30 sekundžių. Jeigu netinka turimas serverio sertifikatas, arba netinka turimas SHA1 kontrolinis kodas, vadinasi, kad saugaus komunikavimo užmegzti nepavyko, sesija nutraukiama, o žinutė neišsiunčiama.



2.2 pav. Išsami sekos diagrama – HTTP ir MQTT nesaugus komunikavimas



2.3 pav. Išsami sekos diagrama – HTTPS ir MQTTS saugus komunikavimas



2.4 pav. Supaprastintos sekos diagramos – HTTP ir MQTT nesaugus ir saugus komunikavimas

2.5. Metodo išvados

1. Kuriamas metodas turi suteikti roboto komunikacijoms saugą, pasižymėti taupiu elektros energijos vartojimu, tikti ribotus skaičiavimo išteklius bei ribotą prieigą prie tinklo turintiems robotams.

2. Apžvelgtos galimos saugos modifikacijos, kuriomis pasižymi tyrimui pasirinkti komunikacijų protokolai – tiek MQTT, tiek HTTP palaiko TLS saugą bei prietaiso identifikavimo naudojimą, tačiau tik MQTT suteikia galimybę rinktis pageidaujamą pranešimų pristatymo lygį.

3. Taip pat pavaizduoti ir aprašyti planuojami komunikacijų duomenų srautai bei eiga, kuriais remiantis numatoma, kad MQTT ir HTTP protokolų naudojimo eiga turėtų praktiškai nesiskirti nenaudojant TLS saugos, tačiau jau ją įdiegus atsiranda keli papildomi ir skirtingi eigos žingsniai.

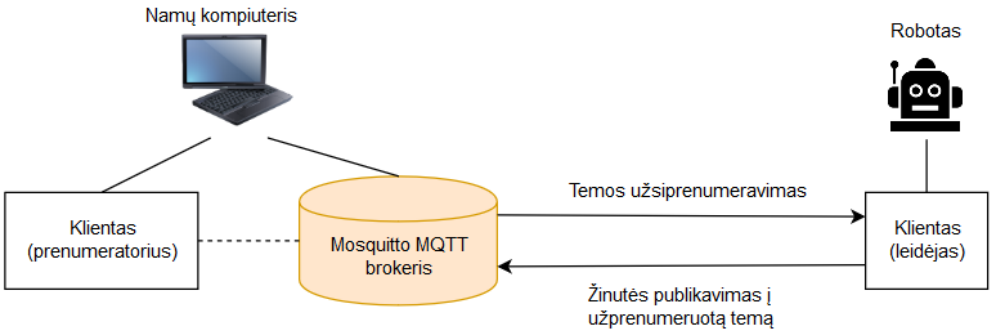
3. AUTONOMINIO ROBOTO SAUGAUS KOMUNIKAVIMO METODO PROTOTIPAS

Kuriant autonominio roboto saugaus komunikavimo metodo prototipą, tyrimams pasirinkti MQTT bei HTTP protokolai, kurie realizuojami naudojant „Arduino IDE“ platformą bei negalingą ESP8266 mikrovaldiklį. Protokolai bus išbandomi tiek neapsaugoti, tiek su TLS apsauga, o keičiant konfigūraciją, gaunami rezultatai bus palyginami ketvirtojoje, eksperimentinėje dalyje. Testuojami protokolai ir konfigūracijos aprašytos tolesnėje apžvalgoje.

3.1. MQTT leidėjo – prenumeratoriaus architektūra

Prieš išbandant komunikaciją su MQTT protokolu, apžvelgiama jo leidėjo – prenumeratoriaus architektūra (*žr.* 3.1 pav.). Kadangi MQTT naudoja užsiprenumeravimo – publikavimo metodą, tad tuo pat metu serveris taip pat gali atlikti ir kliento vaidmenį. Šiuo tiriamuoju atveju MQTT „Mosquitto“ brokeris, skirtas žinučių gavimui bei paskirstymui prenumeratoriams, bus įdiegtas namų kompiuteryje, tad jis atliks brokerio / serverio vaidmenį architektūroje – iš jo bus galima stebėti prisijungiančių klientų skaičių, laiką, siunčiamų žinučių temas. Brokeris atsiunčiamų nešifruotų žinučių klausysis naudodamas 1883 prievadą, o apsaugotų TLS – 8883 prievadą. Kadangi tas pats kompiuterį bus naudojamas ir siunčiamų žinučių iš ESP8266 valdiklio klausymuisi (užsiprenumeravus temą „esp/test“), tad prie brokerio / serverio prisidės ir kliento vaidmuo. Visa tai atliekama turint omenyje tokia situaciją, kai, pavyzdžiui, autonominis namų aplinkos robotas su judesio jutikliais siunčia informaciją į namų kompiuterį kiekvieną kartą kažkam sujudėjus – aktualu, kai išvykstama atostogų ir namai paliekami be priežiūros.

Šiuo atveju robotas, naudojantis ESP8266 mikrovaldiklį, atliks tik kliento – žinučių siuntėjo vaidmenį. Pavyzdžiui, jo užduotis galėtų būti išsiųsti žinutę apie namuose vaikščiojančius asmenis, kiekvieną kartą suveikus judesio davikliui. Šiuo tiriamuoju atveju ir bus išsiunčiama tokia žinutė į temą „esp/test“, tačiau judesio daviklis prijungtas nebus, nes tai tiriant perduodamų duomenų saugumą nėra aktualu.

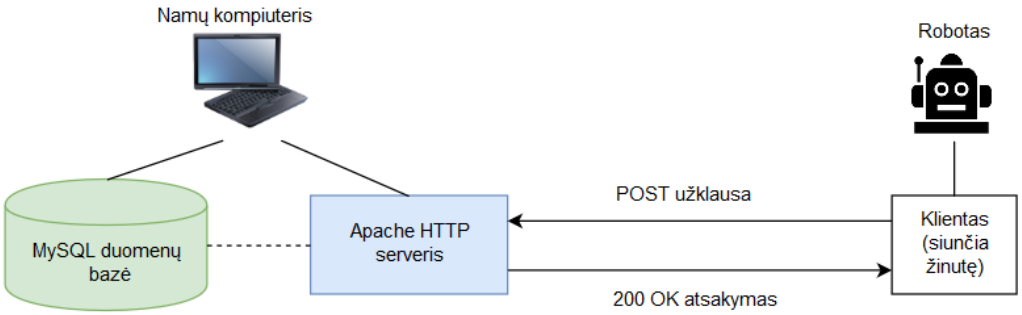


3.1 pav. MQTT serverio – kliento architektūros funkcinė schema

3.2. HTTP serverio – kliento architektūra

Toliau apžvelgiama HTTP protokolo serverio – kliento architektūra (*žr.* 3.2 pav.). HTTP protokolas veikia naudodamas užklausos – atsakymo metodą. Šio tyrimo metu kompiuteryje bus įdiegtas „Apache“ HTTP serveris, taip pat sukurta „MySQL“ duomenų bazė, kurioje bus talpinamos iš ESP8266 gautosios žinutės – jas bus galima peržiūrėti ar ištrinti. Serveris atsiunčiamų nešifruotų žinučių klausysis naudodamas 8080 prievadą, o apsaugotų TLS – 443 prievadą. Taigi, tokiu atveju kompiuteris architektūroje atliks serverio vaidmenį.

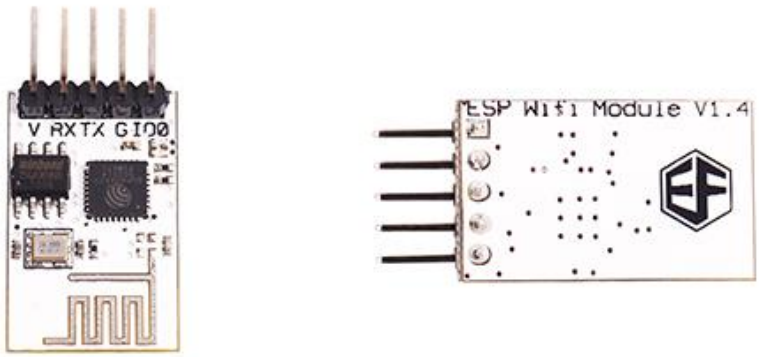
Naudojant HTTP protokolą, robotas, naudojantis ESP8266 mikrovaldiklį, architektūroje atliks kliento vaidmenį. Jis, norėdamas išsiųsti žinutę, pateiks POST užklausą serveriui, iš kurio turėtų sulaukti atsakymo 200 OK, jeigu užklausa bus gauta sėkmingai.



3.2 pav. HTTP serverio – kliento architektūros funkcinė schema

3.3. ESP8266 mikrovaldiklis

Darbo tyrimams pasirinktas 32 bitų ESP8266 mikrovaldiklis (*žr.* 3.3 pav.), kuris šiuo metu itin sparčiai populiarėja tarp įvairių „daiktų interneto“ projektų – dėl santykinai mažos kainos ir didelių panaudojimo galimybių. Oficialiai jis vadinamas tiesiog ESP8266 WiFi moduliu – „sistema ant lusto“ (angl. *system-on-a-chip, SOC*), kuriame yra integruotas TCP / IP protokolo dėklas (angl. *stack*), kurio pagalba bet koks kitas mikrovaldiklis gali prisijungti prie WiFi.



3.3 pav. ESP8266 ESP-01 mikrovaldiklis [32]

ESP8266 mikrovaldiklį galima panaudoti dviem būdais – prijungti jį prie kito, galingesnio mikrovaldiklio, pavyzdžiui, „Arduino Uno“, arba jeigu užduotis nereikalauja labai didelių resursų, tiesiog programuoti jį patį. Šiuo atveju pasirinkta ESP8266 mikrovaldiklio modifikacija yra viena paprasčiausių – ESP-01, kurioje galima rasti 512kB atmintuką (angl. *flash memory*) bei 82kB SRAM (statinė laisvosios prieigos atmintis (angl. *Static Random-Access Memory*). Kadangi į kiekvieną ESP8266 modulį iš anksto įrašoma AT tipo programinė aparatinė įranga (angl. *firmware*), tokiu būdu tampa itin paprasta prijungti ESP8266 prie bet kokio kito, galingesnio mikrovaldiklio, ir gauti visas WiFi teikiamas galimybes, kokias suteiktų daug brangesnis ir sudėtingas, pavyzdžiui, „Arduino WiFi“ skydas.

