LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

AUTORIZĀCIJA UN AUTENTIFIKĀCIJA TESTA VIDES AIZMUGURSISTĒMAI

Autors: Jēkabs Jānis Kalniņš

Studenta apliecības Nr.: jk16070

Darba vadītājs: pētnieks M.dat. Jānis Judvaitis

RĪGA, 2020

ANOTĀCIJA

Bakalaura darba “Autorizācija un autentifikācija testa vides aizmugursistēmai” mērķis ir analizēt dažādus lietu interneta protokolus un, kad tiks atrasts atbilstošākais balstoties uz testa gultnes prasībām, novērtēt autorizācijas un autentifikācijas metodes.

Praktiskās daļas testi tika veikti uz simulētas testa vides, izmantojot MQTT protokolu un Mosquitto un Paho MQTT implementācijas. Papildus testa programmatūrai, tika izveidots Paho abonēšanas klients, kā arī sertifikātu un klientu autorizācijas datu ģenerēšanas palīgprogramma, izmantojot Python programmēšanas valodu.

Analizējot iegūtos datus, tika konstatēts, ka testa gultnes autorizācijai un autentifikācijai labākais risinājums ir izmantot uz sertifikātiem bāzētu šifrēšanu un atsevišķu datubāzi priekš lietotāju autorizācijas un to lomu sadales.

Atslēgvārdi: MQTT, lietu internets, Mosquitto, Paho, testa gultne

ABSTRACT

AUTHORIZATION AND AUTHENTICATION FOR TESTBED BACKEND SYSTEM

In the bachelor's thesis “Authorization and authentication for testbed backend system” the goal is to analyze different internet of things and, after finding the most suitable based on testbed’s requirements, evaluate authorization and authentication methods.

Tests within the practical part of the work were executed on a simulated testbed, using MQTT protocol and Mosquitto and Paho MQTT implementations. In addition to the test software, a Paho subscription client and a certificate and client authorization data generation utility were created using the Python programming language.

Analyzing the obtained data, it was found that the best solution for testbed authorization and authentication is to use certificate-based encryption and a separate database for user authorization and role allocation.

Keywords: MQTT, internet of things, Mosquitto, Paho, testbed

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS 5
IEVADS 7
1. Lietu internets 9
2. Lietu protokolu apskats 11
2.1. MQTT vispārīgs raksturojums 11
2.1.1. Ziņojumu šifrēšana starp galapunktiem 13
2.1.2. Autorizācijas metodes 14
2.2. CoAP vispārīgs raksturojums 15
2.3. AMQP vispārīgs raksturojums 16
3. Testa gultne 18
4. EKSPERIMENTA ĪSTENOJUMS 22
5. REZULTĀTI 24
5.1. Autentifikācija 24
5.2. Autorizācija 25
5.3. Servisa kvalitāte 27
SECINĀJUMI 30
IZMANTOTĀ LITERATŪRA UN AVOTI 31
PIELIKUMI 32

APZĪMĒJUMU SARAKSTS

MQTT – Ziņojumu Ierindošanas Telemetrijas Transports

CoAP - Ierobežoto Lietotņu Protokols

AMQP - Paplašinātais Ziņu Ierindošanas protokols

‘Eclipse Foundation’ - Izstrādes vide modulāru starpplatformu lietojumprogrammām

Mosquitto – MQTT protokola brokeris

Paho – MQTT implementācija

C - Programmēšanas valoda

Java - Programmēšanas valoda

Windows - Operētājsistēma

Python - Programmēšanas valoda

Linux - Operētājsistēma

Brokeris – ziņu organizators MQTT protokolam

PSK - Iepriekš kopīgota atslēga

Central Authority - Centrālā Autoritāte

OpenSSL - Drošības bibliotēka, kas sarakstīta C programmēšanas valodā

QoS – Servisa Kvalitāte

TLS - Transporta Slāņa Drošība

UDP - Lietotāja Datorprogrammu Protokols

HTTP – Hiperteksta Transporta Protokols

RSA - Asimetriskās šifrēšanas algoritms

AES – Augsta līmeņa šifrēšanas standarts

ECC - Eliptisku līkņu kriptogrāfija

DTLS - sakaru protokols, kas nodrošina uz datugrammām balstītu lietojumprogrammu drošību

SASL – vienkāršas autentifikācijas un drošības slānis

GET – pieprasījuma metode

POST - Publicēt

PUT - Ievietot

DELETE - Izdzēst

Observe flag - Novērošanas karogs

RAM - Brīvpiekļuves atmiņa

Store-and-foward - Uzkrājnosūte

Content-based - Saturā balstīts

Point-to-point – Punkts uz punktu

Round-robin - Apļkārte

Sertifikāts - Digitāls dokuments, ko parasti izmanto autentificēšanai un drošas informācijas nodrošināšanai tīklā

USB hub - USB centrmezgls

IEVADS

Mūsdienās notiek tehnoloģiju strauja attīstība, kas atvieglo mūsdienu cilvēka dzīvi - sākot no mobilajām ierīcēm, ar kurām ir iespējams piekļūt gandrīz neierobežotam informācijas daudzumam, līdz termostatiem, kuri ir attīstīti tik tālu, lai spētu autonomi regulēt mājokļa temperatūru. Tādā veidā lietu internets dod iespēju cilvēkiem dzīvot un strādāt daudz ērtākā vidē, nepieciešamās darbības veicot veiklāk nekā ierasts. Papildus gudrajām ierīcēm, kuras izmanto mājoklī, un mājokļa automatizācijai, lietu internets ir svarīga sastāvdaļa biznesā, respektīvi tas palīdz uzņēmumiem reālajā laikā novērot sistēmu darbību, dodot informāciju par šo sistēmu veiktspēju un piegādes ķēžu loģistikas operācijām.

Lai pārbaudītu interneta ierīču veiktspēju, tiek izmantotas testa gultnes, kuras, lai novērstu apkārtējās vides trokšņu traucējumus sistēmai, ir izolētas no apkārtējās vides. Šī iemesla dēļ ir iespējams kontrolēti ievākt datus par jaudas un datu patēriņu, kā arī veikt veiktspējas un izturības pārbaudes.

Saziņas veikšanai starp lietu interneta galapunktiem nav noteikts standarts, pēc kura ir nepieciešams vadīties, lai izveidotu savu produktu. Tāpēc ir radušies dažāda veida protokoli, ar kuru palīdzību tiek nodrošināta ziņu nosūtīšana starp ierīcēm. Nav izveidots viens konkrēts protokols, kurš derētu priekš visām implementācijām, tāpēc ir nepieciešams izpētīt, kurš ir labākais risinājums savam lietu interneta tīklam pirms tā izveides.

Problēmas definīcija. Lai izveidotu testa gultni, ir zināmi saziņas protokoli, kurus ir iespējams izmantot, lai veiktu komunikāciju starp lietotāju un lietu interneta iekārtām. Toties, izmantojot gatavu risinājumu, netiek nodrošināts, ka šis risinājums būs optimāls priekš nepieciešamās operācijas.

Mērķa definīcija. Izpētīt populārākos lietu interneta protokolus un noskaidrot, vai ar tiem ir iespējams nodrošināt pietiekamas autentifikācijas un autorizācijas metodes priekš testa vides izveides.

Lai tiktu sasniegts mērķis, tika definēti darba uzdevumi:

- Izpētīt lietu interneta protokolu variācijas;
- Apkopot un aprakstīt informāciju par izstrādājamo testa gultni;
- Definēt testa gultnes prasības priekš autorizācijas un autentifikācijas;

- Veikt testus ar autorizācijas un autentifikācijas metodēm;
- No iegūtajiem rezultātiem izvēlēties visatbilstošāko risinājumu testa gultnei;

Pētījuma struktūra:

- Noskaidrot, kādas prasības ir testa gultnei;
- Izveidot testa programmatūra;
- Testa programmatūrā uzņemt mērījumus, cik ilgā laika periodā tiek izpildīts viens tests;
- Veikt testu atkārtošanu vairākas reizes, veidojot testu kopu, lai iegūtu vidējo laika periodu, kurā tiek izpildīti testi;
- Testus veikt ar visām autorizācijas un autentifikācijas variācijām;
- Iegūtos rezultātus apkopot, apstrādāt un izveidot tabulas;
- Pēc rezultātiem un testa gultnes prasībām izvērtēt labāko risinājumu.

1. LIETU INTERNETS

Angļu izgudrotājs Francis Ronalds (*Sir Francis Ronalds*) 1823. gadā, izveido pirmo elektrostatisko telegrāfu, savā dārzā ierokot 8 jūdzes garu vadu, kas tika izolēts stikla apvalkā. Katrā no galiem tika pievienots pulkstenis, kuros tika atzīmēti burti ciparu vietā. Pēc 9 gadiem tika izveidota pirmā īstā telegrāfijas ierīce ar 16 melnbaltām pogām [1]. Vēl pēc 12 gadiem, 1844. gadā, Semjuels Morze (*Samuel Morse*) nosūtīja pirmo lielas distances telegrammu no Vašingtonas Alfrēdam Vailam (*Alfred Lewis Vail*) Baltimorā ar tekstu no Bībeles "What hath God wrought" (" Ko Dievs ir darījis").

Pēc gandrīz 100 gadiem Nikolajs Tesla (*Nikola Tesla*) pieminēja intervijā "Colliers" žurnālā: "Kad bezvadu savienojumi tiks pilnībā īstenoti, visa pasaule tiks pārveidota par milzīgām smadzenēm, kas tā ir, jo visas lietas ir ritmiska kopuma daļiņas. Un instrumenti, ar kuriem mēs to panāksim, būs daudz vienkāršāki par mūsdienu tālruni. Cilvēks to varēs transportēt savā vestes kabatā.", kas patiesi bija akurāts pareģojums par mūsu esošu tehnoloģiju vidi, kur viss kļūst arvien saistītāks, un nevaram iedomāties savu dzīvi bez viedierīces pie sava sāna.

Tomēr pagāja vēl vairāk kā 100 gadi un pēc pirmā datoru tīkla "ARPANET" un pēc tam, kad Tims Bērnerss-Lī (*Tim Berners-Lee*) ierosina globālo tīmekli, 1990. gadā Džons Romkejs (*John Romkeys*) izveido pirmo "lietu interneta" ierīci. Tas bija tosteris, kuram bija iespējams ieslēgt jaudas padevi, izmantojot tīklu un mūsdienās plaši izmantoto TPC/IP protokolu.

2005. gadā "Violet" kompānija izgatavo "Nabaztag" ("zaķis" no armēņu valodas). Tas spēja pieslēgties pie tīmekļa, atskaņot mūziku, izpildīt balss komandas, sniegt informāciju par ziņām un laika apstākļiem, sazināties ar citiem "Nabaztag" lietotājiem, izmantojot balss, mūzikas un ausu kustību palīdzību. Un to visu spēja darīt neatkarīgi, neizmantojot saikni ar datoru, kas padarīja šo ierīci par tādejādi pirmo gudro asistentu līdzīgi kā "Amazon Alexa" un "Google assistant".

Lietas, kas ir piesaistītas pie interneta, starp 2008. un 2009. gadu pārsniedz cilvēku skaitu, kuri izmanto internetu, un "Lietu internets" ir dzimis, atsaucoties uz Cisco Sistēmu (*Cisto Systems*) aprēķiniem. Ierīču īpatsvars pret cilvēkum kuri lieto internetu, 2003. gadā bija 0.08 un pieauga līdz 1.84 2010. gadā, kad pie tīmekļa bija pieslēgtas 12.5 miljards ierīces.

