TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Igor Andrejev 232077IACM

KNX IoT raamistikul põhineva testseadme arendamine ja sidumine KNX-võrkudega

Magistritöö

Juhendaja: Andres Rähni

Tehnikateaduste-
magister

Tallinn 2025

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Igor Andrejev

12.05.2025

Annotatsioon

Käesolev magistritöö keskendub KNX IoT spetsifikatsiooni praktilisele rakendamisele ja selle integreerimisele klassikalise KNX hooneautomaatikavõrguga. Töö eesmärk oli välja töötada KNX IoT raamistikul põhinev testseade ning hinnata selle funktsionaalsust, töökindlust ja sobivust reaalsetes tingimustes. Töö käigus analüüsiti KNX ja KNX IoT standardite tehnilisi omadusi ning võrreldi neid teiste levinud protokollidega.

Praktilises osas loodi testkeskkond, kus kasutati ESP32 mikrokontrollerit ja CoAP protokolli, võimaldades andurite ja täiturite juhtimist vastavalt KNX IoT nõuetele. Kommunikatsioon KNX-võrguga toimus läbi KNX/IP-ruuteri ning süsteemiga saavutati kahepoolne andmevahetus. Andmete kodeerimiseks kasutati CBOR formaati ja lisaks implementeeriti OSCORE turvamehhanism. Süsteemi juhtimiseks ja visualiseerimiseks kasutati Node-RED platvormi, mis võimaldas ka lihtsat kasutajaliidest veebipõhise *dashboard*'i näol.

Töö sisaldab ka ülevaadet ametlikest KNX IoT komponentidest, nagu Point API ja 3rd Party API Server. Kuigi nende täielik rakendamine ei olnud töö fookuses, toodi välja nende potentsiaal professionaalsetes projektides. Kokkuvõttes näitab töö, et KNX IoT kontseptsiooni saab rakendada ka piiratud ressursiga seadmetel ning sellist lähenemist saab edukalt kasutada arenduses, testimisel ja tulevaste hooneautomaatikalahenduste prototüüpimisel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 46 leheküljel, 11 peatükki, 20 joonist, 5 tabelit.

Abstract

Development of a test device based on the KNX IoT framework and its integration with KNX networks

This master's thesis focuses on the practical implementation of the KNX IoT specification and its integration with a traditional KNX building automation system. The primary objective of the work was to develop a test device based on the KNX IoT framework. In addition to theoretical analysis, the thesis also presents a working prototype, tested in a real KNX infrastructure.

The first part of the thesis provides an overview of the KNX and KNX IoT standards, their architecture, data transmission protocols, and security mechanisms. It also includes a comparative analysis of alternative automation solutions. Special attention is given to the challenges and benefits of integrating KNX with modern IP-based and cloud-connected IoT systems.

In the practical part of the work, a test device was implemented using the ESP32 microcontroller, selected during the project for its flexibility, connectivity, and support for lightweight protocols. A CoAP server was developed on the device, supporting key elements of the KNX IoT Point API specification. Communication with a real KNX network was established through a KNX/IP router, enabling reliable two-way data exchange. Data was encoded using the CBOR format, and secure transmission was demonstrated by implementing OSCORE for one of the CoAP resources.

For system integration and visualization, the Node-RED platform was used, offering both protocol bridging and a browser-based dashboard interface. The system allowed real-time monitoring and control of sensors and actuators, such as LED lights, dimmers, and environmental data.

The thesis also explores official KNX IoT components such as the KNX IoT Point API and the 3rd Party API Server. Although full-scale implementation of these components

was beyond the project scope, the thesis highlights their architectural potential and relevance for scalable and professional automation applications.

In conclusion, the work demonstrates that even resource-constrained platforms can effectively implement key elements of the KNX IoT specification. The developed prototype provides a flexible foundation for further experimentation, educational use, and the development of more advanced smart building solutions. Future work could expand the platform's functionality or integrate more standardized KNX IoT components.

The thesis is in Estonian and contains 46 pages of text, 11 chapters, 20 figures, 5 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
API Server / API Client	Rakendusliidese server / klient
CBOR	*Concise Binary Object Representation*
CoAP	*Constrained Application Protocol*
ESP32	Espressif ESP32, mikrokontrollerite seeria
ESP-IDF	*Espressif IoT Development Framework*
ETS 6	Engineering Tool Software ver 6
FreeRTOS	*Free Real-Time Operating System*
GPIO	*General Purpose Input/Output*, üldotstarbeline sisend-väljund
HTTP	*HyperText Transfer Protocol*, hüperteksti edastuse protokoll
HVAC	*Heating, Ventilation & Air Conditioning*, KVJ – Küte, vent., jahutus
IP	*Internet Protocol*
IPv6 / IPv4	*Internet Protocol version 6 / 4*
IoT	*Internet of Things*, Asjade internet
JSON	*JavaScript Object Notation*, JavaScripti objektide notatsioon
KNX	*Konnex*, hooneautomaatika standard tehnoloogia
LED	*Light Emitting Diode*, valgusdiood
mesh	*mesh network*, silmusvõrk
MQTT	*Message Queuing Telemetry Transport*, sõnumijärjekorraga telemeetriatransport
Node-RED	*Flow-based Programming Tool*
OSCORE	*Object Security for Constrained RESTful Environments*
PL	*Powerline*, toitevõrgupõhine andmeside
Point API	*KNX IoT Point API*
REST	*Representational State Transfer*, esitusoleku siire
RF	*Radio Frequency*, raadiosagedus
RTT	*Round Trip Time*, pendellevi aeg