Svarbiausia priežastis, kodėl taip sėkmingai išpopuliarėjo šis mikrovaldiklis, yra tai, kad ESP8266 turi pakankamai galingą mikroprocesorių ir užtektinai atminties nedideliems namų projektams – jį paprasta integruoti su įvairiais jutikliais. Išmanusis elektros kištukas, namų automatika, belaidžių įtaisų valdymas, kūdikio pasiklausymo radijas, namų judesio jutiklis, integruota nešiojamoji elektronika, įėjimo kortelė dėl saugumo – tai vieni iš populiariausių „daiktų interneto“ pavyzdžių, kuriuose naudojamas šis mikrovaldiklis.

ESP8266 mikrovaldiklio detalioji schema pateikiama priede Nr. 7.1. Ši ESP8266 mikrovaldiklio modifikacija turi 5 įvestis / išvestis – VCC, RX, TX, GND, GPIO0.

- VCC įvestis skirta mikrovaldiklio maitinimui. ESP8266 privaloma tiekti 3.3 V (ne aukštesnę nei 3.6 V) įtampą, dėl ko dažnai tenka įsigyti loginį įtampos keitiklį (jeigu ESP8266 jungiamas prie kito mikrovaldiklio kaip WiFi skydą), nes dauguma galingesnių mikrovaldiklių gali tiekti tik 5 V įtampą.
- RX (angl. *receive*) įvestis skirta gaunamiems duomenims – pavyzdžiui, jeigu norima įkelti programinį kodą į šį mikrovaldiklį, tam bus naudojama ši įvestis.
- TX (angl. *transmit*) išvestis skirta iš mikrovaldiklio siunčiamiems duomenims.
- GND įvestis / išvestis skirta mikrovaldiklio įžeminimui.
- GPIO0 yra įvestis / išvestis bendram naudojimui – pavyzdžiui, prijungti jutiklius. Taip pat šią įvestį / išvestį privaloma įžeminti keliant naują programinį kodą į mikrovaldiklį – jį įkėlus, įžeminimą galima atjungti.

3.3.1. Maitinimas ir duomenų perkėlimas

Kaip buvo paminėta ankstesniame 3.3 skyrelyje, ESP8266 mikrovaldiklio maitinimui būtina 3.3 V įtampa. Taigi, šį mikrovaldiklį galima maitinti keliais būdais – naudoti galvaninius elementus arba nešiojamus, iš anksto įkraunamus energijos bankus, taip pat galima maitinti prijungus prie kito, galingesnio mikrovaldiklio arba tiesiai į kompiuterio USB (universali nuoseklioji kompiuterio magistralė (angl. *Universal Serial Bus*)). Kadangi šiame tyrime atliekama daug bandymų, o jie dar ir pakartojami dešimtis ar šimtus kartų, tiksliausiems rezultatams gauti pasirinktas maitinimas iš

kompiuterio USB magistralės. Tam prireikė USB TTL (tranzistorius – tranzistoriui logikos, angl. *transistor – transistor logic*) UART (universalaus asinchroninio imtuvo – siųstuvo, angl. *universal asynchronous receiver – transmitter*) – prietaiso tilto (*žr.* 3.4 pav.), kuris verčia lygiagrečius duomenų bitus į nuoseklius duomenų bitus. Rekomenduojama duomenų parsiuntimo sparta, naudojant ESP8266 kartu su USB TTL UART tiltu – 57600 arba 115200 sparta bodais (angl. *baud rate*).

3.4 pav. CP2102 USB į TTL UART tiltas [33]

Naudojant šį tiltą, nebereikalingas loginis įtampos keitiklis, kadangi CP2102 USB į TTL UART tiltas gali tiekti dviejų rūšių įtampą – tiek 5 V, tiek 3.3 V. Iš viso šis tiltas turi 5 įvestis / išvestis – 5 V, GND, RXD, TXD ir 3.3 V, kurių funkcijos atitinka ESP8266 aprašytas funkcijas. Taigi, šis tiltas šio darbo bandymuose atlieka dvi funkcijas – tiekia maitinimą ESP8266 mikrovaldikliui, taip pat jį naudojant galima įkelti programinį kodą į ESP8266 mikrovaldiklį iš kompiuterio.

3.3.2. Suvartojamos elektros energijos matavimas

Kadangi visame darbe tiriami energetiškai suvaržyti autonominiai robotai, vienas iš labai svarbių aspektų tampa elektros energijos suvartojimas. Norint sukurti patį tinkamiausią saugų komunikacijos metodą, būtina atsižvelgti ir į protokolo energetinį efektyvumą – galbūt verta paaukoti kelias, ne pačias svarbiausias saugos funkcijas, dėl kur kas sumažėjusio elektros suvartojimo. Ypač jeigu autonominis robotas dirbtų namų aplinkoje, o ne valstybinės reikšmės įstaigoje.

Suvartojamą elektros energiją, „siurbiamą“ iš USB magistralės, išmatuoti be jokių specialių įtaisų būtų sudėtinga, tad papildomai naudojamas UNI-T įtampos ir srovės matuoklis UT658 (*žr.* 3.5 pav.). Jį galima tiesiogiai prijungti prie kompiuterio USB magistralės, o per jį – CP2102 USB į TTL UART tiltą. Šis matuoklis gali užfiksuoti įtampą 3.0 V – 9.0 V, o srovę 0.0 A – 3.0 A ribose.

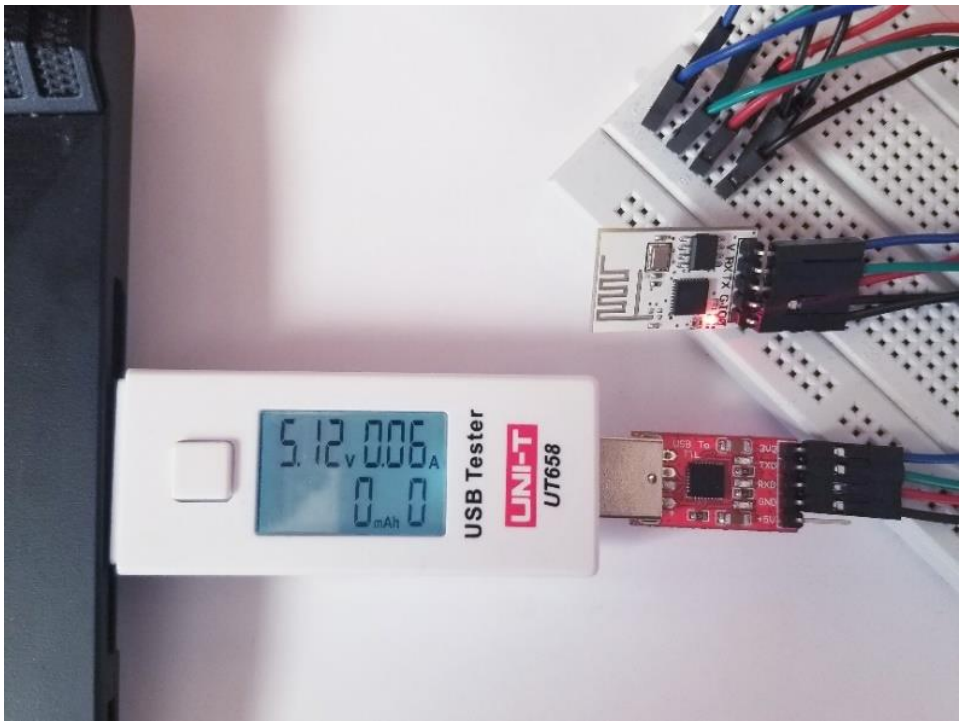
3.5 pav. UNI-T UT658 įtampos ir srovės matuoklis [34]

Norint išmatuoti sunaudojamą elektros energiją, prireiks įtampos, vidutinės srovės ir užduočiai atlikti sugaišto laiko. Pavyzdžiui, jeigu paleidus programinį kodą, įtampa yra 5 V, srovė 1 A, o užduotis atlikta per 30 s, naudojama tokia formulė:

$$5\,V \cdot 1\,A \cdot 30\,s = 150\,Ws$$

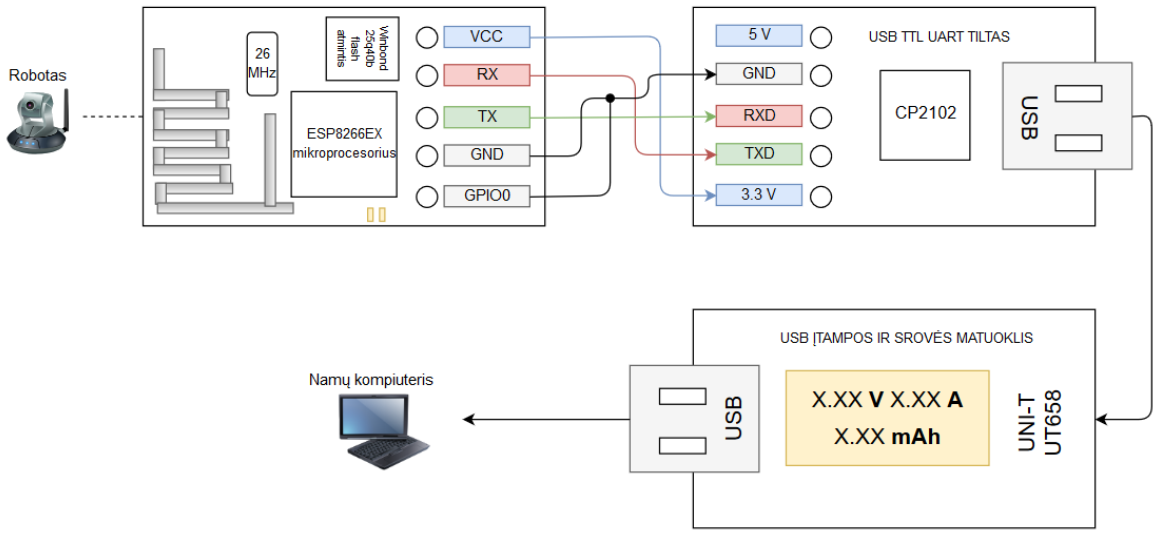
Atlikus skaičiavimus, gaunamos vatsekundės (arba SI sistemos vienetai J (džauliai)). Gautų rezultatų neverta konvertuoti į gerai pažįstamas kWh, nes šiuo atveju elektros suvartojimas nebus toks didelis.