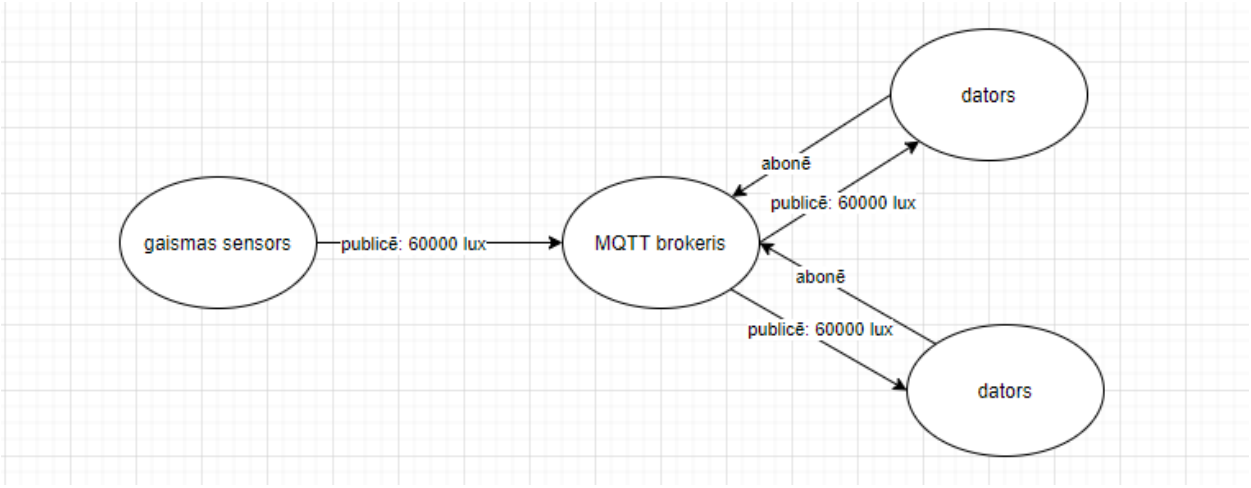
Lietas kļūst arvien veiktspējīgākas un tīkls, kas tās apvieno, kļuva daudz ātrāks ar 5G bezvadu tīkla ieviešanu. Tas ir līdz 20 reizēm ātrāks nekā ilgi izmantotais 4G LTE tīkls. Toties interneta ātrums nav nepieciešams vienīgi, lai ātrāk tiktu veikta lielu failu lejupielāde, bet arī, piemēram, autonomajos transporta līdzekļos, kur automobilim ir nepieciešams ātrā tempā apmainīt informāciju starp sevi un serveri.

2. LIETU INTERNETA PROTOKOLU APSKATS

Testa vide ir balstīta uz lietu interneta ideju, kurā vairākas ierīces jeb lietas, veic komunikāciju savā starpā, lai apmainītos ar informāciju. Ir vairāki protokoli, kuri nodrošina lietu internetu. Populārākie no tiem ir MQTT, CoAP un AMQP, kuri arī turpmāk šajā nodaļā tiks apskatīti, izvērtējot, kurš no tiem ir labākais, lai to varētu izmantot testa vides izveidē.

2.1. MQTT VISPĀRĪGS RAKSTUROJUMS

MQTT (*Message Queuing Telemetry Transport*) jeb Ziņojumu Ierindošanas Telemetrijas Transports ir protokols, kura pamatā ir publikācijas un abonēšanas princips. Komunikācija starp galapunktiem tiek nodrošināta izmantojot serveri (Brokeri) un nekāda savstarpēja saziņa starp ierīcēm (klientiem) nenotiek. Tā kā šis protokols patērē maz resursus to var īstenot zema enerģijas patēriņa ierīcēs, kurām ir limitēti resursi, kā mikrokontrolieri. Populāras MQTT implementācijas nodrošina 'Eclipse Foundation' ar nosaukumu 'Mosquitto' [3], kura ir uzrakstīta 'C' programmēšanas valodā un 'Paho', kas ir uzraksīta 'Java' programmēšanas valodā. []

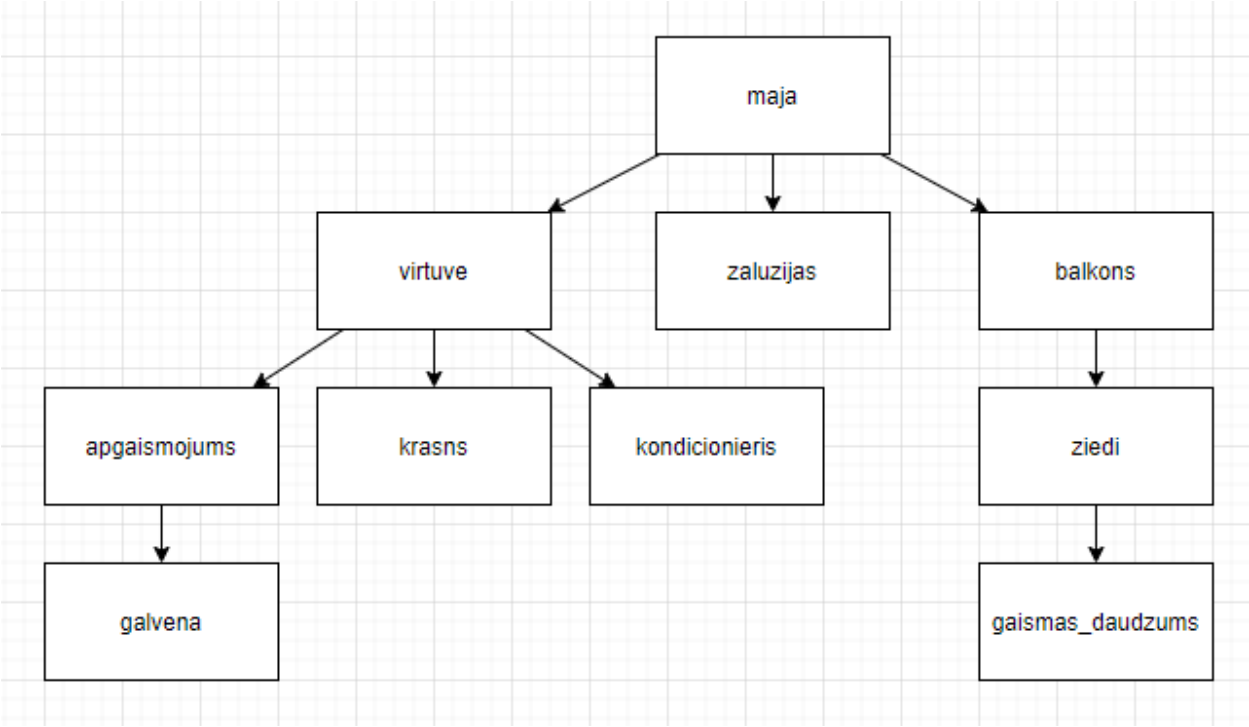


2.1. attēls. MQTT komunikācija starp klientiem un brokeri.

MQTT iespējas:

- Informācijas ieguvei no klienta klientam, ir nepieciešams nosūtīt abonēšanas pieprasījums brokerim. Brokeris nosūtīs informāciju klientam tikai tādā gadījumā, ja tam būs nepieciešamie dati par klienta abonēto tēmu. Informācija, kuru brokeris uzglabā, tiek saņemta no ierīcēm, kuras veic ziņu iesūtīšanu ar diviem parametriem – tēma un dati;

- Tēmām, pēc kurām tiek organizēta ziņu nosūtīšana konkrētajam klientam, ir sava struktūra. Tai ir līdzība ar mapju un datņu struktūru, kura atrodas plaši pielietotās failsistēmās kā NTFS un FAT. Šī struktūra tiek saukta par "koka strukturētu direktoriju" (*tree-structured directory*). Piemēram, virkne *'/maja/virtuve/apgaismojums/galvena'* ir derīga tēma pēc kuras var viegli vadīties, lai organizētu savu klientu tīklu. Toties, lai iegūtu informāciju par notikumiem virtuvē, nav jāabonē katra tēma, kas atrodas zem virknes *'/maja/virtuve'*, ir iespēja abonēšanas tēmā izmantot aizstājējzīmi. Zīmi *'+'* var izmantot, lai aizstātu vienu līmeni tēmā. Virkne *'/maja/virtuve/apgaismojums/+'* abonēs visas tēmas, kas ir tieši zem apgaismojuma. Turklāt zīmi *'#'* var izmantot, lai aizstātu sekojošos līmeņus tēmā. Virkne *'/maja/virtuve/#'* abonēs visas tēmas, kuras atrodas iekš virknes *'/maja/virtuve'* tēmas;



2.2. attēls. Koka strukturētu tēmas.

- Pievienojot jaunu klientu pie tīkla, ir jānotiek tūlītējai datu ievākšanai no tēmām, kuras ir abonētas. Tā kā sensoru mezgli (klienti) pārsvarā informācijas nosūtīšanu brokerim veic ar lielu laika intervālu, MQTT protokols veic klienta pēdējās iesūtītās ziņas saglabāšanu, tādejādi, jaunam klientam *pievienojoties* tīklam, tas nekavējoties saņem informāciju no brokera;
- Ir Iespējams iestatīt klienta testamentu, kas tiks publicēts, kad klients tiks negaidīti atslēgts no brokera. Šo ziņu klients var iestatīt tajā brīdī, kad klients slēdzas klāt pie brokera. Ziņas

struktūra ir tāda pati kā parastai ziņai un, tāpat kā parastā ziņa, tā tiks nosūtīta tiem klientiem, kuri šo tēmu ir abonējuši.

- Saziņas veikšanai ir iespējams izvēlēties Qos (*Quality of Service*) jeb Servisa Kvalitāti. Servisa kvalitātei ir iespējams noteikt 3 dažādus līmeņus, kur attiecīgi 0. - zemākais līmenis, 1. – vidējais un 2. – augstākais līmenis. 0. līmenis nozīmē, ka ziņa tiks nosūtīta, nepārliecinoties, ka tā ir piegādāta. 1. līmenis nozīmē, ka ziņa tiks nosūtīta vismaz vienu reizi, pārliecinoties, ka ziņa ir piegādāta un 2. līmenis nozīmē, ka ziņa tiks piegādāta tieši vienu reizi, izmantojot izaicinājum-rokaspiediena (*4-way handshake*) autentificēšanās protokola metodi. Ziņas var tikt nosūtītas, un klienti var veikt ziņu abonēšanu ar jebkuru no šiem līmeņiem, taču brokeris nosūtīs ziņu klientam ar augstāko iespējamo servisa kvalitātes līmeni;
- Klientam ir iespējams iestatīt ziņu, kuru brokeris nosūtīs abonētajiem klientiem, ja jaudas, tīkla zuduma vai citu iemeslu dēļ, klients tiks atslēgts no brokera;
- Lai novērstu trešo personu iesaistīšanos saziņā starp galapunktiem, ir iespējams izmantot divus šifrēšanas veidus – sertifikātus un PSK (*pre-shared-key*) jeb iepriekš kopīgota atslēga;
- Autentificēties izmantojot failu, kurā atrodas lietotājvārdi un paroles, vai izmantojot atsevišķu autentifikācijas moduli [1].