TCP	*Transmission Control Protocol*, edastusohje protokoll
Thread	IEEE 802.15.4 põhine silmusvõrgu protokollistik
TP	*Twisted Pair*, keerupaar ühendus
UDP	*User Datagram Protocol*, kasutajadatagrammi protokoll
XML	*Extensible Markup Language*, laiendatav märgistuskeel
3rd Party API	*KNX IoT 3rd Party API*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Viimastel aastatel on hoonete automatiseerimise kontseptsioon märkimisväärselt muutunud, liikudes suletud süsteemidelt integreeritud lahendustele, mis ühendavad erinevaid protokolle ja tehnoloogiaid. Üks levinumaid rahvusvahelisi standardeid selles valdkonnas on KNX, mida kasutatakse valgustuse, kütte, ventilatsiooni, kliimaseadmete ja muude hooneautomaatika süsteemide juhtimiseks.

Kaasaegsed infotehnoloogia ja asjade interneti arengusuunad esitavad hoonete automatiseerimissüsteemidele uusi nõudeid. On tekkinud vajadus integreerida hoone automatiseerimise nutiseadmed ühtsesse võrguinfrastruktuuri, kasutades standardseid IP (*Internet Protocol*) - tehnoloogiaid, pilvelahendusi ja võrgu turvameetmeid. Selle ülesande lahendamiseks töötati välja KNX IoT spetsifikatsioon, mis võimaldab integreerida klassikalisi KNX-seadmeid veebipõhiste ja pilvelahendustega, kasutades IPv6 ja avatud andmeedastusprotokolle, näiteks CoAP (*Constrained Application Protocol*) ja CBOR (*Concise Binary Object Representation*).

Siiski on KNX IoT juurutamine alles algusjärgus. Paljud praktilise rakendamise ja IoT-seadmete olemasolevatesse KNX-võrkudesse integreerimisega seotud küsimused vajavad täiendavat uurimist. Seetõttu on oluline luua testseadmeid ja katsestende, mis võimaldavad uue spetsifikatsiooni funktsionaalsust ja ühilduvust praktikas kontrollida.

Käesoleva magistritöö eesmärk on välja töötada testseade ESP32 platvormil, mis toetab CoAP protokolli ja on kooskõlas KNX IoT spetsifikatsiooniga. Töö raames on kavas integreerida see seade reaalsesse KNX-infrastruktuuri, kasutades olemasolevat riistvara (nt KNX IP Router, siiniseadmed ja juhtimiskomponendid). Samuti viiakse läbi traditsioonilise KNX-keskkonna ja veebiliideste vahelise suhtluse seadistamine ja testimine, võimaldades seadmete jälgimist ja juhtimist nii kohapeal kui ka kaugühenduse kaudu internetis.

Töö tähtsus seisneb erinevate hooneautomaatika tehnoloogiate põlvkondade integreerimismehhanismide uurimises, tuginedes KNX-i ühisele andmemudelile. Saadud

tulemused võimaldavad hinnata uue KNX IoT spetsifikatsiooni rakendatavust ja tõhusust, tuvastada selle eelised ja piirangud ning määrata võimalikud kasutusvõimalused.

Töös on seatud järgmised ülesanded:

- Teha ülevaade KNX ja KNX IoT standardite hetkeseisust, määrata nende eelised ja piirangud võrreldes teiste automatiseerimislahendustega.
- Arendada praktiline katsestend, kasutades ESP32-e, CoAP protokolli ning teisi KNX IoT-ga integreerimiseks vajalikke tehnoloogiaid.
- Seadistada ja integreerida testseade olemasolevasse KNX-infrastruktuuri, kasutades ülikooli praktikumiklassis olemasolevat KNX IP-ruuterit ja tüüpilist KNX-riistvara.
- Hinnata välja töötatud lahenduse funktsionaalsust, töökindlust ja jõudlust.

Seega on töö suunatud praktiliste teadmiste omandamisele klassikalise KNX-keskkonna ja kaasaegsete IoT-tehnoloogiate integreerimisel ning katsenäidiste väljatöötamisele selliste lahenduste rakendamisel.