Sujungus visus prietaisus, prijungus prie kompiuterio USB magistralės ir paruošus darbui ESP8266 mikrovaldiklį, CP2102 USB į TTL UART tiltą ir UNI-T UT658 įtampos ir srovės matuoklį, gaunamas toks vaizdas (*žr.* 3.6 pav.):



3.6 pav. ESP8266, tiltas ir matuoklis prijungtas prie mano kompiuterio USB

Visas sujungimas – supaprastinta funkcinė elektrinė schema pateikta 3.7 pav.:



3.7 pav. Kompiuterio USB, ESP8266, tilto ir matuoklio sujungimo schema

3.4. Metodo prototipo išvados

1. Kuriant metodo prototipą pastebėta ir nuspręsta, kad MQTT komunikavimą vietoj serverio ir kliento labiau tinkama apibrėžti kaip leidėjo – prenumeratoriaus architektūrą, kur robotas bus žinučių leidėjas, o kompiuteris bus žinučių prenumeratorius. Taip pat į tą patį kompiuterį įdiegiamas MQTT žinučių brokeris, skirtas paskirstyti žinučių srautą tarp leidėjų ir prenumeratorių, dėl šios priežasties šioje MQTT architektūroje kompiuteris atliks du vaidmenis: brokerio ir papildomo kliento – prenumeratoriaus.

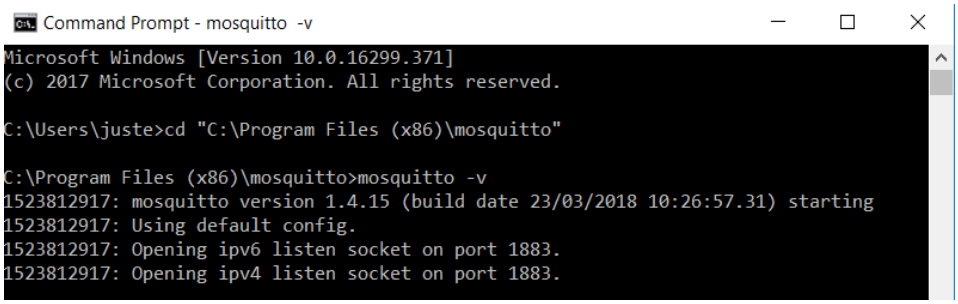
2. HTTP komunikavimas vyksta naudojant įprastą serverio – kliento architektūrą, kur robotas atlieka kliento vaidmenį ir siunčia POST tipo užklausas į „Apache“ serverį, kuris šiuo atveju įdiegtas kompiuteryje. Gaunamų užklausų rezultatai talpinami į „MySQL“ duomenų bazę.

3. Darbo tyrimams atlikti pasirinktas ESP8266 mikrovaldiklis, itin sparčiai populiarėjantis „daiktų interneto“ projektuose. Mikrovaldiklio maitinimui bei kodo perkėlimui naudojamas CP2102 USB į TTL UART tiltas, o išnaudojamos elektros matavimui pasirinktas UNI-T UT658 įtampos ir srovės matuoklis.

4. AUTONOMINIO ROBOTO SAUGAUS KOMUNIKAVIMO METODO EKSPERMENTINIS TYRIMAS

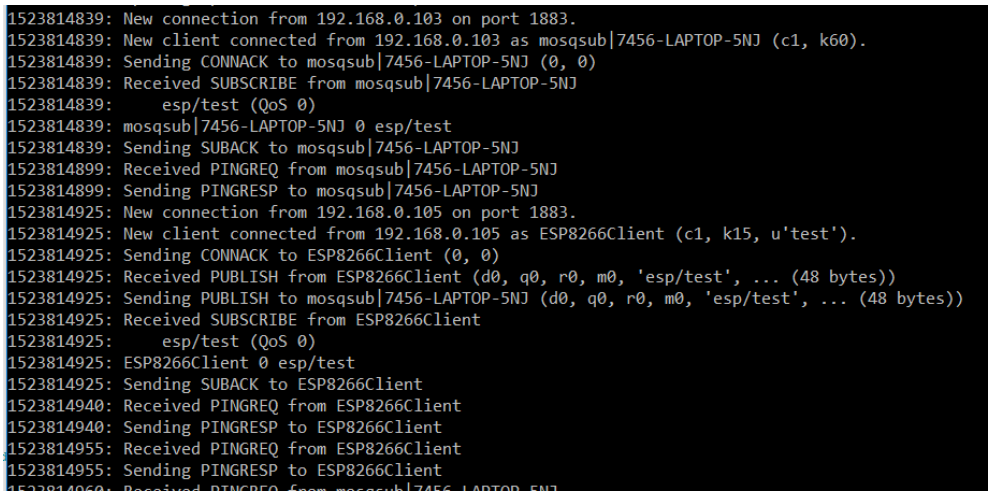
4.1. MQTT protokolo įgyvendinimas mikrovaldiklyje

Įgyvendinant komunikavimą tarp mikrovaldiklio (kliento) ir kompiuterio (serverio, dar kitaip MQTT algoritme vadinamo „brokeriu“), pradedama nuo atviro kodo „Eclipse Mosquitto MQTT v3.1“ [24] brokerio sistemos. Sėkmingai įdiegus ir paruošus kompiuterį veikimui serverio rėžimu, prie Windows paslaugų (angl. *services*) atsiranda veikiantis „Mosquitto“ brokeris. „Mosquitto“ brokerį galima paleisti tiek rankiniu, tiek automatiniu būdu kiekvieną kartą iš naujo perkrovus Windows sistemą. Pagal nutylėjimą, „Mosquitto“ naudoja 1883 prievadą nešifruotam MQTT komunikavimui (*žr*. 4.1 pav.), o 8883 prievadą SSL / TLS apsaugotam komunikavimui.



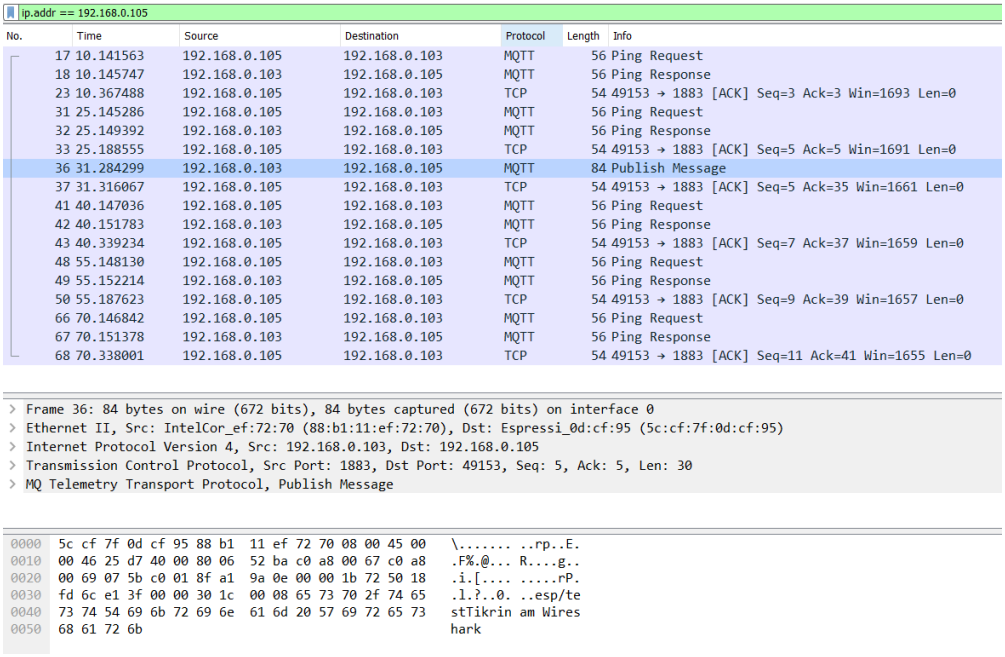
4.1 pav. „Mosquitto“ brokeris klausosi naudodamas 1883 prievadą

Prieš pradedant įgyvendinti MQTT su TLS, išbandoma ar pavyksta paprastas, nešifruotas komunikavimas tarp kompiuterio ir ESP8266 mikrovaldiklio. Tam naudojami „Mosquitto“ brokerio įrankiai kaip „*mosquitto_sub*“, kuris skirtas užsiprenumeruoti pranešimus, ir „*mosquitto_pub*“, kuris skirtas publikuoti pranešimus. Naudojant „Arduino IDE“ platformą ir „*PubSubClient*“ biblioteką [28], iš ESP8266 mikrovaldiklio išsiunčiamas pranešimas, o su Windows Powershell programa užsiprenumeruojama „*esp/test*“ tema. Bendravimas buvo užmegztas sėkmingai (*žr*. 4.2 pav.).