2.1.1. ZIŅOJUMU ŠIFRĒŠANA STARP GALAPUNKTIEM

Šifrēšana ir process, kurā notiek informācijas kodēšana, lai kāda trešā persona, izņemot ziņas saņēmēju, nevarētu apskatīt ziņas saturu. Ir divas galvenās šifrēšanas metodes - vienvirziena un divvirziena. Pēc vienvirziena ziņas šifrēšanas, iegūto informāciju vairs nevar atgūt, un tas ir noderīgi glabājot paroles, jo, ja pat kāds iegūst šifrēto paroli, viņš nevarēs noskaidrot, kāda tā bija pirms šifrēšanas. Divvirziena šifrēšana tiek izmantota, ja ir nepieciešams atgūt informāciju no šifrētās ziņas, kā, piemēram, tas tiek darīts drošas komunikācijas nodrošināšanā starp galapunktiem.

2.1.1.1. PSK

Vieglākais no variantiem, kuru implementēt, ir PSK. Serveris uzģenerē atslēgu pāri, kur abas atslēgas tiek nosūtītas klientam, un, kad klients pieslēdzas serverim, viņš veic servera publiskās atslēgas pārbaudi un nosūta ar šo atslēgu nošifrētus datus. PSK tiek izmantotas šifrēšanas

metodēs ar nosaukumu WPA-PSK un WPA2-PSK, kuras arī var tikt izmantotas priekš bezvadu tīkliem.

2.1.1.2. SERTIFIKĀTI

Otrs variants ir izmantot sertifikātus. Sertifikātu un atslēgu ģenerēšana notiek izmantojot 'openssl' bibliotēku. Uz servera tiek uzģenerēts CA (*central-authority*) jeb centrālā autoritāte, sertifikāts un privātā atslēga, ar kuru turpmāk tiks verificēti visi veidotie sertifikāti. Pēc CA izveides, tiek izveidota servera privātā atslēga un servera sertifikāts, kur sertifikātu ir nepieciešams parakstīt ar CA. Šāds process ir jāveic priekš katra klienta, kurš vēlas turpmāk nodrošināt saziņu ar serveri. Sertifikāta parakstīšanā ir jāizmanto CA privātā atslēga, jo klients pats nevar veikt sava sertifikāta parakstīšanu. Šādu procedūru veic servera administrators, un pēc sertifikāta parakstīšanas, tas tiks drošā veidā nogādāts uz klienta ierīci. Veicot saziņu ar serveri, klientam vajadzēs šifrēt serverim pārsūtamos datus ar savu privāto atslēgu, un autentificēt savu sertifikātu pie CA, lai pārbaudītu, vai tas vēl ir derīgs.

2.1.2. AUTORIZĀCIJAS METODES

Autorizācijas ir drošības mehānisms, kas tiek izmantots, lai noskaidrotu klientu privilēģijas vai piekļuves līmeni saistībā ar datoru programmām, failiem un servisiem. Saistībā ar MQTT, autorizācija tiek izmantota, lai ierobežotu, kādas tēmas klients var abonēt, un uz kādām tēmām tas var nosūtīt ziņas.

2.1.2.1. PAROĻU FAILS

Priekš vienkāršas autorizācijas ieviešanas ļoti labs variants ir izmantot paroļu failu. Lai to izmantotu, konfigurācijas failā ir jādefinē mainīgais '*password_file*', kur tā vērtība būs ceļš uz paroļu faila atrašanās vietu. Ierobežojums, ar kuru saskaras šis risinājums, ir tāds, ka brokeris tikai vienu reizi ielādē konfigurācijas failu, tas notiek brokera startēšanās laikā. Tāpēc arī paroļu fails tad tiks ielādēts un būs iespējams izmantot tikai tos lietotājus, kuri tajā atradīsies pie brokera palaišanas. Ja ir pievienots jauns lietotājs un ir nepieciešams šim lietotājam autorizēties, ir nepieciešams pārlādēt brokeri. Lai panāktu lietotāju lomu darbību, kā to nodrošina autorizācijas modulis, ir nepieciešams izveidot ACL (*Access control list*), jeb piekļuves kontroles failu, kurš konfigurācijas failā tiks definēts ar mainīgo '*acl_file*' un kā vērtība ir jāiestata ceļš uz ACL failu. Lietotāju tiesības tiek noteiktas izmantojot noteiktu sintaksi. Pirmajā rindā tiek definēts lietotājs, kuram tiks definētas tiesības, piemēram, '*user bob*'. Turpmākajās rindās tiek definētas šī lietotāja tiesības uz tēmām, kuras tas var abonēt un sūtīt ziņas. Lai atļautu lietotājam tikai nosūtīt ziņu uz

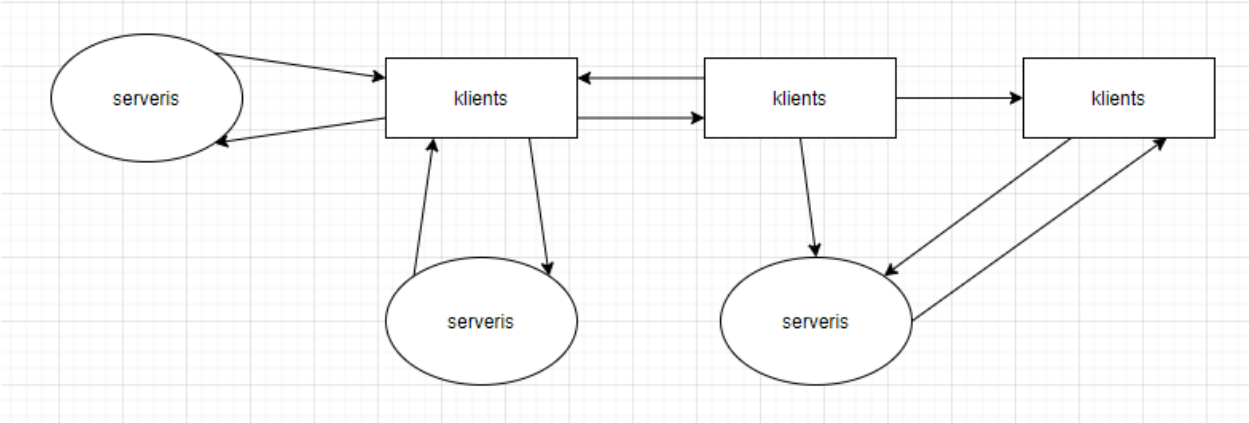
tēmu ‘*foo/bar*’, būs jāietur šāda sintakse ‘*topic write foo/bar*’, lai lietotājs spētu saņemt ziņas no šīs tēmas, ‘*write*’ vietā ir jāizmanto vārds ‘*read*’. Toties, lai lietotājs varētu gan saņemt informāciju no tēmas, gan publicēt ziņas tajā, nav nepieciešams rakstīt abas iepriekšējās rindas un jāizlaiž specificētais noteikums, piemēram ‘*topic foo/bar*’.

2.1.2.2. AUTORIZĀCIJAS MODULIS

Lielākām sistēmām labs veids kā nodrošināt autorizāciju ir izmantot autorizācijas modeli. Moduļa izmantošana ļauj skaidri atdalīt programmas sastāvdaļas, lai būtu vienkāršāk veikt programmas atkļūdošanu un koda otrreizēju izmantošanu citā projektā. Tāpat autorizācijas modulis ļauj vieglāk implementēt datubāzes savienojumu. Lai izmantotu moduli, nepieciešams konfigurācijas failā norādīt mainīgo ‘*auth_plugin*’ ar vērtību, kas ir ceļš uz moduli. Papildus konfigurācijas vērtības var būt nepieciešamas atkarībā no izvēlētā moduļa. Modulim ir jābūt kompilētam binārijam.

2.2. COAP VISPĀRĪGS RAKSTUROJUMS

CoAP (*Constrained Application Protocol*) jebIerobežoto Lietotņu Protokols līdzīgi kā HTTP, ir dokumentu pārsūtīšanas protokols. CoAP ir specializēts, lai varētu tikt izmantots priekš ierobežotām ierīcēm un tīkliem, kuros ir iespējami pārtraukumi. CoAP paketes ir daudz mazāka izmēra nekā HTTP, tās ir vieglāk izveidot un nav nepieciešams daudz RAM resursu, lai tas apstrādātu. Atšķirībā no MQTT, CoAP sūta datus izmantojot UDP nevis TLS, kur klienti ar serveri komunicē, izmantojot metodi, kuru angliski sauc par ’*connectionless datagrams*’. Mazi mikrokontrolieru tīkli tādejādi var veidot komunikāciju tikai izmantojot IP adresēšanu. Iespējama arī UDP apraide un multiraide priekš adresēšanas. Saziņai izmanto klienta servera modeli, kur klients nosūta pieprasījumu uz serveri, un serveris sūta atbildi. Klientam ir iespējams iegūt 4 veidu atbildes: GET, POST, PUT un DELETE. [5]



2.3. attēls. CoAP komunikācija starp serveriem un klientiem.

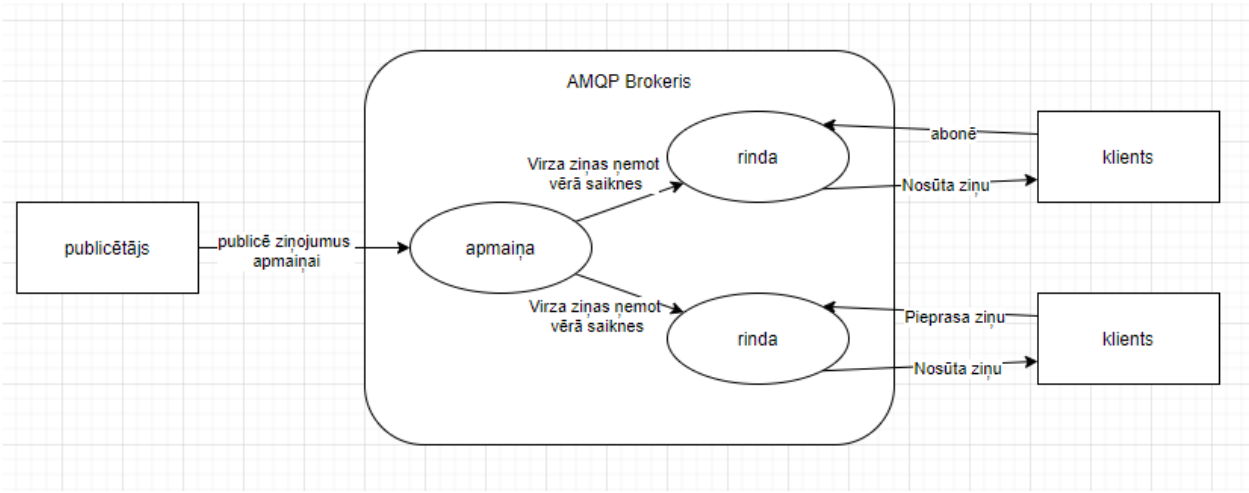
CoAP iespējas:

- Ir pieejamas divas ziņu servisa kvalitātes izvēlnes: apstiprināmās un neapstiprināmās. Ja ir nepieciešams apstiprināt ziņas piegādi, kad neviena daļa no ziņas netiks pazaudēta sūtīšanas procesā, atbildē tiks nosūtīts ziņojums, ka ziņa ir pienākusi;
- Satura pārrunas (*Content negotiation*) tiek izmantotas, lai norādītu, kādu satura tipu vēlas saņemt klients, nosūtot vēlamos formātus serverim;
- Ziņu šifrēšanai ir iespējams izmantot DTLS, kas nodrošina līdzīgu darbību kā TLS, bet darbojoties ar UDP. Šifrēšanu nodrošina ar publisko-privāto atslēgu algoritmu paveidiem, tādiem kā RSA, AES, ECC;
- Iestatīt resursu vērošanu, nosūtot GET ziņojumā novērošanas karogu (*observe flag*);
- Serveri piedāvā sarakstu ar iespējamajiem resursiem, kas atrodas iekš *"/.well-known/core"*. Katram saraksta elementam ir iespēja noskaidrot, kādus pakalpojumus tas sniegs un kādā satura tipā tas tiks sūtīts [2].