Töö järgnevad peatükid annavad süsteemse ülevaate uurimisteemaga seotud tehnoloogiatest ja praktilistest lahendustest. Teises peatükis käsitletakse KNX standardi ajalugu, arhitektuuri, andmeedastusmeetodeid ja kasutusvaldkondi. Kolmandas peatükis analüüsitakse KNX-i eeliseid ja puudusi ning võrreldakse seda populaarsete alternatiivsete hooneautomaatikalahendustega. Neljandas peatükis tutvustatakse KNX IoT kontseptsiooni, selle arhitektuurilisi lahendusi, kasutatavaid protokolle (CoAP, CBOR) ning turvameetmeid. Viiendas peatükis käsitletakse tarkvarapõhiseid simuleerimiskeskkondi – KNX Virtual, ETS6 ja KNX IoT Virtual, mis võimaldavad testida KNX süsteeme ilma füüsilise riistvarata. Kuuendas peatükis keskendutakse CoAP-serveri arendamisele ESP32 mikrokontrolleril ning selle ettevalmistamisele KNX IoT süsteemiga integreerimiseks. Seitsmendas peatükis käsitletakse Node-RED platvormi kasutamist, sealhulgas CoAP ja KNX seadmete seadistamist ning kahepoolse suhtluse realiseerimist. Kaheksandas peatükis antakse ülevaade arendatud testseadmetest, nende funktsionaalsusest, sensorite ja täiturite tööst ning praktilistest rakendustest. Üheksandas peatükis analüüsitakse süsteemi jõudlust ja töökindlust testide kaudu. Kümnes peatükk keskendub KNX IoT Point API ja 3rd Party API serveri

arhitektuurilisele ja praktilisele analüüsile. Üheteistkümnendas peatükis esitatakse töö kokkuvõte ja järeldused.

2 KNX

KNX (Konnex) on avatud ja rahvusvaheline hooneautomaatika standard, mida kasutatakse laialdaselt kaasaegsete hoonete tehnosüsteemide automatiseerimiseks. See standard ühendab erinevaid kommunikatsiooni- ja juhtimislahendusi ühtsesse süsteemi, mis võimaldab tagada hoonete tõhusat juhtimist ja kõrget kasutusmugavust [1].

Antud peatükis käsitletakse KNX standardi ajalugu, tehnilisi omadusi ning selle kasutamise võimalusi ja piiranguid. Samuti kirjeldatakse KNX-i võrguarhitektuuri ja andmeedastusmeetodeid ning tutvustatakse süsteemi haldamiseks mõeldud ETS (Engineering Tool Software) - tarkvara.

2.1 KNX standardi põhikontseptsioon ja ajalugu

KNX on rahvusvaheline standard, mis on loodud hoonete erinevate insenerisüsteemide – näiteks valgustuse, kütte, ventilatsiooni, kliimaseadmete, turvasüsteemide ja energiatõhususe – automaatseks juhtimiseks ja juhtimise lihtsustamiseks. KNX-i kasutatakse laialdaselt nii elumajades ja korterites kui ka suurtes kommertsobjektides – näiteks kontorihoonetes, haiglates, hotellides ja kaubanduskeskustes.

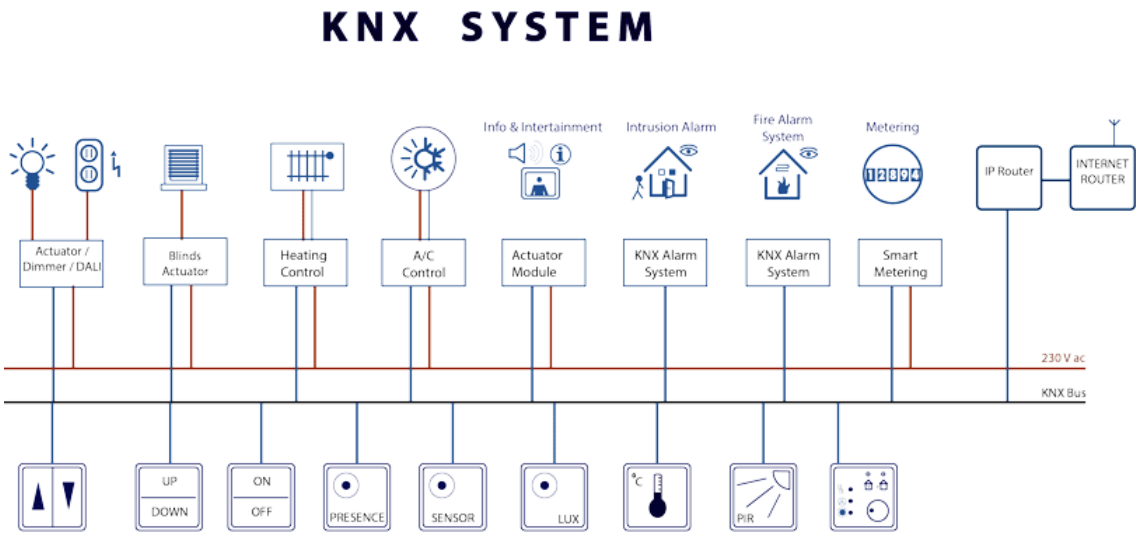
Standard tekkis 2000. aastate alguses kolme varasema hooneautomaatika tehnoloogia – EIB (European Installation Bus), BatiBUS ja EHS (European Home Systems) – ühendamise tulemusena, mille eesmärgiks oli luua ühtne ja standardiseeritud lahendus erinevate insenerisüsteemide automatiseerimiseks [2].