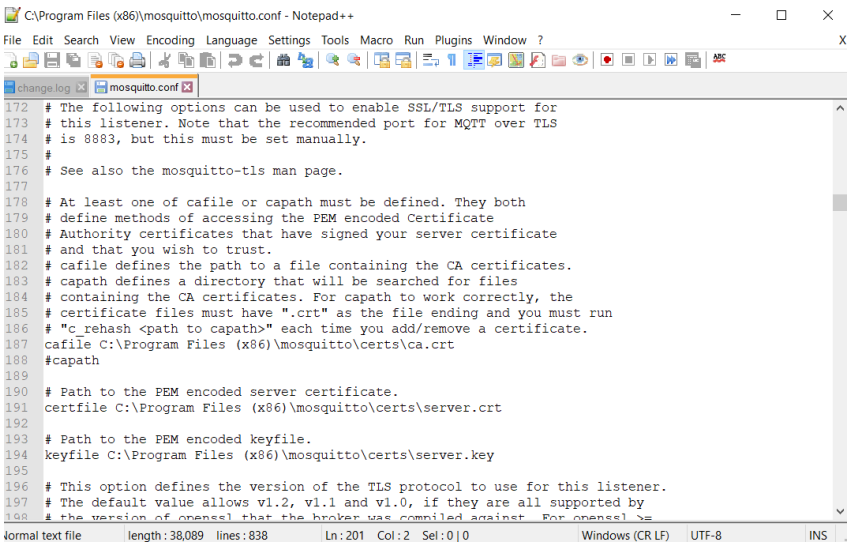
4.2 pav. Užmegztas bendravimas tarp kompiuterio ir ESP8266, naudojant MQTT 1883 prievadą

Taip pat naudojant „WireShark“ programinę įrangą galima įsitikinti, kad žinutės keliauja tikrai neužšifruotos ir žinutė „*Tikrinam Wireshark*“ aiškiai, bet kuriam besiklausančiam, matoma (*žr.* 4.3 pav.):



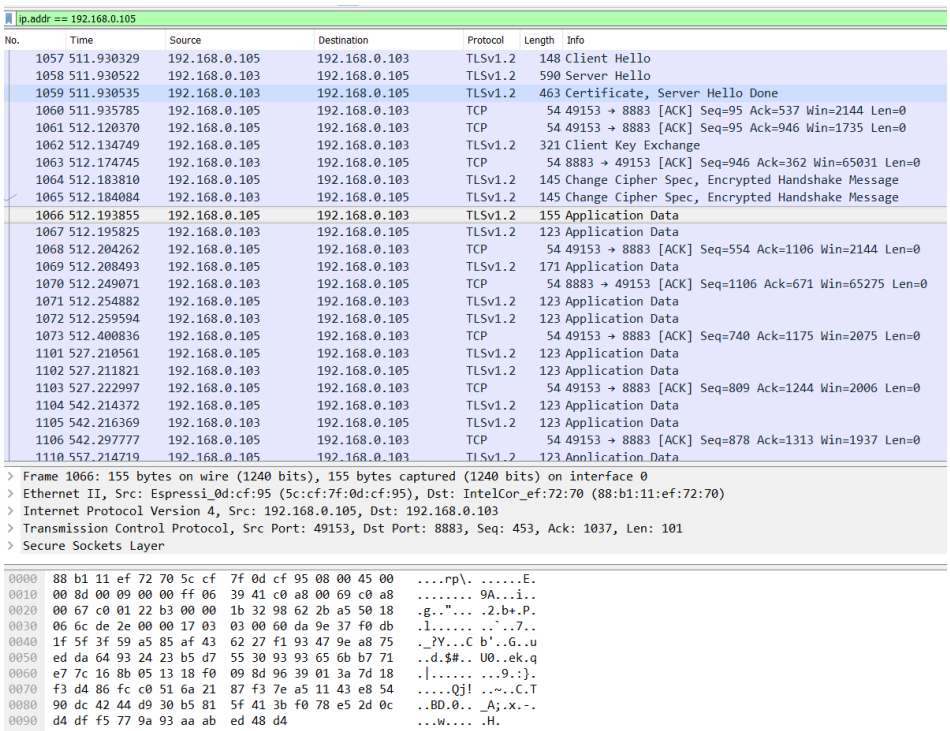
4.3 pav. MQTT neužšifruotos žinutės vaizdas „WireShark“ programinėje įrangoje

Kadangi įsitikinama, kad nešifruotas bendravimas tarp ESP8266 ir kompiuterio, naudojant MQTT protokolą, vyksta sėkmingai, buvo imtasi kitos darbo dalies – sukurti saugų komunikavimą, tam pasitelkiant SSL / TLS. Tam į kompiuterį reikia įdiegti OpenSSL [25], kad būtų galima sukurti serverio raktus, sertifikatus bei sertifikatų įgaliojimus CA (sertifikatų valdžia (išduodanti sertifikatus) (angl. *Certificate Authority*)). Pirmiausiai sukuriamas raktas, skirtas sertifikatų įgaliojimui CA. Tuomet sukuriamas CA sertifikatas *ca.crt* 5 metams, naudojant CA raktą *ca.key,* kuris buvo sukurtas prieš tai. Tam įvedami šalies, miesto, organizacijos pavadinimai, taip pat tikslus kompiuterio sričių vardų sistemos DNS (angl. *domain name system*) pavadinimas, šiuo atveju tai buvo LAPTOP-5NJT72A0. Toliau sukuriamas serverio raktas, kurį naudos brokeris. Vėliau pateikiama užklausa serverio sertifikato *server.csr* kūrimui, o pildant duomenis juos reikia pateikti lygiai tokius pat, kokie buvo rašomi *ca.crt* – ypač svarbu, kad atitiktų „*common name*“ laukelis, kuris privalo būti serverio DNS vardas. Šiuo atveju Windows programinė įranga veikia lokaliam tinkle, todėl buvo panaudotas kompiuterio vardas, tačiau tiktų ir IP adresas. Toliau panaudojamas CA raktą *ca.key* serverio sertifikato patvirtinimui ir pasirašymui – taip sukuriamas *server.crt* failas. Iš visų sukurtų naujų failų prireiks tik *server.crt, ca.crt* ir *server.key.* Jie nukopijuojami į „Mosquitto“ direktoriją */certs.* Toliau atliekami pakeitimai *mosquitto.conf* faile tam, kad būtų galima pradėti naudoti SSL / TLS. Tam būtina pakeisti „*default listener*“ reikšmę iš 1883 prievado į 8883, taip pat užpildyti laukelius „*cafile*“, „*certfile*“ bei „*keyfile*“, į juos įrašant tikslų kelią iki sertifikatų ir raktų direktorijos (*žr.* 4.4 pav.).



4.4 pav. *mosquitto.conf* failo konfigūravimas, norint pradėti naudoti SSL / TLS

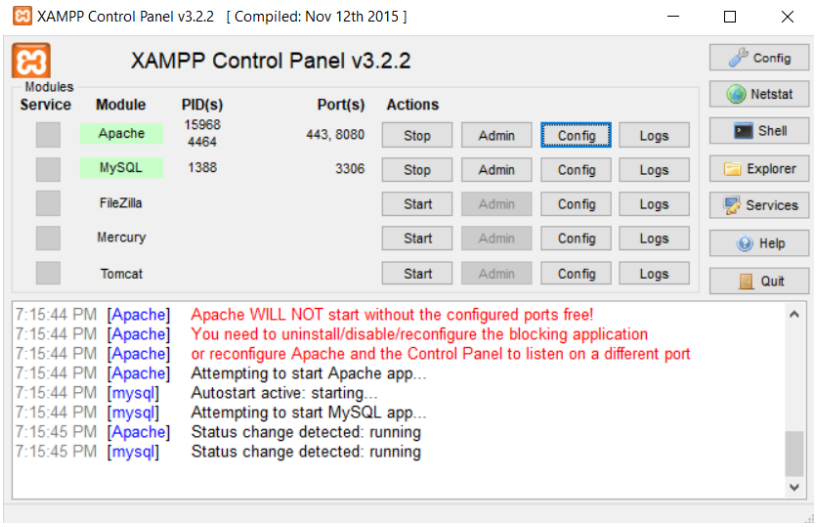
Atlikus visus pakeitimus, iš naujo paleidžiamas „Mosquitto“ brokeris. Norint įgyvendinti SSL / TLS saugą ir kitoje komunikacijų pusėje – mikrovaldiklyje ESP8266, „OpenSSL“ pagalba konvertuojamas *server.crt* failo formatas į dvejetainį *server.crt.der,* kadangi būtina jį perkelti į ESP8266. Kadangi „Arduino IDE“ platforma įprastai nepalaiko atskirų, su kodu nesusijusių bei neatitinkančių *.ino* formato failų įkėlimo į mikrovaldiklius, tam pasitelkiamas atskiras programos įskiepis – „*ESP8266FS*“ [26]. Jį naudojant galima įkelti į ESP8266 mikrovaldiklį tiek programinį kodą, tiek sertifikatą *server.crt.der*. Protokolo veikimo patikrinimui vėl naudojama „WireShark“ programinę įrangą ir įsitikinama, kad žinutės keliauja naudojant TLSv1.2 – užtikrinamas saugumas, jos užšifruotos ir nematomos, neperskaitomos svetimiems besiklausantiems asmenims (*žr.* 4.5 pav.).



4.5 pav. TLSv1.2 užšifruoto bendravimo vaizdas „WireShark“ programinėje įrangoje su MQTT protokolu

4.2. HTTP protokolo įgyvendinimas mikrovaldiklyje

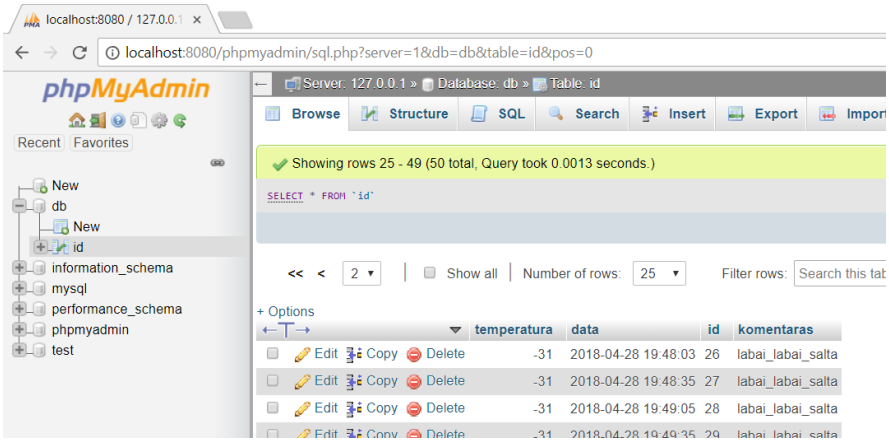
Mėginant įgyvendinti komunikavimą HTTP protokolu tarp mikrovaldiklio (šiuo atveju taip pat kliento) ir kompiuterio (serverio), privaloma pasirūpinti serverio, kuris palaikytų HTTP protokolą, įdiegimu. Tam buvo pasirinktas atvirojo kodo, nemokamos programinės įrangos paketas „XAMPP“ [27], kuris leidžia kompiuteryje tyrimo tikslais įsirengti vietinį serverį. Iš visų siūlomų programos modulių darbe buvo panaudoti tik du: „Apache“ – HTTP serveriui susikurti, bei „MySQL“ – susikurti duomenų bazei. Sėkmingai įdiegus „XAMPP“ ir atitinkamai modifikavus *httpd.conf* bei *httpd-ssl.conf* failus, matomas veikiantis vietinis serveris (*žr.* 4.6 pav.).