2.3. AMQP VISPĀRĪGS RAKSTUROJUMS

AMQP (*Advanced Message Queuing Protocol*) jeb Paplašinātais Ziņu Ierindošanas protokols ir binārs protokols ar mūsdienīgām iespējām: multi-kanālu iespējamība, satura pārrunas, asinhrona un droša saziņa. Tas iekļauj sevī visas MQTT funkcijas, izņemot testamentu un pēdējās-vērtības-rindas, bet tas ir veidots, lai tās var nākotnē ieviest, ja ir nepieciešamība. Pretēji MQTT, kur tīkls starp galapunktiem ir gandrīz privāts, AMQP ir paredzēts saziņai starp dažādām pusēm, kur saziņa notiek tīklā un infrastruktūrā, kas nav šo pušu kontrolē. Saziņa ir daudz plašāka nekā MQTT, jo ir iespējami uzkrājnosūte (*Store-and-foward*), saturā balstīts modelis (*content-based*),

punkts uz punktu (*point-to-point*), apļkārte (*round-robin*) un vēl daudzi citi ziņu ierindošanas veidi. Priekš drošības tiek izmantots SASL mehānisms, kas ļauj lietotājam izvēlēties drošības sistēmu, kāda tām ir nepieciešama. Izmantojot drošības starpniekserverus, ir iespējams izveidot hierarhiskos ugunsmūrus, kuri atļauj autentificēt lietotājus pirms tie izveido saziņas kanālu, kas pilnībā nošķir servera resursus. Ierīcēm, kurām ir ierobežota atmiņa un nepieciešama uzticama ziņu apmaiņa, ir iespējams kontrolēt, cik daudz ziņas tiks piegādātas, lai ierīce netiktu pārpludināta ar vairāk ziņām, nekā tā var apstrādāt vienlaicīgi. Neskatoties uz to, ka AMQP nodrošina daudz lielāku funkcionalitāti nekā MQTT, abus protokolus ir iespējams realizēt uz ierīcēm, kurām RAM ir zemāka par 64Kb. [6]



2.4. attēls. AMQP komunikācija starp klientu un serveri.

3. TESTA GULTNE

Svarīgi, lai veiktu pētījumus vai pasaulē laistu jaunu produktu, kas saistīts ar sensoru tikliem, ir notestēt laicīgi vai piedāvātais risinājums ir iespējams. Mūsdienās lietu interneta un Bezvadu Sensoru tehnoloģijas strauji attīstās, tāpēc testēšana ir kļuvusi pieejamāka un ērtāka. Ņemot vērā šos sasniegumus, ir radušās daudzas lētas un pieejamas testa vides, atkļūdošanas rīki un platformas priekš prototipēšanas. Papildus, izstrādāt lielus bezvadu sensoru tīklus (tādi, kas ir lielāki par vismaz 20 sensoru mezgliem) ir iespējams testa vidēs, jo, to testēšana reālos pasaules apstākļos, prasa daudz darba un šis process ir lēns.

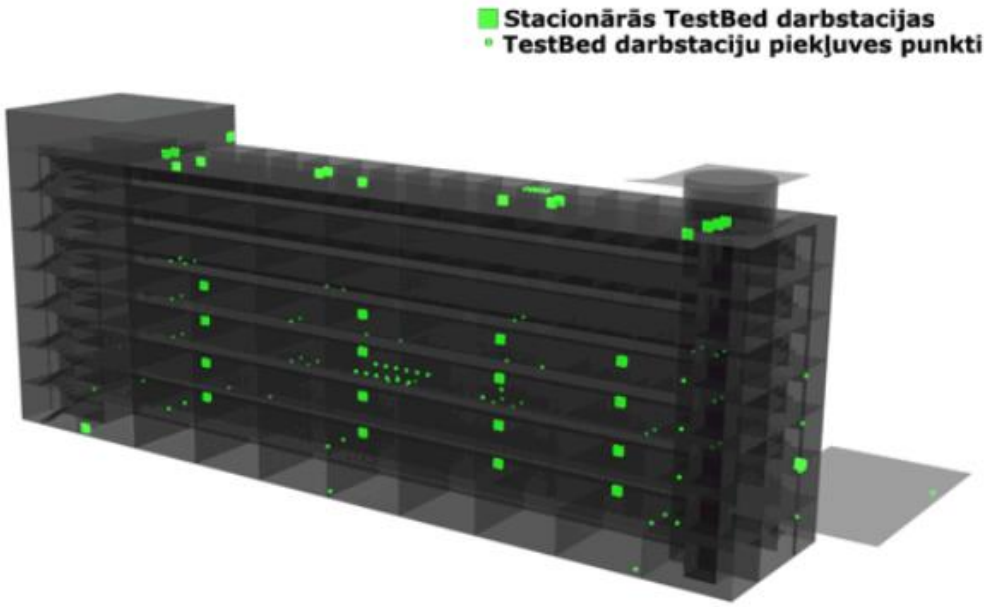
Testēšana reālos apstākļos ietver sevī daudz faktorus, ar kuriem būs jāsaskaras regulāri:

- Konfigurācija, pirmkoda izmaiņu augšupielāde visos sensoru mezglos;
- Bateriju dzīves ilgums;
- Bezvadu tīklu pārklājums;
- Apkārtējās vides ietekme.

Papildus testa gultnes dod iespēju:

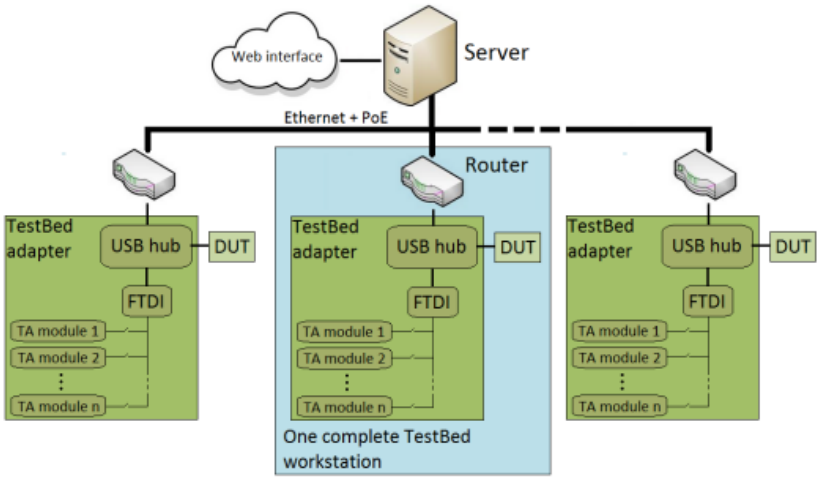
- Implementēt specifiskus lietotāja scenārijus (satiksme, enerģijas taupīšana, veselības aprūpe);
- Eksperimentēt, savienojot dažādas tehnoloģijas kopā ne produkta vidē;
- Definēt gala projekta prasības, izmantojot testa rezultātus;
- Atrast jaunas produktu un servisu iespējas, kas sākotnēji netika apsvērtas.

Darbā apskatītajai testa gultnei, kura atrodas EDI (Elektronikas un Datorzinātņu Institūts), ir vairāk kā 100 sensoru mezgli, kas iegūst vienu vai vairākus no turpmāk nosauktajiem datu veidiem - temperatūra, gaismas spilgtums gaisa mitrums un citi. To novietojums ir gan iekštelpās, gan ārpus tām, 7 dažādos stāvos, lai varētu nodrošināt dažādu testu veikšanu. Papildus vēl ir pieejamas50 mobilas testa stacijas, kuras ir iespējams novietot jebkur vidē. [7]



3.1. attēls. Stacionārās TestBed darbastacijas un to piekļuves punkti.

Testa gultne sastāv no centrālā servera un gultnes darba stacijām. Serverī glabājas visi dati, tiek darbināts web serveris ar kontroles paneli kā mājaslapu, lai varētu piekļūt pie paneļa no jebkuras ierīces, kura atbalsta *JavaScript*. Visi dati tiek veidoti vai ievākti no testējamās ierīces vai testa gultnes adaptera. Abas no šīm ierīcēm ir pievienotas rūterim, kurš ir testa gultnes darbstacija, ar USB savienojumu. Tehniski testējamā ierīce ir pieslēgta uzreiz pie testa gultnes adaptera, bet ir iespējams ievākt informāciju no abiem reizē, jo testa gultnes adapterī ir pieejams ’USB hub’, jeb universālās secīgās kopnes centrmezgls, caur kuru rūteris var pieslēgties pie testējamās ierīces. Rūteris testa gultnē tiek izmantots, lai kontrolētu četru loģiskā adaptera moduļu datu pārraidi. Tas var klausīties tikai vienā no moduļiem vienlaicīgi, tāpēc viens no rūtera pienākumiem ir nodrošināt, ka testa gultnes adapteri nosūta informāciju tikai tajā brīdī, kad tā tiek pieprasīta.



3.2.attēls. EDI testa gultnes arhitektūra.

Testa gultnes adapteris sastāv no 3 galvenajiem moduļiem: galvenais kontroles modulisis, jaudas patēriņa modul un datu apstrādes modulis. Mobilā testa gultne sniedz iespēju pievienot ārējus mezglus, kuri nav tieši pieslēgti pie EDI lokālā tīkla, bet ir pievienoti pie testa gultnes iekšējā tīkla, izmantojot virtuālo privāto tīklu. Izmantojot šo iespēju, vairākas testa gultnes dažādās vietās būs iespējams kontrolēt vienlaicīgi no vienas darba stacijas, atļaujot datu novērtēšanu un atkļūdošanu uz stacionāriem un mobiliem sensoru mezgliem. Situācija, kad ir liela tehniskā sagatavotība pirms strādājošas testa gultnes programmatūras izveides, ir radusies, jo šī ir otrā programmatūras implementācija [2].

Testa gultnē sensoru mezglos netiek veikta datu apstrāde, bet dati tiek pārsūtīti uz serveri. Servera galvenās funkcijas ir saglabāt datus, kas tiek saņemti no sensoru mezgliem, un tos pārsūtīt tālāk klientiem, kuri veiks datu apstrādi pēc viņu nepieciešamības. Tāpat serverim ir jāuzrauga sensoru mezglu tīkla darbība, piereģistrējot, kuri mezgli ir atslēgušies no tīkla, un jāziņo par nestandarta situācijām tīkla administratoram.

Testa videi ir jānodrošina informācijas apkopošanas un dalīšanās funkcijas. Dalīšanās nozīmē, ka ir jābūt iespējai pieslēgties klāt serverim no attālinātas vietas un jādod iespēja personai vai organizācijai, kas ir pieslēgusies pie šī servera iegūt informāciju par sensoru mezglu. Informācijai, kuru vēlamies iegūt, ir jābūt filtrētai pēc lietotāja nepieciešamības, lai netiktu sūtīti visi sensoru mezgla vai visa sensoru tīkla dati. Papildus, visām saziņām starp serveri un sensoru

mezgliem un starp serveri un lietotājiem ir jābūt drošām, lai persona no malas nevarētu nolasīt informāciju, kas tiek apmainīta starp šiem punktiem.