2.2 KNX võrguarhitektuur ja tööpõhimõtted

KNX-süsteem põhineb tüüpiliselt siinipõhimõttel (*bus*), kus kõik seadmed on ühendatud ühisele kommunikatsioonisiinile või juhtmevaba võrgu segmenti ja vahetavad infot spetsiaalsete sõnumite ehk telegrammide kaudu. Selline lahendus tagab ühtse suhtlusplatvormi, mille kaudu seadmed saavad andmeid edastada ja vastu võtta.

Üks KNX-i olulisemaid eeliseid on detsentraliseeritud võrguarhitektuur, mis tähendab, et süsteemil puudub kohustuslik keskne juhtseade. Iga seade suudab suhelda otse teiste seadmetega, mis tagab suurema töökindluse ja välistab ühe keskse punkti rikkest tingitud

täieliku võrgu toimimise katkemise (Joonis 1). See arhitektuur muudab süsteemi paindlikumaks, robustsemaks ja vähendab halduskulusid [3].



Joonis 1. KNX-i andmesideliini näide [4]

2.3 KNX süsteemi tehnilised piirangud ja võimalused

Viimase standardi järgi (TP1-256) saab ühele keerupaari liinile ühendada kuni 256 seadet, mis oluliselt laiendab võrreldes varasema 64 seadme piiranguga [5]. Liini maksimaalne pikkus võib ulatuda ligikaudu 1000 meetrini, mis annab süsteemile hea skaleeritavuse ka suuremates hoonetes [6].

Kui on vaja ühendada rohkem seadmeid või luua suuremaid süsteeme, kasutatakse liiniühendajaid (*line couplers*), piirkonnaühendajaid (*area couplers*) ja lüüse (*gateways*), mis võimaldavad ühendada mitu liini ühte suuremasse võrku. See arhitektuurne lahendus muudab KNX-süsteemi sobivaks nii väiksematesse kui ka suurtesse rajatistesse.

2.4 KNX andmeedastusmeetodid

KNX toetab mitut erinevat andmeedastusmeediat, mis muudab selle kasutatavaks erinevates olukordades ja keskkondades. Erinevate meetodite tundmine aitab valida sobivaima lahenduse konkreetsele projektile, lähtudes näiteks paigaldustingimustest, kuludest ja nõutud töökindlusest:

- KNX TP (Twisted Pair) – see lahendus, kus andmeedastus toimub keerupaari kaabli kaudu. Edastuskiirus on ligikaudu 9600 bitti sekundis. TP meetod on usaldusväärne ja suhteliselt immuunne elektromagnetiliste häirete suhtes.
- KNX RF (Radio Frequency) – juhtmevaba andmeedastus raadiokanali kaudu sagedusel 868 MHz. Seda kasutatakse olukordades, kus kaablite paigaldamine on keeruline või liiga kulukas. Edastuskiirus on kuni 16 kbit/s.
- KNX Powerline – võimaldab andmeedastust olemasolevate elektrijuhtmete kaudu. Seda meetodit kasutatakse vähem, sest elektrijuhtmed võivad tekitada häireid, mis vähendavad andmeedastuse kvaliteeti. Edastuskiirus on madalam, umbes 1200 bitti sekundis.
- KNX IP – andmeedastus Ethernet/IP-võrkude kaudu. See võimaldab ühendada kaugseadmeid ja integreeruda teiste võrkudega [7].

2.5 KNX haldus ja tarkvara ETS

KNX-süsteemi töökindluse ja efektiivsuse tagamiseks on oluline mitte ainult riistvara, vaid ka tarkvaraline pool. Selleks, et erinevate tootjate seadmed toimiksid ühtselt ja kooskõlas, kasutatakse spetsiaalset haldustarkvara.