4.6 pav. „XAMPP“ valdymo ekranas

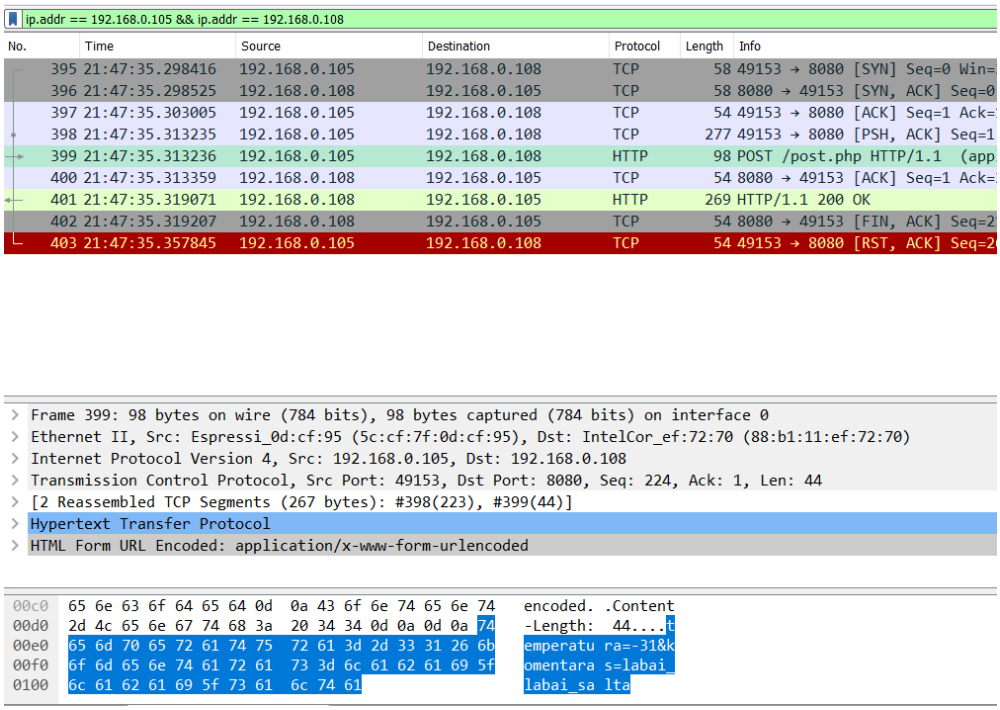
Pagal nutylėjimą, „XAMPP“ naudoja 80 prievadą nešifruotam HTTP komunikavimui, o 443 prievadą SSL / TLS apsaugotam komunikavimui, dar kitaip vadinamam HTTPS. Kadangi darbui naudojamame kompiuteryje pagal nutylėjimą 80 prievadas jau buvo užimtas, pasirinktas 8080 prievadas.

Norint patogiau matyti visus siunčiamus duomenis, sukurta „MySQL“ duomenų bazė (*žr.* 4.7 pav.), su tokiais laukeliais kaip žinutės identifikavimo kodas, data, siunčiama temperatūra ir pranešimas.



4.7 pav. „MySQL“ duomenų bazė

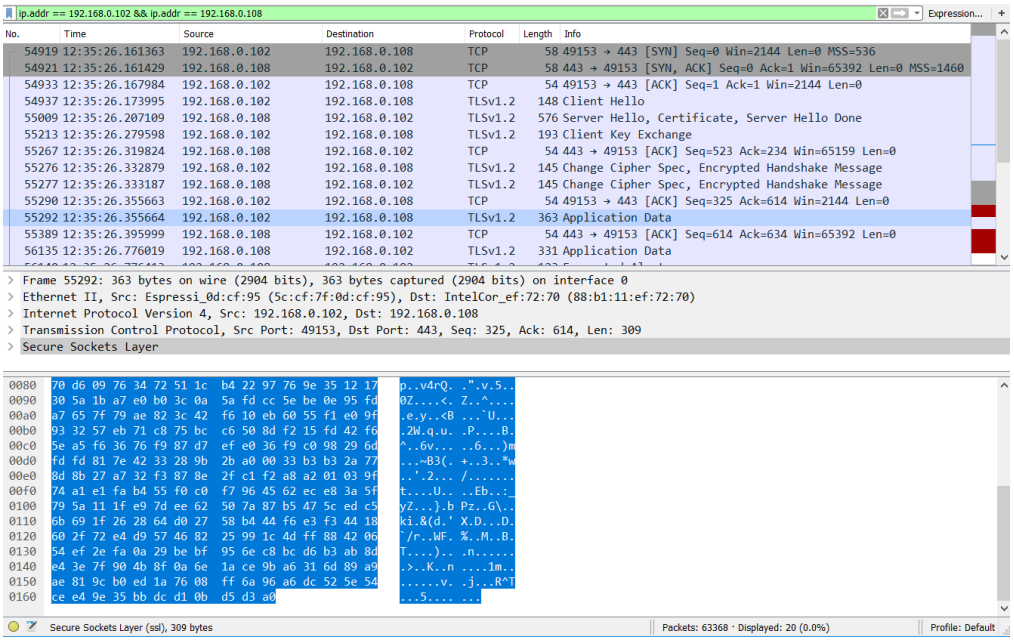
Lygiai taip pat kaip ir su MQTT, prieš pradedant įgyvendinti HTTPS, išbandomas nešifruotas komunikavimas tarp kompiuterio ir ESP8266 mikrovaldiklio. Vėl naudojant „Arduino IDE“ platformą ir „*esp8266 / Arduino*“ biblioteką [29], iš ESP8266 mikrovaldiklio išsiunčiamas pranešimas, kuriame parašyta atsitiktinė temperatūra bei komentaras. Nešifruotas komunikavimas buvo užmegztas sėkmingai (*žr.* 4.8 pav.).



4.8 pav. Neužšifruotos žinutės vaizdas „WireShark“ programinėje įrangoje, naudojant HTTP

Kadangi įsitikinta, kad nešifruotas bendravimas tarp ESP8266 ir kompiuterio, naudojant HTTP protokolą, vyksta sėkmingai, galima pradėti kitą darbo dalį – sukurti saugų komunikavimą, tam į pagalbą pasitelkiant SSL / TLS. Atsidarius https://localhost:443, atsisiunčiamas pačios pasirašytas (angl. *self-signed*) sertifikatas, iš kurio, šiuo atveju, reikia tik kontrolinio kodo (angl. *fingerprint*) SHA1. Jis pateikiamas programiniame kode, kad ESP8266 visuomet prieš siunčiant informaciją patikrintų, ar sutampa valdiklio ir vietinio serverio kontroliniai kodai.

Vėl norint patikrinti HTTPS protokolo veikimą, naudojama „WireShark“ programinė įranga ir įsitikinama, kad žinutės keliauja naudojant TLSv1.2 – jos neperskaitomos svetimiems besiklausantiems asmenims (*žr.* 4.9 pav.).



4.9 pav. TLSv1.2 užšifruoto bendravimo vaizdas „WireShark“ programinėje įrangoje su HTTP protokolu

Kadangi saugus komunikavimas pavyko tiek naudojant MQTT, tiek HTTP protokolus, galima pradėti jų efektyvumo tyrimą, palyginant tiek kiekybinius reikalavimus, tiek kokybines galimybes.

4.3. Protokolų šifravimo algoritmų modifikacijos

Atliekamo tyrimo metu, ESP8266 klientas serveriui pasiūlydavo keturis šifrų komplektus:

- TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c);
- TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d);
- TLS_RSA_WITH_AES_128_CBC_SHA (0x002f);
- TLS_RSA_WITH_AES_256_CBC_SHA (0x0035).

Iš šių šifrų komplektų serveris pasirinkdavo TLS_RSA_WITH_AES_256_CBC_SHA256, nesvarbu, koks protokolas būdavo naudojamas – ar MQTTS, ar HTTPS. Jeigu pageidaujamas dar didesnis saugumas, visuomet galima įdiegti ir išbandyti sudėtingesnius šifrų komplektus, tačiau reikėtų nepamiršti, kad tuomet tikrai padidės apsikeičiamų duomenų kiekis bei prailgės atliekamos operacijos laikas.