Lai gan tehniskais aprīkojums EDI testa gultnei ir nodrošināts, un uz sensoru mezgliem atrodas strādājoša programmatūra, taču, kad autors sāka strādāt pie testa gultnes, netika nodrošināts veids, kā klientiem pieslēgties pie testa gultnes attālināti, t.i. informāciju no sensoru mezgliem varēja iegūt tikai no EDI lokālā tīkla. Tāpat netika nodrošināta autorizācija, kas nebija nepieciešama, kamēr bija iespēja pieslēgties tikai no lokālā tīkla, jo tādejādi spēja pieslēgties tikai EDI darbinieki. Autorizācijas ieviešana dotu iespēju testa gultnei, atkarībā no lietotāja, sniegt dažādus pakalpojumus un novērst trešo personu ļaunprātīgu testa gultnes izmantošanu. Netika arī nodrošināta datu šifrēšana starp galapunktiem. Tāpat kā autorizācija, tā nebija tieši nepieciešama, kad bija iespēja pieslēgties no lokālā tīkla, bet, lai pasargātu klienta datus, tādus kā paroles un lietotājvārdus, no iespējamas trešās personas piekļuves, ir nepieciešams atrast risinājumu datu šifrēšanai.

4. EKSPERIMENTA ĪSTENOJUMS

Darba eksperimentālajai daļai priekš galapunktu saziņas tika izmantots MQTT protokols. Tas tika izvēlēts dēļ tā vienkāršās un minimālistiskās komunikācijas. Netika izvēlēts CoAP, jo ir svarīgi ka sensoru mezglam ir nepieciešams nosūtīt ziņu tikai vienam klientam (serverim). CoAP galvenokārt ir ierīce-ierīce komunikācija, kur praktiski būtu nepieciešams nodrošināt saziņu starp ierīcēm un nebūtu tik vienkāršu risinājumu, kā ieviest autorizāciju vai tēmu aizliegumus, kā tas ir ar MQTT protokolu, kur visu ir iespējams konfigurēt brokerī. AMQP toties iekļauj visas MQTT iespējas, bet priekš šī pētījuma autors izvēlējās MQTT, jo tas izpilda visus nepieciešamos uzdevumus, un nebija nepieciešamas daudzās iespējas, kuras nodrošina AMQP

Kā brokeri autors izvēlējās izmantot Mosquitto un kā klientu - Paho MQTT implementāciju. Abas ir atvērtā koda bibliotēkas un izdevējs ir *Eclipse*. Paho implementācijai tiek arī dota priekšroka, jo tiek piedāvāta jau gatava Python bibliotēka, kas ir svarīgi, jo sensoru tīkla iekārtas un testa gultnes aizmugursistmēmas pārsvarā darbojas ar Python valodu.

Testu veikšanai tika izmantota simulēta testa gultnes vide, nevis testēts uz EDI esošās sistēmas, jo institūta iekšējo noteikumu dēļ, praktikantiem tika liegta piekļuve institūta telpām. Testi tika veikti uz Linux platformas ar Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz procesoru un 8Gb RAM. Lai tiktu notestētas testa gultnes autorizācijas un autentifikācijas iespējas testa gultnē, tika izveidoti šādi faili:

- mqtt.conf (pārsaukt mqttdb.conf) – mosquitto konfigurācija;
- aclfile – piekļuves kontroles fails;
- passwd – fails kurā atrodas lietotājvārdi un paroles;
- auth-plugin.so – autorizācjias modulis;
- pskfile – fails, kurā atrodas lietotājiem izdalītās atslēgas ar viņa lietotājvārds;
- subscriber.py (pārsaukt user.py) – abonētāja programma, kas pēc “mosquitto_sub” principa pieslēdzas pie brokera;
- ca.crt – centrālās autoritātes sertifikāts;
- ca.key – centrālās autoritātes privātā atslēga;
- server.cr t – servera sertifikāts;
- server.key – servera privātā atslēga;

- client.crt – klienta sertifikāts;
- client.key – klienta privātā atslēga.

Mosquitto brokeris tiek inicializēts, konfigurācijas failu padodot tam kā parametru. Konfigurācijas failā tiks ievietoti autentifikācijas un šifrēšanas parametri attiecīgi testam, kas tajā momentā tiks veikts. Priekš visiem testiem tika izmantots 8883 ports, kuru izmanto, lai nodrošinātu TCP servisu un tādejādi pirmo soli priekš drošas ziņu pārsūtīšanas. Netika izmantots pēc noklusējuma iestatītais 1883 ports, jo porta nomaiņa uz 8883 neievieš veiktspējas mazinājumu. Ziņas uz brokeri netika sūtītas no atsevišķas programmas, kā tika veidota abonētāja programma, bet tika nosūtītas pa taisno no testa programmas, jo papildus atmiņa, kas tika patērēta, palaižot atsevišķu programmu, pārāk noslogoja procesoru, ietekmējot testa rezultātus.

Katrā testā tika simulēti 100 sensoru mezgli, izmantojot Paho '*single*' metodi, ar kuru tika sūtīti ziņojumi brokerim. Katrs no sensoru mezgliem tika simulēts izmantojot pavedienus, kur katrs pavediens ir viens mezgls, lai sensoru mezgli spētu darboties paralēli viens ar otru. Kā arī, lai izpētītu visus variantus, tika veikti testu raundi, kuros ietilpa 100 testi, ar katru konfigurāciju, kur katrs mezgls nosūtīja 100 ziņas. Kopā tika veikti 18 testu raundi. Galvenie 6 raundi bija variācijas starp autorizācijas un autentifikācijas metodēm, kur katram no šiem testu veidiem, tika veikti divi papildus testi, kur tika mainīts QoS līmenis.

Nepieciešamais ziņu izmērs tika izvēlēts, ņemot vērā esošā tīkla sensora mezgla seriālās līnijas savienojuma ātrumu. Seriālās līnijas ātrums pašlaik ir 3840bps, kur 1Kbytes/s sekundē ir 8192bps. Tādejādi ir iespējams aprēķināt ka sensora mezgla ātrums ir 4.6875 1Kbytes/s. Tā kā mezgla ātrums, apaļojot uz leju, ir 4.6Kb/s, izvēlētais ziņas ātrums ir 46 simboli, un ziņas tiks nosūtītas ar 10ms pārtraukumu, tādejādi simulējot mezgla savienojumu.

5. REZULTĀTI

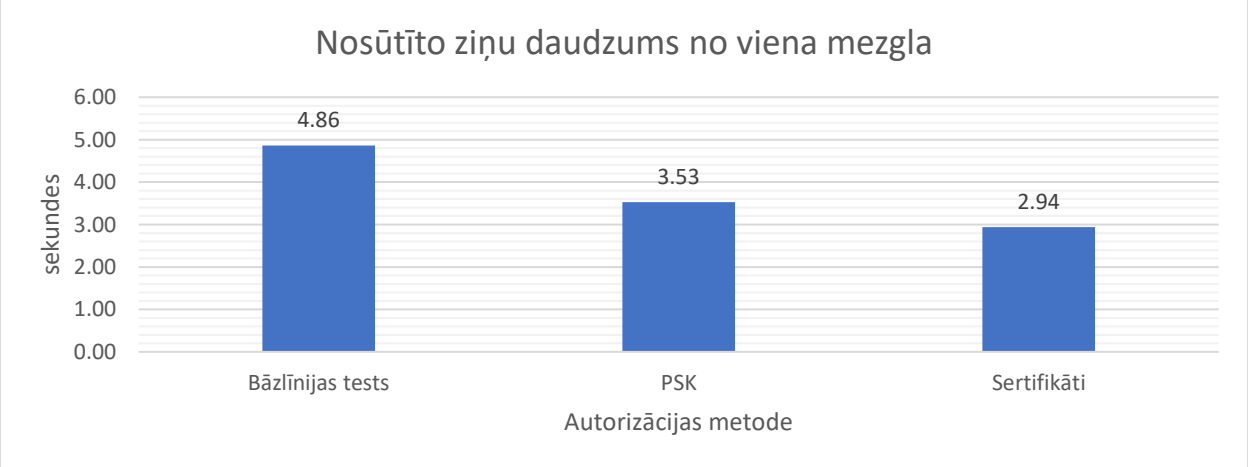
Kā bāzlīnija tika izmantota tās testu kopas rezultāti, kuri neizmantoja autentifikācijas un autorizācijas moduļus, papildus, tika izmantots QoS 0. līmenis. Bāzlīnijas testos, bez autentifikācijas un autorizācijas moduļa izmantošanas, vidējais laiks kurā tika izpildīti testi, bija 20.557 sekundes, kur viena ziņa no visiem sensoriem tika nosūtīta 2.056 milisekundēs, un sekundē no viena sensora mezgla tika nosūtītas 4.8645 ziņas.

5.1. AUTENTIFIKĀCIJA

Sākotnējais modulis, kas tika izvērtēts bija šifrēšanas sistēma. MQTT piedāvā divus risinājumus PSK un sertifikātus. Laiks, kurā tiek uzģenerētas un izdalītas atslēgas, netika ņemts vērā, jo tas notiks tikai vienreiz testa vides inicializēšanas laikā.

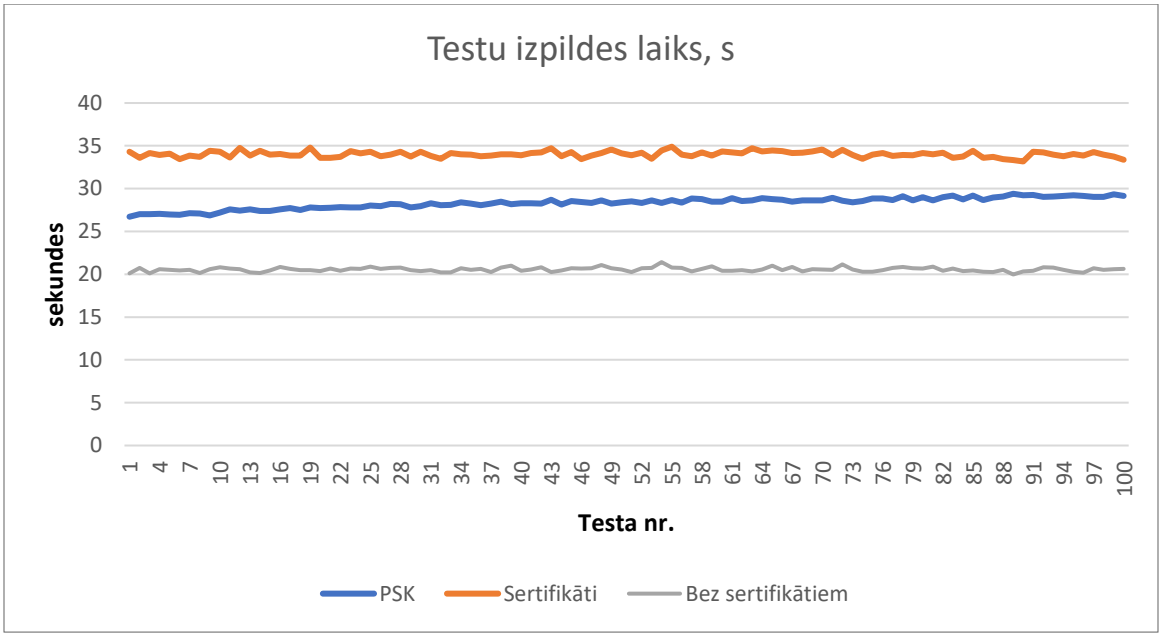
Veicot testus ar sertifikātu autentifikāciju, vidējais laiks, kurā izpildījās tests bija 34.024 sekundes, kur viena ziņa no visiem sensoriem tika nosūtīta 3.402 milisekundēs un sekundē no viena mezgla tika nosūtītas 2.939 ziņas. Tas ir liels izpildes laika kritums salīdzinoši ar bāzes testu, kur sertifikātu ieviešana samazināja ātrdarbību par 37.81%.