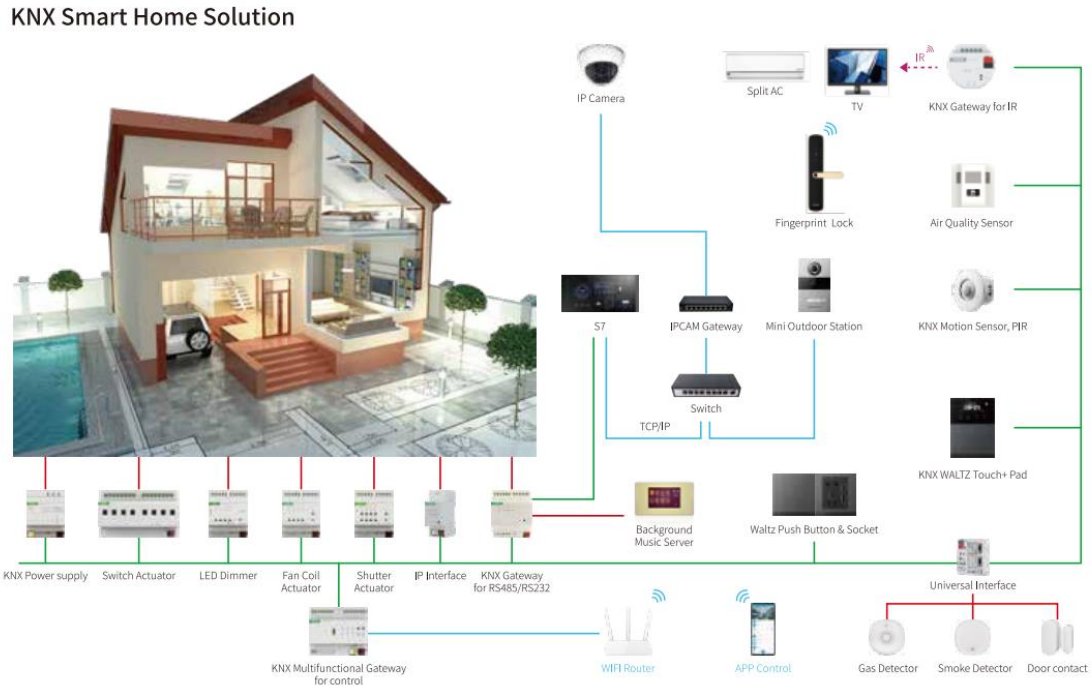
KNX süsteemide seadistamiseks, juhtimiseks ja hooldamiseks kasutatakse spetsiaalset tarkvara nimega ETS (Engineering Tool Software). See on kõikide tootjate ülene vahend, millesse kasutaja saab vajadusel tootja rakendusi (*Apps*) juurde lisada. ETS võimaldab seadmeid konfigureerida, grupiaadresse määrata, seadmete tööd jälgida ja vajadusel teha täiendavaid muudatusi. ETS tarkvara on disainitud nii, et see toetab eri tootjate seadmeid, tagades süsteemi paindlikkuse ja ühilduvuse erinevate komponentidega [8].

2.6 KNX rakendamise valdkonnad

Tihti KNX-i kasutatakse, et parandada hoonete mugavust, turvalisust ja energiatõhusust. KNX-süsteem võimaldab automatiseerida ja optimeerida erinevate hoonesüsteemide tööd vastavalt kasutajate vajadustele ja konkreetsele olukorrale.

Näiteks võimaldab KNX-süsteem automaatselt juhtida valgustust sõltuvalt ruumis viibimisest, reguleerida küttesüsteeme vastavalt ruumitemperatuurile, juhtida ventilatsioonisüsteeme vastavalt õhukvaliteedile ning integreeruda turvasüsteemidega,

tagades seeläbi hoone üldise turvalisuse ja kasutusmugavuse (Joonis 2). Lisaks pakub KNX energiatõhusaid lahendusi, mis aitavad vähendada energia- ja halduskulusid ning toetavad säästvat arengut [9].



Joonis 2. KNX targa maja süsteem [10]

3 KNX: eelised, puudused ja võrdlev analüüs

KNX on standard, millel on mitmeid eeliseid, aga ka mõningaid piiranguid võrreldes teiste sarnaste tehnoloogiatega. Käesolevas peatükis käsitletakse KNX-i eeliseid ja puuduseid ning võrreldakse neid teiste levinud hooneautomaatika protokollidega.

Hooneautomaatika protokollide võrdlemisel on oluline mõista, et iga protokoll töötab erinevatel kommunikatsioonikihtidel: alates füüsilisest kihist, mis määrab andmete ülekande keskkonna, kuni rakendus- ja loogiliste kihtideni, mis tegelevad andmevahetuse haldamise ja konkreetsete käskude töötlemisega. KNX ja teised sarnased protokollid hõlmavad sageli terviklikku protokollipinu, pakkudes seega laia funktsionaalsust ja terviklikku lahendust hooneautomaatika jaoks.

3.1 KNX eelised ja puudused

KNX on rahvusvaheliselt tunnustatud standard hooneautomaatika jaoks, mille suurim eelis on selle kõrge töökindlus. KNX-süsteemi töökindlus tuleneb peamiselt juhtmega ühendusest ja süsteemi disainist, kus iga sõnum kinnitatakse vastuvõtmisega. Lisaks on KNX detsentraliseeritud süsteem, mis ei nõua kohustuslikku keskset kontrollerit, võimaldades paindlikku laiendamist ja suurendades üldist töökindlust [11].

Veel üheks oluliseks KNX-i eeliseks on laialdane tootjate tugi, mis tagab seadmete hea ühilduvuse ja pikaajalise kasutuskindluse [12]. Samas on KNX-i puudusteks keerukus paigaldamisel ja programmeerimisel, mis eeldab spetsialiseeritud teadmisi ja ETS-tarkvara kasutamist. Samuti on KNX-seadmed ja tarkvara suhteliselt kallid, mistõttu võib see tehnoloogia olla väiksemate projektide või lõppkasutajate jaoks liiga kulukas. Lisaks võib KNX-i olla keeruline integreerida pilvepõhiste teenuste ja IoT-lahendustega, sest see standard loodi algselt suletud süsteemide jaoks [13].