4.4. Protokolų kiekybinių parametrų palyginimas

4.4.1. Protokolų užimama vieta ir skaičiavimo išteklių naudojimas

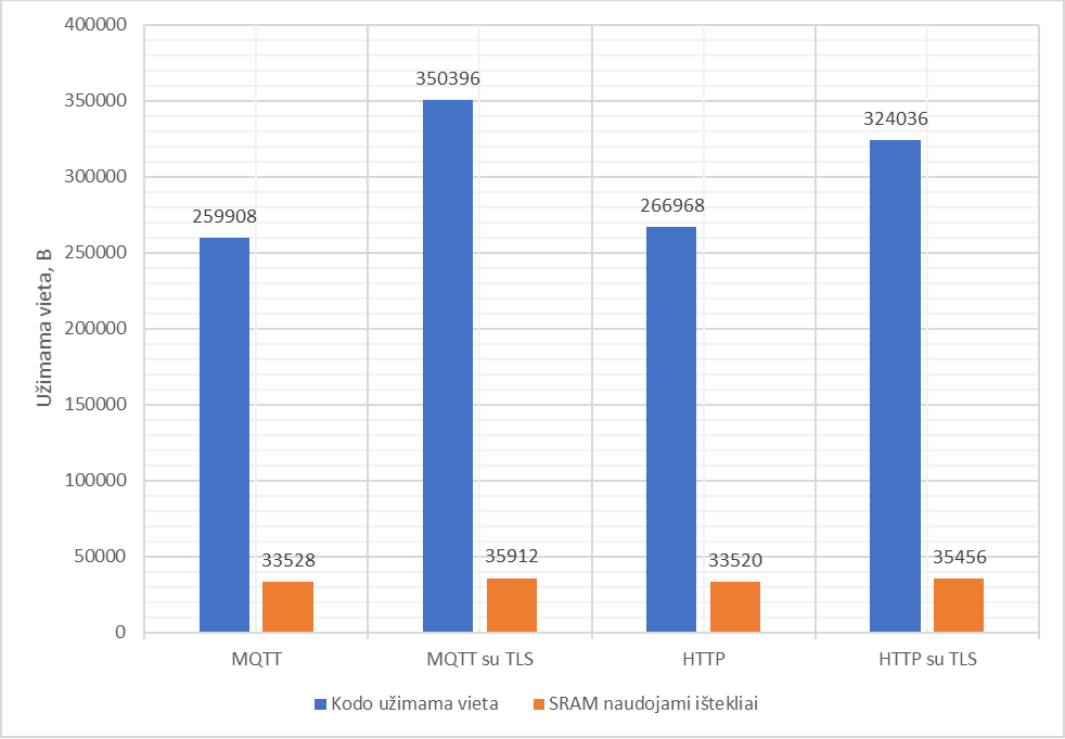
Pirmiausia apžvelgiama ir palyginama, kiek užima kiekvieno protokolo modifikacijos kodas ir kiek skaičiavimo išteklių jis naudoja. Šiai informacijai išgauti neprireikė jokių papildomų įrankių, kadangi tyrimas atliekamas naudojant „Arduino IDE“ platformą, kurioje šiuos duomenis galima gauti

kiekvieną kartą paleidus kodą, kadangi iš anksto pasirenkama naudojamo įrenginio modifikacija – tiek atminties dydis, tiek skaičiavimo ištekliai. Gautieji rezultatai pateikiami 4.1 lentelėje.

4.1 lentelė. Kodo dydis ir SRAM užimtumas valdiklyje

	Kodo užimama vieta, B	**Kodo užimama vieta, %**	**SRAM naudojami ištekliai, B**	**SRAM naudojami ištekliai, %**
MQTT	259908	52.01	33528	40.93
MQTT su TLS	350396	70.12 *(80*)*	35912	43.84
HTTP	266968	53.43	33520	40.92
HTTP su TLS	324036	64.85	35456	43.28

*****Nors MQTT su TLS kodas užima beveik tiek pat baitų kiek ir kitos modifikacijos, realiai užimamos vietos dydis gerokai padidėja dėl tiesiogiai į valdiklį įkeliamo sertifikato.



4.10 pav. Kodo dydis ir SRAM užimtumas valdiklyje

Beveik iš karto galime pastebėti, kad modifikacijos su TLS sauga užima 10 – 30 % daugiau vietos atmintyje negu paprastos ir neapsaugotos, tačiau SRAM užimtumas išlieka beveik toks pats.

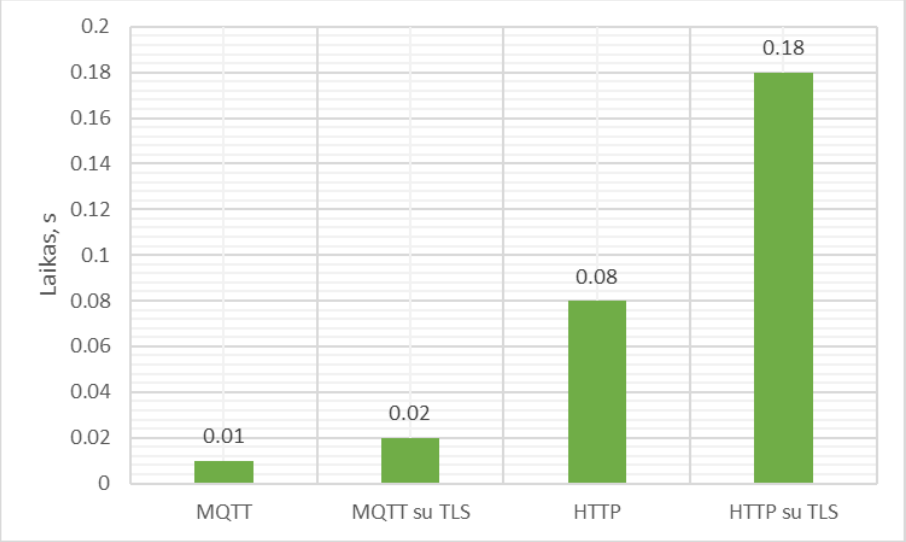
4.4.2. Protokolų greitaveika

Toliau darbe lyginama modifikuotų protokolų greitaveika. Kaip pavyzdys pasirinkta ± 90 baitų ilgio žinutė (pavyzdžiui, „*Temperatūra lygi 200. NESAUGUS. MQTT. Jūsų namuose kažkas vaikšto. Išsiųsta iš ESP8266*“), ir matuotas laikas, per kiek sekundžių išsiunčiamas ir gaunamas pranešimas.

Tam, kad būtų galima pamatuoti laiką, prireikia „Wireshark“ programos, kurioje galima stebėti, kiek laiko užtruko komunikavimas nuo pirmojo iki paskutiniojo duomenų paketo. Rezultatai buvo gauti bandymą pakartojus po 200 kartų kiekvienai protokolo modifikacijai ir išvedus vidurkį. Bandymo rezultatai pateikti 4.2 lentelėje.

4.2 lentelė. Žinutės išsiuntimo ir gavimo trukmė

	MQTT	**MQTT su TLS**	**HTTP**	**HTTP su TLS**
Žinutei išsiųsti ir gauti sugaištas laikas, s	0.01	0.02	0.08	0.18



4.11 pav. Žinutei išsiųsti ir gauti sugaištas laikas

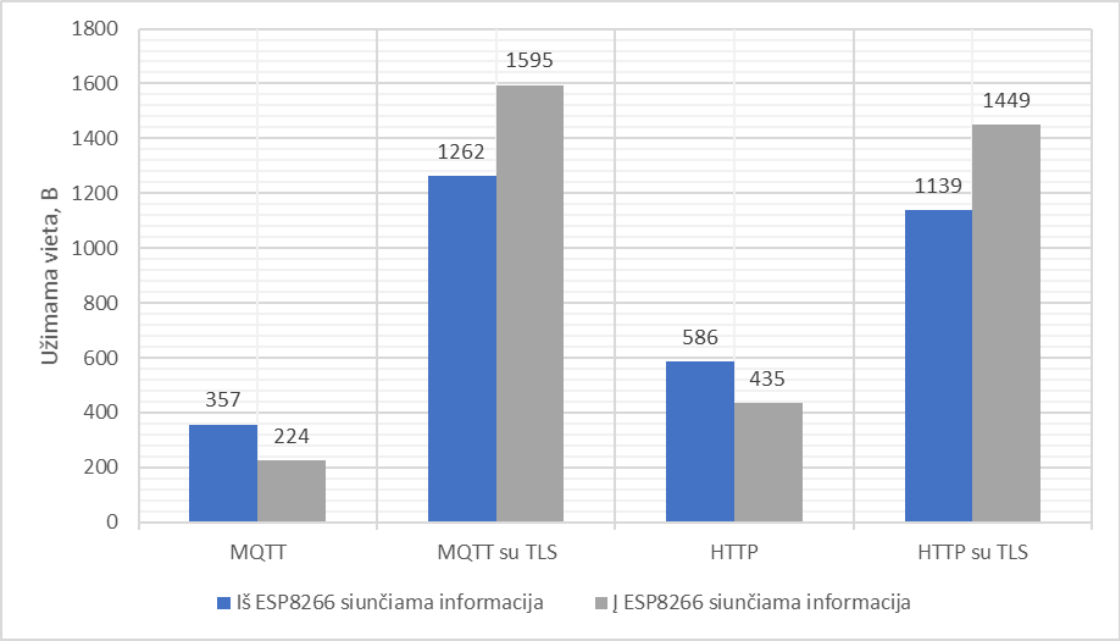
Šiuo atveju gaunami kur kas skirtingesni rezultatai, negu lyginant užimamos atminties kiekį ir resursų panaudojimą – MQTT protokolas už HTTP greitaveika skiriasi net 4 – 18 kartų, kas tampa itin svarbu, jeigu žinučių kiekis yra didelis arba juos reikia išsiųsti ne vienam, o net keliems serveriams.