Veicot testus ar PSK, ātrdarbība tika novērota labāka salīdzinoši ar sertifikātu rezultātiem, kur PSK vidējais laiks, kurā izpildījās tests bija 28.328 sekundes, kur viena ziņa no visiem sensoriem tika nosūtīta 2.833 milisekundēs, un sekundē no viena mezgla tika nosūtītas 3.530 ziņas. Tas ir par 20.02% labāk nekā autentifikācija ar sertifikātu metodi un 17.79% lēnāk nekā bāzlīnijas testi.



5.1. attēls. Laiks kurā tiek nosūtīta viena ziņa, izmantojot dažādas autentifikācijas metodes.

Šādu atšķirību ātrdarbībā starp abām metodēm var izskaidrot ar to, ka priekš sertifikātu metodes ir jāveic sertifikāta pārbaude un jāizmanto sarežģītāks algoritms priekš ziņas šifrēšanas un atšifrēšanas. Priekš sertifikātiem ir jāveic 6 failu nolasīšana – 3 klientam un 3 serverim. Šie faili ir galapunkta privātā atslēga un sertifikāts un centrālās autoritātes sertifikāts.



5.2. attēls. Laiks kurā tiek izpildīts viens testu raunds.

Papildus, priekš sertifikātu metodes tika izmantota Paho bibliotēka, bet priekš PSK Mosquitto bibliotēka. Tas pēc primārajiem testiem, salīdzinot sertifikātu metodi starp abām bibliotēkām, parādīja, ka Paho bibliotēka izpildīja testus par 5 sekundēm ilgāk. Kā arī osquitto bibliotēkas iegūtie rezultāti katru reizi atšķīrās, atkārtojot testus. Rezultātu nevienveidību ir arī iespējams novērot 7. attēlā, kur var novērot, ka testa izpildes laiks vidēji ar katru reizi nedaudz palielinās.

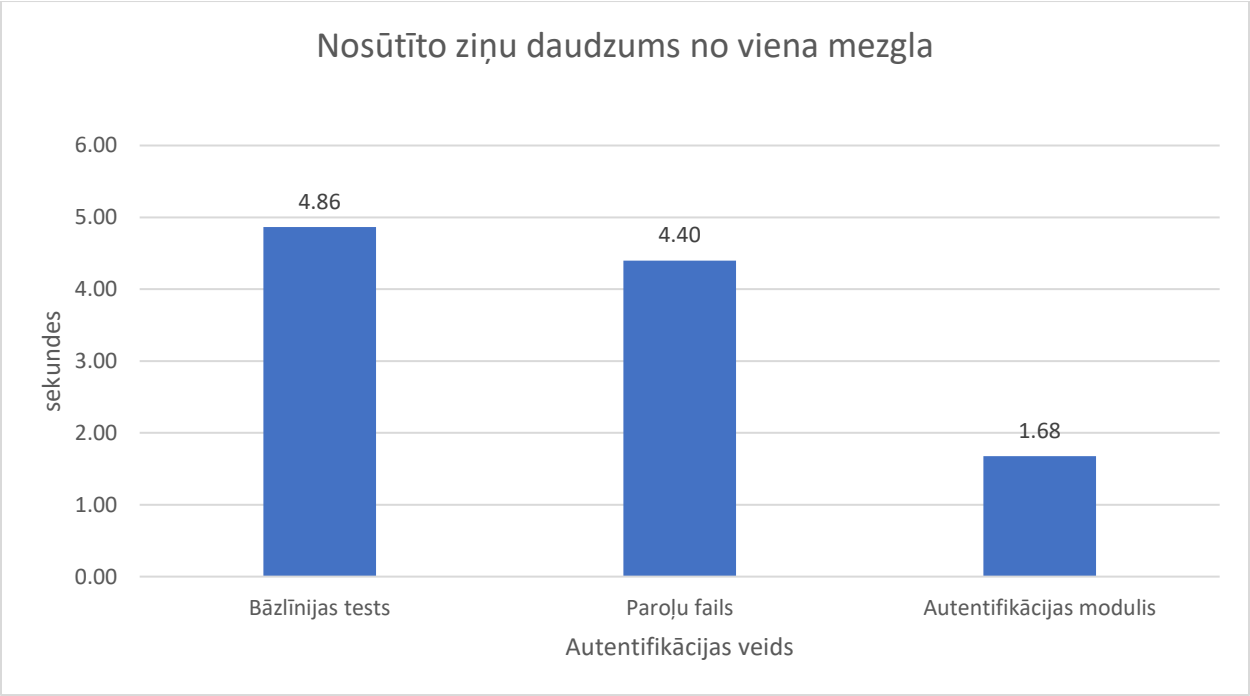
5.2. AUTORIZĀCIJA

Otrs modulis, kuru ir nepieciešams pievienot, ir lietotāju autorizācija. Iebūvētais risinājums ir izmantot failu, kurā glabājas paroles ar lietotājvārdu. Papildus vēl ir iespēja izmantot atsevišķu autorizācijas moduli, kas dod iespēju izmantot datubāzi. Abi šie moduļi ir jānotestē lietojot un nelietojot šifrēšanu, lai noskaidrotu to mijiedarbību.

Neizmantojot iepriekš aprakstītos ziņas šifrēšanas paņēmienus, paroļu failu izmantošana ir ātrākais risinājums, kur vidējais laiks, kurā izpildījās tests bija 22.747 sekundes, kur viena ziņa no

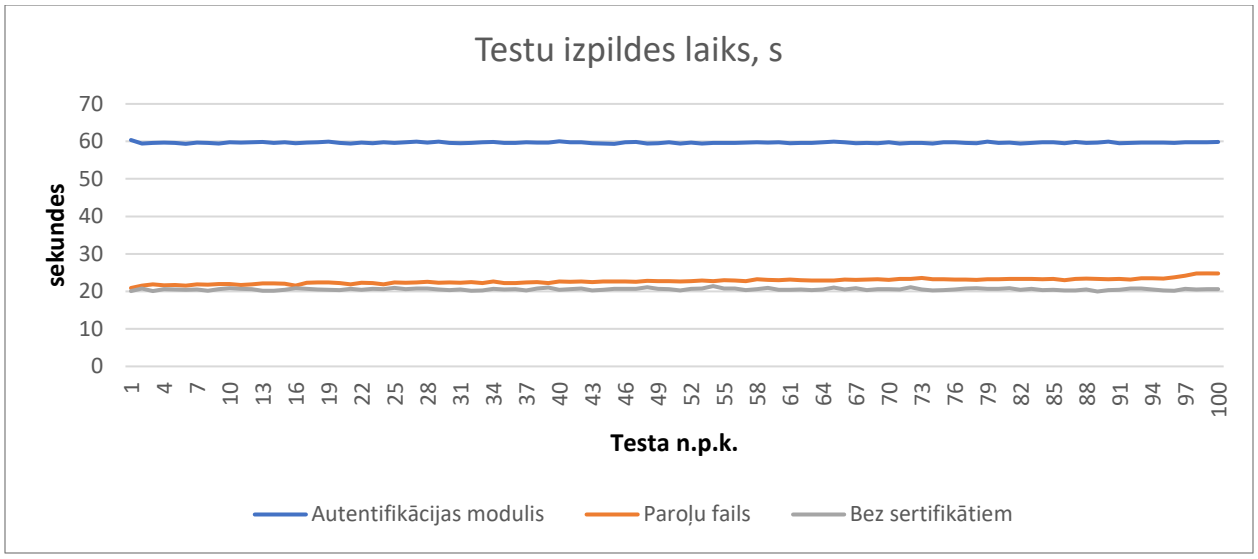
visiem sensoriem tika nosūtīta 2.274 milisekundēs, un no viena mezgla tika nosūtītas 4.396 ziņas. Salīdzinoši bāzlīnijas testiem, testi tika izpildīti par 9.63% lēnāk.

Izmantojot autorizācijas moduli, tika novērota daudz lielāka novirze no bāzlīnijas testa. Moduļa vidējais laiks, kurā izpildījās tests bija 59.663 sekundes, kur viena ziņa no visiem sensoriem tika nosūtīta 5.966 milisekundēs, un sekundē no viena mezgla tika nosūtītas 1.676 ziņas. Tas ir 2.90 reizes lēnāks ziņu pārsūtīšanas ātrums salīdzinot ar bāzlīnijas testu un par 61.87% lēnāks par paroļu faila metodi.



5.3.. attēls. Laiks, kurā tiek nosūtīta viena ziņa, izmantojot dažādas autorizācijas metodes.

Līdzīgi kā bez šifrēšanas izmantojot sertifikātus, tika novērots, ka paroļu faila lietošana sniedz lielāku veiktspēju, salīdzinot ar autentifikācijas moduli. Procentuāli autentifikācijas moduļa ietekme uz nosūtīto ziņu skaitu samazinājās par 15.79%, bet laika atšķirība starp abiem testiem ir tikai 3 sekundes.



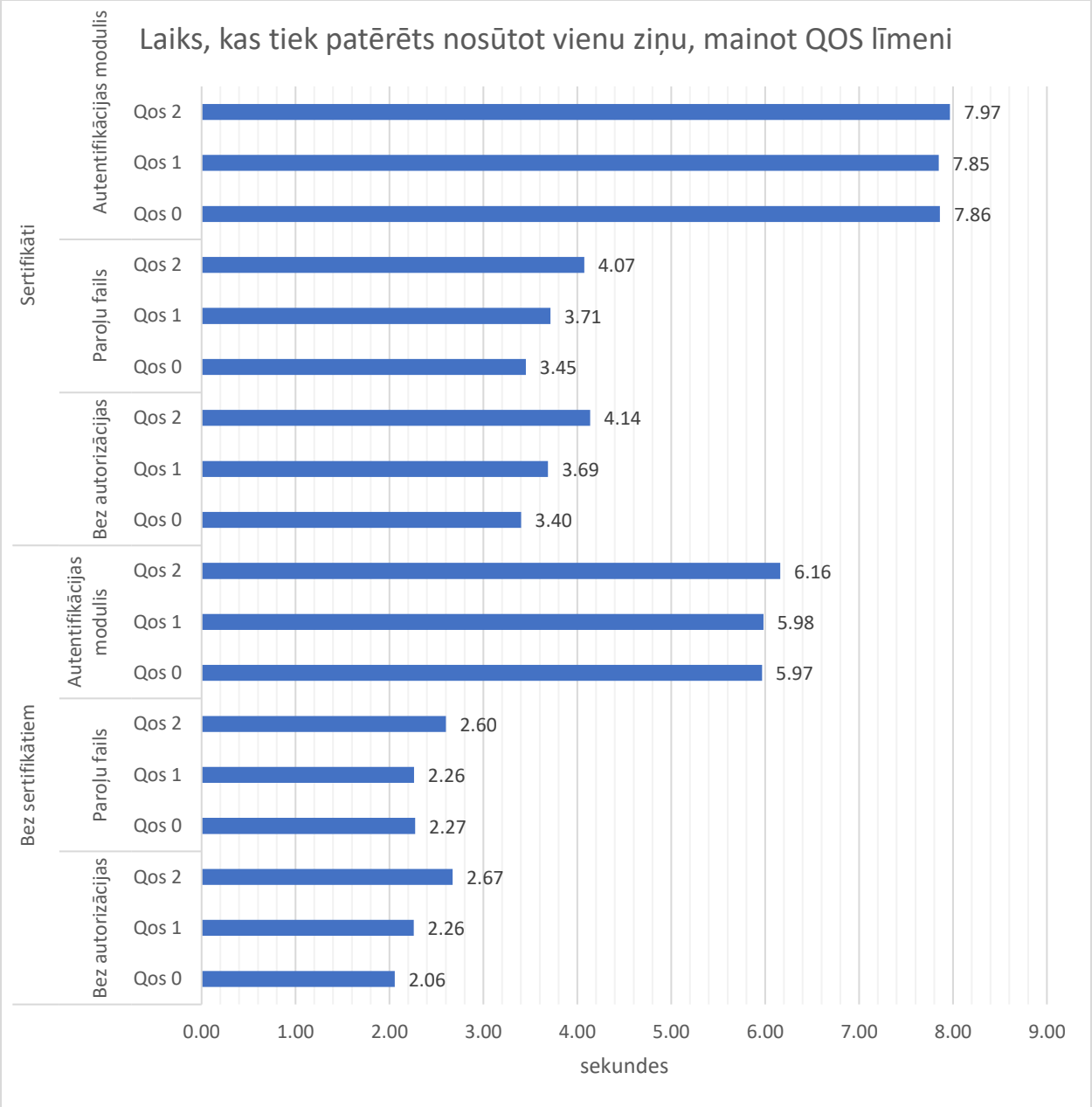
5.4. attēls. Laiks kurā tiek izpildīts viens testu raunds.