3.2 Alternatiivsed hooneautomaatika protokollid

Lisaks KNX-standardile kasutatakse hooneautomaatikas ka mitmeid muid kommunikatsiooniprotokolle, nagu BACnet, LONWorks ja Matter, millel kõigil on oma tugevused ja nõrkused. Need tehnoloogiad iseloomustavad erinevaid süsteemseid lähenemisviise ja sobivad erinevate vajadustega projektidele. Üldiselt võib neid lahendusi

pidada terviklikeks süsteemideks, mis sisaldavad nii füüsilist kui ka rakenduslikku kommunikatsioonikihti.
BACnet, LONWorks ja Matter on alternatiivsed lahendused KNX-ile, millele järgnevalt antakse põhjalikum ülevaade, tuues välja nende olulisemad omadused ja erinevused võrreldes KNX-ga.

3.2.1 BACnet

BACnet (Building Automation and Control Networks) on standard, mida kasutatakse peamiselt suurtes hoonetes HVAC-süsteemide juhtimiseks ja jälgimiseks. Selle tugevusteks on hea skaleeritavus, paindlikkus erineva suuruse ja keerukusega süsteemides ning laialdane tugi paljude tootjate poolt, eriti Põhja-Ameerikas. Samuti võimaldab BACnet mitut tüüpi kohalike võrkude kasutust ja hõlpsat süsteemi laiendamist.
Samas on BACnetil ka piiranguid. Väiksemate lõppseadmete (nt valgustus või ventilaatorid) juhtimine ei pruugi olla efektiivne. Lisaks võib suurem arv seadmeid võrku üle koormata, mis eeldab keerukamat ja kallimat võrguhaldust. Kuigi standard toetab turvalisust, ei ole see funktsioon kõikides seadmetes täielikult rakendatud.

3.2.2 LonWorks

LonWorks on sarnaselt KNX-ile hajutatud juhtimissüsteem, mida kasutatakse laialdaselt nii hooneautomaatikas kui ka tööstuslikes rakendustes. Selle tugevusteks on seadmetasandil lihtsam arhitektuur ja võimalus kasutada veebipõhiseid tööriistu, mis säästavad aega ja kulusid. Lisaks on turul endiselt mitmeid arendajaid, kes pakuvad LON-põhiseid tooteid.
Samas on LonWorks paljuski vananev tehnoloogia, mille arhitektuur sõltub spetsiaalsest riistvarast (nt Neuron chip). Võrgu töökindlus võib kannatada, kui kontroller või ühendus katkeb. Süsteemi laiendamine toimub ainult läbi LonMark organisatsiooni, mis piirab paindlikkust [14].

3.2.3 Matter

Matter on avatud lähtekoodiga sideprotokoll, mida arendab Connectivity Standards Alliance (CSA) koostöös selliste tehnoloogiaettevõtetega nagu Apple, Google ja Amazon. Selle eesmärk on tagada erinevate tootjate seadmete vaheline sujuv koostöö ja kasutajamugavus nutikodudes.

Matter põhineb IP-tehnoloogiatel (Ethernet, Wi-Fi ja Thread), mis võimaldavad seadmete omavahelist suhtlust ilma pilveteenuste vahenduseta. See tagab madala latentsuse, hea töökindluse ja kaasaegse turvalisuse.

Matteri peamiseks eeliseks on hea ühilduvus populaarsete nutikodu platvormidega nagu Google Home, Apple HomeKit ja Amazon Alexa. Samas on tegemist suhteliselt uue tehnoloogiaga, mille seadmete levik ja praktiline rakendus on alles kujunemisjärgus, mistõttu selle töökindlus ja pikaajaline stabiilsus pole veel täielikult kinnitust leidnud [15].

3.3 Võrgu- ja sidetehnoloogiad hooneautomaatikas (alamtasemed)

Võrgu- ja sidetehnoloogiad määravad, kuidas seadmed omavahel suhtlevad füüsilisel ja võrgutasemel. Need tehnoloogiad on olulised seadmete vahelise kommunikatsiooni tagamiseks ning mõjutavad otseselt süsteemi töökindlust, energiatarvet ja paindlikkust. Järgnevalt käsitletakse populaarseid sidetehnoloogiaid: Zigbee, Z-Wave, Thread, Wi-Fi, Bluetooth ja KNX RF.
Võrgutehnoloogiate valikul tuleb arvestada nende leviala, energiatarbimist, seadmete ühilduvust ning süsteemi skaleeritavust. Erinevad tehnoloogiad sobivad erineva suurusega projektidele ning neil kõigil on omad eelised ja piirangud.

3.3.1 Zigbee

Zigbee on juhtmevaba sideprotokoll, mis töötab sagedusel 2,4 GHz ja võimaldab luua suure hulga seadmete vahelisi võrke. Protokoll kasutab *mesh*-võrgu struktuuri, mis tagab seadmete hea leviala ja töökindluse väiksemates ruumides või kodudes.

Zigbee peamisteks eelisteks on lihtne paigaldus, madal energiatarve ja suhteliselt madal hind. Samas võib juhtmevaba side olla tundlik raadiosageduslike häirete suhtes ning seadmete ühilduvus eri tootjate vahel pole alati täielikult tagatud. Seetõttu kasutatakse Zigbee-protokolli enamasti väiksemates nutikodu-lahendustes, kus rõhk on paindlikkusel ja lihtsusel.