4.4.3. Papildomas duomenų srautas apsaugant protokolus

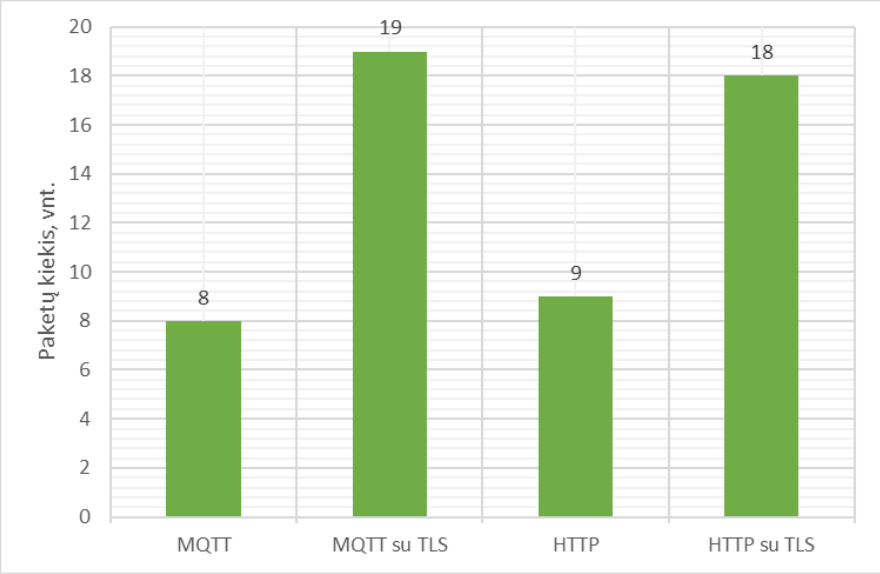
Norint gauti dar daugiau informacijos apie kiekvieno protokolo modifikacijos efektyvumą, atliktas tyrimas, norint palyginti papildomų duomenų kiekį – įeinančių ir išeinančių duomenų, papildomų paketų kiekį, kuris atsiranda įdiegus saugų komunikacijos metodą. Šiam tyrimui atlikti taip pat naudojama „Wireshark“ programą, kurioje galima pamatyti kiekvieno paketo informaciją – tarp jų ir dydį. Kaip pavyzdys ir vėl buvo naudojama ± 90 baitų ilgio žinutė (pavyzdžiui, „*Temperatūra lygi 200. NESAUGUS. MQTT. Jūsų namuose kažkas vaikšto. Išsiųsta iš ESP8266*“). Gautieji rezultatai pateikiami 4.3 lentelėje.

4.3 lentelė. Išsiunčiamos ir gaunamos informacijos dydis bei paketų kiekis

	Iš ESP8266 siunčiama informacija, B	Į ESP8266 siunčiama informacija, B	Paketų kiekis, vnt.
MQTT	357	224	8
MQTT su TLS	1262	1595	19
HTTP	586	435	9
HTTP su TLS	1139	1449	18



4.12 pav. Išsiunčiamos ir gaunamos informacijos dydis



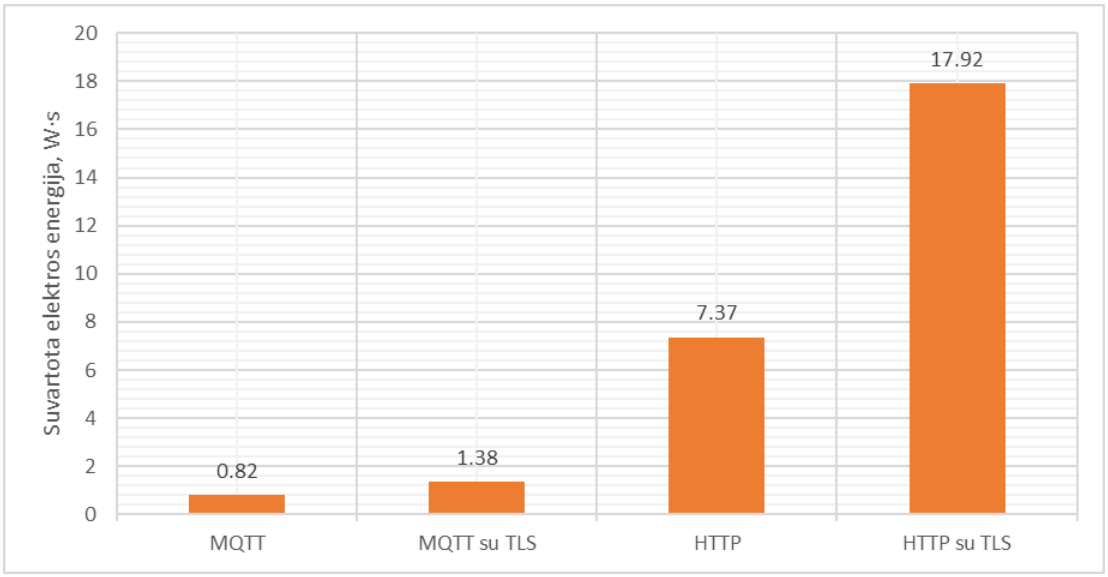
4.13 pav. Komunikacijai reikalingas paketų kiekis

4.4.4. Protokolų suvartojamos elektros energijos palyginimas

Galiausiai buvo atlikti suvartojamos elektros energijos tyrimai. Kadangi šiame darbe privaloma atsiminti, kad robotas, kuriame šis valdiklis teoriškai įdiegiamas, tikrai turėtų ribotus energijos išteklius, tad kuriamas saugios komunikacijos metodas turi būti energetiškai taupus. Norint išmatuoti suvartojamą elektros energiją, naudojamas USB įtampos bei srovės matuoklis „UNI-T UT658“ (nuotrauka bei schema pateikiama 3.3.2 skyrelyje). Kad rezultatai būtų kuo tikslesni, ta pati (aukščiau minėta) pavyzdinė žinutė siunčiama 200 kartų, ir stebima suvartojama elektros energija. Gautieji rezultatai pateikiami 4.4 lentelėje.

4.4 lentelė. Suvartojama elektros energija

	MQTT	**MQTT su TLS**	**HTTP**	**HTTP su TLS**
Suvartota elektros energija, W·s	0.82	1.38	7.37	17.92
Vidutinė naudojama srovė, A	0.08	0.09	0.09	0.1



4.14 pav. Suvartojama elektros energija

Jan Bartnitsky [35] naudodamas „RaspberryPi“ mikrovaldiklį ir atlikdamas panašius bandymus su MQTT bei HTTP protokolais prieina prie tokios pat išvados kaip ir šiame darbe – MQTT protokolas su TLS sauga pasižymi didesne greitaveika bei mažesniu elektros energijos suvartojimu nei HTTP protokolas su TLS sauga. Tačiau šis programuotojas atliko papildomą bandymą – norėdamas išsiųsti 1000 žinučių, vietoj 1000 siunčiamų žinučių su 1000 POST užklausų, jis pabandė išsiųsti 1 POST užklausą su joje esančiomis 1000 žinučių. Tokiu atveju HTTP su TLS našumas pralenkia MQTT su TLS – suvartojama mažiau elektros energijos bei užduotis atliekama greičiau. Vadinasi, jeigu robotui nebūtinas komunikavimas realiu laiku ir yra galimybė kaupti pranešimus, vėliau juos išsiuntus supakuotus į vieną paketą (angl. *batch*), tokiu būdu galima sutaupyti energijos ir laiko – žinutės išsiunčiamos 19 kartų greičiau, o persiunčiamų duomenų kiekis sumažėja beveik 100 kartų [35].

4.5. Metodo eksperimentinio tyrimo išvados

1. Eksperimentiniame tyrime buvo įgyvendintos iš viso keturios protokolų modifikacijos – MQTT ir HTTP protokolai be jokios papildomos apsaugos, taip pat MQTT ir HTTP su TLS protokolo apsauga. Protokolų modifikacijų saugumu buvo įsitikinta naudojant „Wireshark“ programą, per kurią buvo stebimi siunčiami duomenų paketai. Komunikavimas vyko naudojant TLS_RSA_WITH_AES_256_CBC_SHA256 šifro komplektą.

2. Pirmiausia buvo tiriama protokolų užimama vieta mikrovaldiklyje bei stebimas skaičiavimo išteklių naudojimas. Protokolų modifikacijos su TLS sauga užima 10 – 30 % daugiau vietos atmintyje negu paprastos ir neapsaugotos, tačiau SRAM užimtumas išlieka beveik toks pats. Nors daugiausiai vietos užėmė ir daugiausiai skaičiavimo resursų naudojo MQTT su TLS, žymaus skirtumo šiuo klausimu tarp HTTP ir MQTT protokolų rezultatų nenustatyta.

3. Vėliau buvo tiriama protokolų greitaveika. MQTT protokolas už HTTP greitaveika skiriasi net 4 – 18 kartų, tikslus skaičius priklauso nuo to, ar naudojama TLS apsauga. Taip pat buvo tiriamas papildomas duomenų srautas bei paketų kiekis, atsirandantis dėl TLS apsaugos įdiegimo. Komunikacijai naudojant MQTT su TLS bei HTTP su TLS didelio skirtumo tarp jų dviejų nenustatyta – tačiau šiuo atveju MQTT su TLS atsirado net šiek tiek didesnis papildomas duomenų srautas ir paketų kiekis negu HTTP su TLS, nes MQTT naudojo papildomą saugos elementą – kliento autentifikaciją.

4. Galiausiai buvo tiriamas protokolų naudojamas elektros energijos kiekis. Siunčiant tą pačią žinutę 200 kartų nustatyta, kad MQTT protokolas naudoja 8 – 22 kartus mažiau elektros energijos negu HTTP protokolas, tikslus skaičius priklauso nuo to, ar naudojama TLS apsauga.

5. Komunikacijų saugumo požiūriu protokolai beveik nesiskiria, nes abiejuose buvo įdiegta TLS apsauga. Siūloma protokolą rinktis nuo esamos situacijos.

5. IŠVADOS IR REKOMENDACIJOS

1. Atlikus literatūros analizę paaiškėjo, kad dauguma robotų, net naudojamų karinėje ar medicinos pramonėje, yra neparuošti apsiginti nuo atakų, nes komunikacijos įprastai vyksta neužšifruotos. Buvo pateiktos galimos grėsmės dėl neatsakingo požiūrio į komunikacijų saugą, taip pat apžvelgti populiariausi robotų komunikacijai naudojami protokolai, pateiktas HTTP ir MQTT protokolų funkcionalumo palyginimas.

2. Sudarant metodiką buvo nustatytos gairės, kurių privaloma laikytis kuriant saugų komunikavimo metodą robotui – dėl roboto autonominės prigimties ir suvaržytų išteklių būtina atsižvelgti į elektros energijos sąnaudas, komunikavimui sugaištą laiką bei protokolo naudojamus skaičiavimo išteklius.