Kā ir redzams attēlā, salīdzinoši ar autentifikācijas moduli, paroļu faila izmantošana gandrīz vispār neietekmē veiktspēju.

Priekš autorizācijas netika veikti testi izmantojot PSK autentifikācijas metodi, jo Paho bibliotēka to neatbalstīja. Lai gan būtu informatīvi iegūt rezultātus par visiem iespējamajiem veidiem kā veidot autorizāciju un autentifikāciju, autors, PSK zemākās drošības un atbalsta trūkuma dēļ, neveica autorizācijas testus ar PSK autentifikāciju.

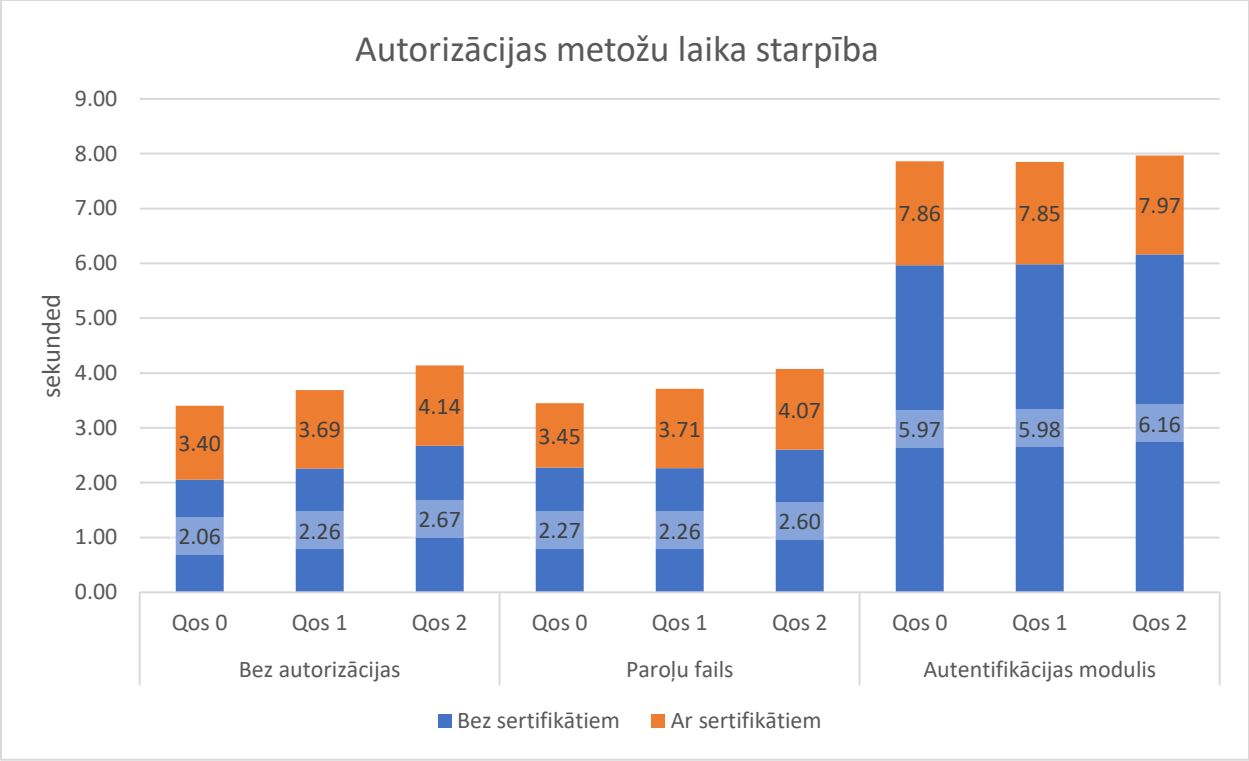
5.3. SERVISA KVALITĀTE

Svarīgi ir arī notestēt kā QoS ietekmē izpildes laiku. Zinot ietekmi, ir iespējams izvērtēt vai pārliecība, ka ziņa ir pienākusi, ir pietiekami svarīga, lai samazinātu izsūtīto ziņu daudzumu. QoS testi tika veikti katrai autentifikācijas un autorizācijas variācijai, tas ir 12 papildus testu raundi, iepriekšējiem tie bija 6.



5.5. attēls. Laiks, kas tiek patērēts nosūtot vienu ziņu, mainot QoS līmeni.

Ar un bez sertifikātiem netika novērota liela laika starpības izmaiņa starp abu tā paša autorizācijas metodes izvēles.



11. attēls – Tabula parāda laika atšķirību starp autentifikācijas metodēm dažādās autorizācijas metodēs un to QOS līmeņos.

Tabulas datos ir iespējams ieraudzīt, ka palielinot QOS līmenim, tiks samazināta veiktspēja. Piemēram, neizmantojot sertifikātus un autorizāciju, izvēloties QoS 1. līmeni, salīdzinoši ar QoS 0. līmeni, nosūtīto ziņu ātrums samazinājās par 8.96%, toties palielinot QoS uz 2. līmeni, nosūtīto ziņu ātrums samazinājās par 29.09%.

Ieviešot sertifikātus, bez autorizācijas un izvēloties QoS 1. līmeni, salīdzinoši ar QoS 0. līmeni, nosūtīto ziņu daudzums samazinās par 7.88% un palielinot uz 2. līmeni, nosūtīto ziņu daudzums samazinās par 17.77%.

Toties, ja tiek izmantota gan autorizācija, gan autentifikācija, gan QoS 1. līmenis, gan 2. līmenis ietekmē nosūtīto ziņu daudzumu līdzīgi – 3.92%, 3.82% attiecīgi. Pēc tā var noprast ka iestājas kāda veida limits, dēļ kura vairs netiek ietekmēta ātrdarbība, kas ir atkarīga no QOS. Protams, jāņem vērā, ka visi testi tika veikti bez ziņu zudumiem un, ja tiktu ieviesti ziņu zudumi, rezultāti būtu daudz savādāki.

SECINĀJUMI

Balstoties uz praktiskajā un teorētiskajā daļā iegūtajiem rezultātiem, darba autors spēj veikt atbilstošus secinājumus:

- Katrs lietu interneta protokols ir paredzēts noteiktai tīkla topoloģijai, tāpēc ir jāizvēlas atbilstošākais protokols;
- Ziņas nosūtīšanas ātrumu var ļoti ietekmēt izmantotā bibliotēkas metode, pat izmantojot tos pašus parametrus;
- Datu bāzes izmantošana priekš autorizācijas rada lielu veiktspējas trūkumu;
- Sertifikātu izmantošana priekš autentifikācijas ir drošāks veids, salīdzinoši ar PSK, kā panākt drošu saziņu starp galapunktiem, jo tiek izmantotas garākas atslēgas priekš šifrēšanas, un tā priekšrocības atsver tā ātrdarbības ietekmi;
- Autorizācijas modulis, kas iekļauj sevī datubāzes atbalstu, ir labs risinājums, ja ir nepieciešams nodrošināt vieglu autorizēšanās implementēšanu jau esošā lietotāju reģistrā;
- Ja testa gultnei nav jānodrošina nepārtraukta jaunu lietotāju pievienošana vai nav nepieciešama testa gultnes konstanta darbība, izmantot paroļu failu risinājumu ir vilinošs risinājums, mazās ātrdarbības ietekmes dēļ;
- Autors priekš testa gultnes autentifikācijas un autorizācijas ieteiktu izmantot ar sertifikātiem bāzētu autentifikāciju un autorizācijas moduli, kas iekļauj sevī datubāzes atbalstu;

Autoram neizdevās pilnīgi sasniegt izvirzīto darba mērķi, jo praktiski netika iegūti visi nepieciešamie PSK autentifikācijas metodes testu rezultāti, kas radās dēļ neatbilstošas MQTT protokola bibliotēkas izvēles.

Lai būtu iespējams precīzāk notestēt, kāda ietekme būs testētajām metodēm uz ātrdarbību, autors iesaka notestēt visas nepieciešamās metodes uz reālajām testa gultnē izmantotajām iekārtām.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Ronalds B. F. 2016. Father of the Electric Telegraph. - Grām. Londona: Imperial College Press. 620pp.
2. Judvaitis J., Salmins A., Nesenbergs K. 2016. Network Data Traffic Management Inside a TestBed. - Advances in Wireless and Optical Communications (RTUWO). 152-155pp.
3. Light R. A. 2017. Mosquitto: server and client implementation of the MQTT protocol. - Journal of Open Source Software, 2(13):265.
4. Sengul C., Kirby A., Fremantle P. 2020. MQTT-TLS profile of ACE draft-ietf-ace-mqtt-tls-profile-04. - Internet Draft. 28pp.
5. Shelby Z., Hartke K., Bormann C. 2014. The Constrained Application Protocol (CoAP). Internet Engineering Task Force (IETF). 112pp.
6. A General-Purpose Middleware Standard. 2006. AMQP Advanced Message Queuing Protocol. - Advanced Message Queuing Protocol Specification. 65pp.
7. Anonymous 2019. Latvian research institute offers wireless sensor network and internet of things testbed for research and commercial use. ENTERPRISE EUROPE NETWORK. https://een.ec.europa.eu/partners/latvian-research-institute-offers-wireless-sensor-network-and-internet-things-testbed.