3.3.2 Z-Wave

Z-Wave on juhtmevaba sideprotokoll, mis töötab Euroopas sagedusel 868 MHz. Erinevalt Zigbeest kasutab Z-Wave tavaliselt keskset juhtseadet, mille kaudu toimub kogu võrgu juhtimine ja jälgimine.
Z-Wave'i peamised eelised on kasutajasõbralik seadistamine ja hea eri tootjate seadmete ühilduvus. Samas võib keskne kontroller olla süsteemi haavatavaks punktiks ning piiratud on ka toetatavate seadmete arv võrreldes Zigbee ja teiste tehnoloogiatega. Z-Wave sobib hästi keskmise suurusega nutikodudesse ja lihtsamatesse automatiseerimisprojektidesse [16].

3.3.3 Thread

Thread on juhtmevaba võrguprotokoll, mis töötab samuti sagedusel 2,4 GHz ning toetab IP-põhist suhtlust. Thread kasutab mesh-võrgu struktuuri ja on optimeeritud IoT-seadmete jaoks, pakkudes head energiatarvet ja suurepärast töökindlust väiksemates võrkudes.

Threadi eeliseks on IP-ühilduvus, mis võimaldab seadmete lihtsat integreerimist internetti ja pilveteenustega. Thread sobib hästi nutikodude ja väiksemate äriprojektide jaoks, kuid selle levik turul on seni olnud piiratud ning seadmete kättesaadavus võib olla väiksem võrreldes Zigbee ja Z-Wave tehnoloogiatega [17].

3.3.4 Wi-Fi ja Bluetooth

Wi-Fi on üks levinumaid juhtmevaba sideprotokolle, mida kasutatakse laialdaselt nii kodudes kui ka äriruumides. Wi-Fi võimaldab suuri andmeedastuskiirusi ja head ühilduvust eri seadmetega. Selle peamisteks eelisteks on suurepärane leviala, väga hea ühilduvus ja lihtne paigaldus. Samas on Wi-Fi energiatarve kõrgem kui Zigbee või Thread protokollidel.

Bluetooth on juhtmevaba tehnoloogia, mida kasutatakse peamiselt lühikese ulatusega sidelahendustes. Selle eelisteks on madal energiatarve, lihtne kasutus ja hea ühilduvus mobiilseadmete ning tarvikutega. Piiratud leviala tõttu sobib Bluetooth eelkõige väiksematesse ruumidesse ja seadmete vahel, mis asuvad üksteisele lähedal [18].

3.3.5 KNX RF Multi ja KNX RF Multi S-Mode

KNX RF Multi on KNX-süsteemi juhtmevaba versioon, mis töötab raadiosagedusel 868 MHz ja toetab turvalist suhtlust KNX-seadmete vahel ilma kaabliteta. See tehnoloogia pakub tuge mitmekanalilisele sidele, dünaamilisele võrguhaldusele ja paremale seadmete avastamisele, võimaldades paindlikku ja laiendatavat hooneautomaatikat ka siis, kui kaabeldus pole võimalik või soovitav.

KNX RF Multi S-Mode (System Mode) on RF Multi täiendus, mis lisab struktureerituma seadistamisviisi ning tagab parema koostalitlusvõime olemasolevate KNX TP süsteemidega. S-Mode võimaldab seadistusi sarnaselt juhtmega KNX-ile ning sobib hästi professionaalseteks rakendusteks, kus on oluline süsteemi stabiilsus, turvalisus ja hooldatavus.

KNX RF Multi SLE (Secure Low Energy) on veelgi energiatõhusam KNX RF versioon, mis ühendab madala energiatarbe Bluetooth-tehnoloogia eelised KNX-standardile omase turvalisuse ja töökindlusega. See režiim sobib patareitoitel seadmetele, näiteks anduritele ja lülititele, võimaldades neil töötada pikka aega ilma hooldusetа ning säilitades samas tugeva krüpteerimise ja autentimise.

Tabelis 1 tehakse võrdlus nende tehnoloogiate vahel.

Tabel 1. Võrdlus Zigbee, Z-Wave, Thread, Wi-Fi/Bluetooth ja KNX RF Multi vahel [19], [20], [21], [22], [23].

Omadus	**Zigbee**	**Z-Wave**	**Thread**	**Wi-Fi/ Bluetooth**	**KNX RF**
Side tüüp	Juhtmevaba (*mesh*)	Juhtmevaba (keskne/ *mesh*)	Juhtmevaba (*mesh*, IP)	Juhtmevaba (BT vajadusel ka *mesh*)	Juhtmevaba (täiustatud *mesh*)
Sagedus	2,4 GHz	868 MHz	2,4 GHz	Wi-Fi = 2,4/5 GHz, BT = 2,4 GHz	868 MHz
Energiatarve	Väga madal	Väga madal	Väga madal	Wi-Fi = kõrge, BT = madal	Madal, (SLE = väga madal)