3. Pasiūlyti du saugaus komunikavimo metodai, naudojant HTTP ir MQTT protokolus. Papildomai saugai įdiegti pasiūlyta naudoti TLS protokolą, kuris užtikrintų bendravimo žinutėmis konfidencialumą. Sprendimas įgyvendintas naudojant „Arduino IDE“ platformą, C / C++ kalbą, „Mosquitto“ brokerį bei „XAMPP“ paketą.

4. Tikrinant įgyvendintą metodą su „Wireshark“ programa įsitikinama, kad komunikavimas sėkmingai vyksta apsaugant jį su TLSv1.2, o žinutės išlaiko konfidencialumą ir neperskaitomos besiklausantiems iš šalies.

5. Palyginus jau apsaugotų protokolų kokybinius parametrus pastebima, kad MQTT gali pasiūlyti didesnį lankstumą saugumo lygmenų atžvilgiu, o ištyrus protokolų kiekinius parametrus pastebima, kad protokolų modifikacijų užimama vieta, naudojami skaičiavimo ištekliai bei papildomas duomenų srautas, atsirandantis apsaugant protokolus beveik nesiskiria, tačiau MQTT su TLS veikia net 9 kartus greičiau už HTTP su TLS, taip pat MQTT su TLS sunaudoja beveik 13 kartų mažiau elektros energijos negu HTTP su TLS. Tačiau jeigu nebūtinas komunikavimas realiu laiku ir yra galimybė kaupti pranešimus, vėliau juos išsiuntus supakuotus į vieną paketą (angl. *batch*), tokiu būdu galima sutaupyti energijos ir laiko, net ir pasirinkus HTTP protokolą.

6. Rekomenduojama saugiam komunikavimui protokolą rinktis atitinkamai nuo situacijos. Jeigu komunikacija vyksta tarp dviejų pusių (tik serverio ir vieno kliento) ir duomenys gali būti siunčiami nebūtinai realiu laiku, o kaupiami ir siunčiami supakuoti į vieną paketą (angl. *batch*), rekomenduojama naudoti HTTP su TLS. Naudojimo pavyzdys – patalpoje yra temperatūros ir drėgmės stebėjimo robotas. Jis fiksuoja duomenis visą dieną kas 6 valandas, o savaitės gale išsiunčia rezultatus į serverį. Jeigu komunikacija vyksta tarp keleto įtaisų (keli robotai / klientai ir keli brokeriai / serveriai), o duomenys turi būti siunčiami ir gaunami realiu laiku, rekomenduojama naudoti MQTT su TLS. Naudojimo pavyzdys – keliose namo kambariuose yra po aplinkos stebėjimo robotą. Juos įjungus, jie fiksuoja judesius aplinkoje ir realiu laiku siunčia rezultatus keliems šeimos nariams.

6. LITERATŪRA

[1] Matellan, V., Rodriguez-Lera, F.J. and Balsa, J. Cybersecurity in Robotic Systems. *ERCIM NEWS*. 2016, no. 106, pp. 44-45 [žiūrėta 2018-05-18]. Prieiga per: https://ercim-news.ercim.eu/images/stories/EN106/EN106-web.pdf ISSN 0926-4981.

[2] Finnicum, M. and King, S.T. Building Secure Robot Applications. *HotSec* [interaktyvus]. 2011 [žiūrėta 2018-05-18]. Prieiga per: https://www.usenix.org/legacy/event/hotsec11/tech/final_files/Finnicum.pdf

[3] Morante, S., Victores, J.G. and Balaguer, C., 2015. Cryptobotics: Why robots need cyber safety. *Frontiers in Robotics and AI*. 2015, no. 2, p. 23. Prieiga per doi: https://doi.org/10.3389/frobt.2015.00023

[4] Gage, D.M. Security Considerations for Autonomous Robots. *Security and Privacy, 1985 IEEE Symposium*. 1985, pp. 224. Prieiga per IEEE. ISSN: 1540-7993.

[5] Vuong, T., Filippoupolitis, A., Loukas, G. and Gan, D. Physical indicators of cyber attacks against a rescue robot. *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference*. 2014, pp. 338-343. Prieiga per IEEE. ISBN: 978-1-4799-2736-4.

[6] Dupont, B. Cybersecurity Futures: How Can We Regulate Emergent Risks?. *Technology Innovation Management Review*. 2013, vol. 3, no. 7, p. 6 [žiūrėta 2018-05-18]. Prieiga per: https://timreview.ca/article/700

[7] Mohanarajah, G., Hunziker, D., D'Andrea, R. and Waibel, M. Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*. 2015, vol. 12, no. 2, pp. 481-493. Prieiga per IEEE. ISSN: 1558-3783.

[8] Denning, T., Matuszek, C., Koscher, K., Smith, J.R. and Kohno, T. A spotlight on security and privacy risks with future household robots: attacks and lessons. *In Proceedings of the 11th international conference on Ubiquitous computing*. 2009, pp. 105-114 [žiūrėta 2018-05-18]. Prieiga per ACM: https://dl.acm.org/citation.cfm?id=1620545 ISBN: 978-1-60558-431-7

[9] *Most U.S. Drones Openly Broadcast Secret Video Feeds* [interaktyvus]. 2012 [žiūrėta 2018-05-18]. Prieiga per: https://www.wired.com/2012/10/hack-proof-drone/

[10] Hans, M., Graf, B. and Schraft, R.D. Robotic home assistant care-o-bot: Past-present-future. *In Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop*. 2002, pp. 380-385. Prieiga per IEEE. ISBN: 0-7803-7545-9.

[11] Thrun, S., Burgard, W. and Fox, D., 2005. Probabilistic robotics (intelligent robotics and autonomous agents series). Intelligent Robotics and Autonomous Agents. *The MIT Press*. 2005, no. 2, p. 47. ISBN: 0262201623.

[12] Thrun, S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A.B., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N. and Schulte, J. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *The International Journal of Robotics Research*. 2000, vol. 19, no. 11, pp. 972-999. Prieiga per doi: https://doi.org/10.1177/02783640022067922

[13] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G. and Lau, K. Stanley: The robot that won the DARPA Grand Challenge. *Journal of field Robotics.* 2006, vol. 23, no. 9, pp. 661-692. Prieiga per doi: https://doi.org/10.1002/rob.20147

[14] Shelby, Z., Hartke, K. and Bormann, C. *The constrained application protocol (CoAP)*. 2014. RFC 7252. Prieiga per: IETF. ISSN: 2070-1721.

[15] Amaran, M.H., Noh, N.A.M., Rohmad, M.S. and Hashim, H. A comparison of lightweight communication protocols in robotic applications. *Procedia Computer Science*. 2015, vol. 76, pp. 400-405. Prieiga per: Science Direct. ISSN: 1877-0509.

[16] Banks, A. and Gupta, R. *Mqtt version 3.1. 1. OASIS Standard* [interaktyvus]. 2014, [žiūrėta 2018-05-18]. Prieiga per: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.pdf

[17] *TLS overhead* [interaktyvus]. 2010 [žiūrėta 2018-05-18]. Prieiga per: http://netsekure.org/2010/03/tls-overhead/

[18] Stapko, T. Practical embedded security: building secure resource-constrained systems. *Newnes*. 2011. ISBN: 0080551319.

[19] Singhal, N. and Raina, J.P.S. Comparative analysis of AES and RC4 algorithms for better utilization. *International Journal of Computer Trends and Technology*, 2011, vol. 2, no. 6, pp. 177-181. Prieiga per: Research Gate. ISSN 2349-0829.

[20] Fluhrer, S., Mantin, I. and Shamir, A. Weaknesses in the key scheduling algorithm of RC4. *In International Workshop on Selected Areas in Cryptography*, Springer, Berlin, Heidelberg. 2001, pp. 1-24. ISBN 978-3-540-45537-0.

[21] Kinney, S.L. Trusted platform module basics: using TPM in embedded systems. *Elsevier.* 2006. ISBN: 0080465161.

[22] Katz, J., Menezes, A.J., Van Oorschot, P.C. and Vanstone, S.A. Handbook of applied cryptography. *CRC press*. 1996. ISBN: 9781439821916.

[23] Mei, Y., Lu, Y.H., Hu, Y.C. and Lee, C.G. Deployment of mobile robots with energy and timing constraints. *IEEE Transactions on Robotics*, 2006, vol. 22, no. 3, pp. 507-522. Prieiga per: Research Gate. ISSN: 1941-0468.

[24] *Mosquitto MQTT broker* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://mosquitto.org/download/

[25] *OpenSSL* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://www.openssl.org/source/

[26] *Arduino ESP8266 filesystem uploader* [interaktyvus]. 2016 [žiūrėta 2018-05-18]. Prieiga per: https://github.com/esp8266/arduino-esp8266fs-plugin

[27] *XAMPP Apache distributor* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://sourceforge.net/projects/xampp/

[28] *Arduino Client for MQTT* [interaktyvus]. 2016 [žiūrėta 2018-05-18]. Prieiga per: https://github.com/knolleary/pubsubclient

[29] *Arduino core for ESP8266 WiFi chip* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi

[30] Garcia-Morchon, O., Kumar, S., Struik, R., Keoh, S. and Hummen, R. *Security Considerations in the IP-based Internet of Things*. 2012. Prieiga per IETF.

[31] Becher, A., Benenson, Z. and Dornseif, M. Tampering with motes: Real-world physical attacks on wireless sensor networks. *In International Conference on Security in Pervasive Computing*, Springer, Berlin, Heidelberg. 2006, pp. 104-118. ISBN 978-3-540-33377-7.

[32] *ESP8266 ESP-01* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://www.elecfreaks.com/estore/esp8266-serial-wifi-module.html

[33] *CP2102 USB į TTL UART tiltas* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://anodas.lt/cp2102-usb-2-0-i-ttl-uart-konverteris

[34] *USB įtampos ir srovės matuoklis UT658 UNI-T* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://www.lemona.lt/?page=item&i_id=256752

[35] *HTTP vs MQTT performance tests* [interaktyvus]. 2018 [žiūrėta 2018-05-18]. Prieiga per: https://flespi.com/blog/http-vs-mqtt-performance-tests

7. PRIEDAI

7.1. priedas. ESP8266 mikrovaldiklio detalioji schema