PIELIKUMI

PIELIKUMS 1. Mosquitto konfigurācijas fails

```
      # auth_plug# auth_opt_host localhost

      # auth_opt_port 5432

      # auth_opt_dbname testbed

      # auth_opt_user testbed

      # auth_opt_pass testbed

      # auth_opt_userquery SELECT pw FROM account WHERE username = $1 limit 1

      # auth_opt_superquery SELECT COALESCE(COUNT(*),0) FROM account WHERE usern
ame = $1 AND super = 1

      # auth_opt_aclquery SELECT topic FROM acls WHERE (username = $1) AND (rw &
 $2) > 0

      port 8883

      # Pieslēgt ssl sertifikātus

      # cafile ./backend/my-ca.crt

      # certfile ./backend/server.crt

      # keyfile ./backend/server.key

      # require_certificate true


      # Paroļu fails

      # password_file ./backend/passwfile


      # PSK autentifikācijas modulis

      # pre-shared-key security

      # psk_file ./backend/pskfile

      # psk_hint hint

      # tls_version tlsv1.2

      # use_identity_as_username true

/backend/auth-placl (piekļuveszācijas modulis
```

PIELIKUMS 2. Aclfile - (piekļuves kontroles saraksts) faili

```
# This affects access control for clients with no username.
topic read $SYS/#

# This only affects clients with username "bob".
user bob
topic foo/bar
```

PIELIKUMS 3. passw - paroļu fails

```
bob:$6$MXX/XQ3aS/KSgDKd$70wmyfFPwwkHqKuXL87PJ4P8/mW0+RmFYnbKf223U9g6t+qA36
Ocb23mea8cmNlhQFJJ5fLOniSs6TUCUs8p+w==
```

PIELIKUMS 4. pskfile - fails, priekš Pre-shared key izmantošanas

```
bob:7465737475736572
```

PIELIKUMS 5. subscriber.py - fails, abonētāja programma

```
#!/usr/bin/env python
import sys
#import context  # Ensures paho is in PYTHONPATH

import paho.mqtt.client as mqtt
import json # config is loaded from config
import argparse # Needed to parse arguments given in the command line

import ssl # enable the user of public private keys


parser = argparse.ArgumentParser()

parser.add_argument('-u', '--username', required=True, default=None)
parser.add_argument('-p', '--password', required=True, default=None)
parser.add_argument('-t', '--topic', required=True, default="$SYS/#")
parser.add_argument('-Q', '--qos', required=False, default=0)
parser.add_argument('--tls-
version', required=False, default=None, help='TLS protocol version, can be one of
 tlsv1.2 tlsv1.1 or tlsv1\n')
parser.add_argument('-auth', '--
auth', required=True, default=False) # to go around auth
parser.add_argument('-certs', '--
certs', required=True, default=False) # to go around certs


args, unknown = parser.parse_known_args()

with open('conf.json') as json_data_file:
    conf = json.load(json_data_file)

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    if userdata == True:
        print("rc: " + str(rc))

def on_log(mqttc, userdata, level, string):
    print(string)



# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

def main(argv):
```

```
    print("does stuff")
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.on_connect = on_connect
    print("args.noauth", args.auth)

    if args.tls_version == "tlsv1.2":
       tlsVersion = ssl.PROTOCOL_TLSv1_2
    elif args.tls_version == "tlsv1.1":
       tlsVersion = ssl.PROTOCOL_TLSv1_1
    elif args.tls_version == "tlsv1":
       tlsVersion = ssl.PROTOCOL_TLSv1
    elif args.tls_version is None:
       tlsVersion = None
    else:
       print ("Unknown TLS version - ignoring")
       tlsVersion = None


    print("Certs:" + args.certs)
    if str(args.certs) == "True":
        print("##### Setting certs!")
        client.tls_set(ca_certs="backend/my-
ca.crt", certfile="client.crt", keyfile="client.key", cert_reqs=ssl.CERT_REQUIRED
, tls_version=ssl.PROTOCOL_TLSv1_2)

    print("Auth:" + args.auth)
    if str(args.auth) == "True":
        print("##### Setting certs!")
        print(args.username, args.password)
        client.username_pw_set(args.username, args.password)
    print("port", conf["base"]["port"])


    client.connect(conf["base"]["host"], conf["base"]["port"], 60)

    print("Connected")

    client.subscribe(args.topic, int(args.qos))
    # Blocking call that processes network traffic, dispatches callbacks and
    # handles reconnecting.
    # Other loop*() functions are available that give a threaded interface and a
    # manual interface.
    client.loop_forever()


if __name__ == "__main__":
    main(sys.argv[1:])
```

PIELIKUMS 6. usergenerate.py - fails, kas izveido priekš klientiem sertifikātus un lietotājvārdus/paroles/tiesības datubāzē

```
#!/usr/bin/env python
import sys
import argparse
import hashing_passwords as hp
import json # config is loaded from config
import psycopg2 # connection to postgreSQL db

# Certification building
import datetime
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa


parser = argparse.ArgumentParser()
# Print help
parser.add_argument('-H','--HELP', action='store_true')
# Generate mote db username/passw and certificates
parser.add_argument('-MU', '--moteuser', required=False, default=None)
parser.add_argument('-MP', '--motepassw', required=False, default=None)
parser.add_argument('-GC', '--gencert', action='store_true')
parser.add_argument('-FC', '--fromconfig', required=False, default=None)

args, unknown = parser.parse_known_args()


if args.fromconfig:
    with open(args.fromconfig) as json_data_file:
        conf = json.load(json_data_file)




def main(argv):

    if args.HELP:
        print("help is provided")

    if args.fromconfig:

        private_key = open("../backend/my-ca.key", "r").read()
        cert_authority = open("../backend/my-ca.crt", "r").read()

        root_key = serialization.load_pem_private_key(
            private_key.encode("ascii"), password=b"testbed", backend=default_bac
kend()
```

```
        )

        root_cert = x509.load_pem_x509_certificate(
            cert_authority.encode("ascii"), default_backend()
        )

        print(root_cert)
        print(root_key)

        hashpw = hp.make_hash(conf["mote"]["password"])

        #insert mote info in db
        insert_mote(conf["mote"]["name"], hashpw, conf["mote"]["super"])
        insert_mote_rights(conf["mote"]["name"], conf["mote"]["rights"])

    if args.gencert:
        cert_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=2048,
            backend=default_backend())

        new_subject = x509.Name([
            x509.NameAttribute(NameOID.COUNTRY_NAME, conf["cert"]["country"]),
            x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, conf["cert"]["stat
e"]),
            x509.NameAttribute(NameOID.LOCALITY_NAME, conf["cert"]["city"]),
            x509.NameAttribute(NameOID.ORGANIZATION_NAME, conf["cert"]["organizat
ion"])
        ])

        cert = x509.CertificateBuilder().subject_name(
            new_subject
        ).issuer_name(
            root_cert.issuer
        ).public_key(
            cert_key.public_key()
        ).serial_number(
            x509.random_serial_number()
        ).not_valid_before(
            datetime.datetime.utcnow()
        ).not_valid_after(
        datetime.datetime.utcnow() + datetime.timedelta(days=30)
        ).add_extension(
            x509.SubjectAlternativeName([x509.DNSName(conf["cert"]["cn"])]),
            critical=False,
        ).sign(root_key, hashes.SHA256(), default_backend())
       # print(new_subject)

        with open("phone_cert.crt", "wb") as f:
            f.write(cert.public_bytes(encoding=serialization.Encoding.PEM))

        with open("phone_cert.key", "wb") as f:
```

```
            f.write(cert_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption(),
        ))


def insert_mote(name, pasw, issuper):

    conn = psycopg2.connect(host=conf["db"]["host"],database=conf["db"]["database
"], user=conf["db"]["user"], password=conf["db"]["password"])
    cur = conn.cursor()

    sql = "INSERT INTO account(username, pw, super) values (%s,%s, %s)"
    cur.execute(sql, (name, pasw, issuper))
    conn.commit()
    conn.close()


def insert_mote_rights(name, mote_list):

    conn = psycopg2.connect(host=conf["db"]["host"],database=conf["db"]["database
"], user=conf["db"]["user"], password=conf["db"]["password"])
    cur = conn.cursor()

    sql = "INSERT INTO acls(username, topic, rw) VALUES (%s,%s,%s)"
    for key in mote_list:
        print("key val", key, mote_list[key])
        cur.execute(sql, (name, key, mote_list[key])) # creates user rights
        conn.commit()

    cur.close()

if __name__ == "__main__":
    main(sys.argv[1:])
```

PIELIKUMS 7. tests.py – fails, kurā tika veikti visi darbā veiktie testi

```
from subprocess import call
import string
import random

from datetime import datetime
import _thread
import time
import ssl
import paho.mqtt.publish as publish

import os
import psutil


lock = _thread.allocate_lock()
num_threads = 0
thread_started = False

# Inputs for testing
QOS_LEVEL = 0
CERTS = False
AUTH = False

MESSAGE_COUNT = 100
TEST_COUNT = 100
NODE_COUNT = 100
PAYLOAD_SIZE = 46 #
PAYLOAD_DELAY = 0.01 # sec



def randomMessage():
    letters = string.ascii_lowercase
    payload = ''.join(random.choice(letters) for i in range(PAYLOAD_SIZE))
    return payload
payload = []
for x in range(10):
   payload.append(randomMessage())


def main():

    def publishMessage(topic, payload, qos, username, password):
        tlsVersion = ssl.PROTOCOL_TLSv1_1
        hostname = "127.0.0.1"
        port = 8883
        if AUTH is True:
            auth = {
                "username":username,
                "password":password
```

```
                }
        else:
            auth = None

        if CERTS is True:
            tls = {
                "ca_certs":"backend/my-ca.crt",
                "certfile":"client.crt",
                "keyfile":"client.key",
                "tls_version":ssl.PROTOCOL_TLSv1_2,
                "ciphers":None
                }
        else:
            tls = None



        publish.single(topic = topic, payload=payload, qos=qos, hostname=hostname
,port=port, keepalive=60, auth=auth, tls=tls,protocol= tlsVersion, transport="tcp
")

    def sendMessage(threadName):
        global num_threads, thread_started
        lock.acquire()
        num_threads += 1
        thread_started = True
        lock.release()

        for x in range(MESSAGE_COUNT):
            message = str(random.choice(payload))

            publishMessage("foo/bar",message, QOS_LEVEL, "bob","bob")
            time.sleep(PAYLOAD_DELAY)

        lock.acquire()
        num_threads -= 1
        lock.release()
        # return new

    def Average(lst):
        return sum(lst) / len(lst)

    #launch mosquitto
    call([  'gnome-terminal', '-x', "mosquitto", "-
c","backend/mqttdb.conf"]) # launch broker

    #set subscriber
    call([  'gnome-terminal', '-x', "python3", "user.py", "-u", "bob", "-
p", "bob", "-t", "foo/bar", "-Q", str(QOS_LEVEL), "-auth", str(AUTH), "-
certs",str(CERTS)])

    #publish messages
```

```
    print("Payload: " + str(PAYLOAD_SIZE))
    result_list = []
    global thread_started
    for y in range(TEST_COUNT): # Number of test runs

        start = time.time() # start execution timer

        for x in range(NODE_COUNT): # amound of nodes to create
            _thread.start_new_thread(sendMessage,("Thread-"+str(x),))
            # call([   "mosquitto_pub","-p","8883","-u","peter","-t","foo/bar","-
m", "peter sends love "+ datetime.now().strftime("%H:%M:%S.%f")])

        # loop while threads execute
        while not thread_started:
            pass
        while num_threads > 0:
            # print(num_threads)
            pass

        done = time.time() # end execution timer

        elapsed = done - start # calculate time

        print("Rounded result: " + str(round(elapsed,5)) + " Test number: " + str
(y)) # print results

        result_list.append(elapsed) # add result to list of results

        thread_started = False


    print("Median result: " + str(Average(result_list))) # calculate average time
 of test
    print("Total time: " + str(sum(result_list)))

main()
```

Bakalaura darbs „Autorizācija un autentifikācija testa vides aizmugursistēmai” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Jēkabs Jānis Kaniņš

Rekomendēju/nerekomendēju darbu aizstāvēšanai *(nederīgo svītro vadītājs)*

Vadītājs: pētnieks M.dat. Jānis Judvaitis _____ __.06.2020.

Recenzents: lektora p.i. Dr. dat. Bogdans Ozerkins

Darbs iesniegts Datorikas fakultātē 25.05.2020.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2020. prot. Nr. _____

Komisijas sekretārs(-e): _____