Omadus	**Zigbee**	**Z-Wave**	**Thread**	**Wi-Fi/ Bluetooth**	**KNX RF**
Andme-edastuskiirus	Keskmine (~250 kbps)	Madal (~100 kbps)	Keskmine (~250 kbps)	Wi-Fi = Gbps+, BT = kuni 2 Mbps	*Fast ch.*= 16.4kbit/s, *Slow ch.*= 8.2kbit/s
Levikuala	Hea (10–100m)	Hea (30–100m)	Keskmine (15-50m)	Wi-Fi =50–100m, BT = 5–10m	Hea (20–150m)
Ühilduvus	Keskmine	Kõrge	Kõrge	Väga kõrge	Väga kõrge – KNX standard
Paigaldamise keerukus	Madal	Madal	Keskmine	Madal	Keskmine – ETS tarkvara nõutav

3.4 Rakenduskiht ja sõnumivahetusprotokollid

Rakenduskiht ja sõnumivahetusprotokollid määravad seadmete vaheliste sõnumite edastamise loogilised reeglid ja meetodid. Need protokollid võimaldavad seadmetel tõhusalt ja usaldusväärselt suhelda sõltumata sellest, milline füüsiline sidekanal või võrgutehnoloogia on kasutusel. Rakenduskihtide protokollid on sageli olulised IoT-seadmete haldamiseks ning andmeedastuseks pilveteenuste või teiste süsteemidega.

Rakendusprotokollide valimisel tuleb arvesse võtta protokollide keerukust, energiatarvet, andmeedastuse kiirust ja süsteemi skaleeritavust. Järgnevalt tutvustatakse populaarseid rakendusprotokolle MQTT, CoAP ja HTTP (*HyperText Transfer Protocol*)/REST(*Representational State Transfer*), tuues välja nende omadused ja kasutusvaldkonnad.

3.4.1 MQTT

MQTT (*Message Queuing Telemetry Transport*) on rakenduskiht protokoll, mis toimib TCP/IP-võrgus ja sobib väikeste sõnumite efektiivseks edastamiseks IoT-seadmetele. MQTT põhineb publish-subscribe mudelil, kus andmed liiguvad vahendava maakleri (*broker*) kaudu, mitte otseselt seadmelt seadmele.

MQTT peamisteks eelisteks on lihtne ülesehitus, madal võrguressursside tarve ning hea sobivus suure hulga seadmete haldamiseks. MQTT on eriti levinud IoT-rakendustes, kus

on vaja ühendust pidada pilveteenustega või jälgida andureid ja väikese energiatarbega seadmeid [24].

3.4.2 CoAP

CoAP on lihtne, UDP (*User Datagram Protocol*) -põhine rakenduskiht protokoll, mis sobib piiratud ressurssidega seadmetele nagu väikese võimsusega andurid või nutiseadmed. CoAP pakub REST-laadset liidest ja võimaldab lihtsat suhtlust seadmete vahel ning integratsiooni veebiteenustega.

CoAPi peamised eelised on madal energiatarve, efektiivne andmevahetus ning sobivus väikese jõudlusega seadmetele. Samas võib CoAP olla vähem usaldusväärne võrreldes MQTT-ga, kuna see töötab UDP alusel, mis ei garanteeri andmete edastamist [25].

3.4.3 HTTP/REST

HTTP/REST on veebiteenustes ja API-de loomisel kasutatav standardprotokoll, mis võimaldab seadmete ja serverite vahel andmevahetust standardsete HTTP-meetodite abil (*GET, POST, PUT, DELETE*). HTTP/REST sobib hästi olukordades, kus on vaja luua lihtsaid ja laialdaselt toetatud liideseid.

HTTP/RESTi eeliseks on hea ühilduvus olemasolevate veebipõhiste süsteemidega ja lihtne seadistamine. Selle puuduseks on kõrgem energiatarve ja suurem võrguliiklus võrreldes MQTT või CoAP protokollidega, mistõttu HTTP/REST ei pruugi olla parim valik piiratud ressurssidega seadmete jaoks [26].

Tabelis 2 võrreldakse eelnevalt kirjeldatud rakendusprotokolle.

Tabel 2. Võrdlus MQTT, CoAP ja HTTP/REST vahel

Omadus	**MQTT**	**CoAP**	**HTTP/REST**
Transpordiprotokoll	TCP	UDP	TCP
Energiatõhusus	Väga kõrge	Väga kõrge	Keskmine-madal
Andmete edastamine	Publish-subscribe	Request-response	Request-response
Sobivus piiratud seadmetele	Väga hea	Väga hea	Piiratud
Usaldusväärsus	Kõrge	Keskmine	Kõrge
Peamine kasutusala	IoT-seadmed, pilveteenused	Väiksed andurid, IoT-seadmed	Veebirakendused, API-d

Erinevad hooneautomaatika tehnoloogiad, olgu need siis standardid, sideprotokollid või rakenduskihtide lahendused, täidavad erinevaid ülesandeid ja igal neist on oma eelised ning piirangud. Praktilistes projektides kasutatakse sageli mitut tehnoloogiat paralleelselt või kombineeritult, sõltuvalt süsteemi keerukusest, kasutusotstarbest, eelarvest ja integratsioonivajadustest.

4 KNX IoT – arhitektuur, protokollid ja rakendusvõimalused

KNX IoT on KNX-standardi uusim laiendus, mille eesmärk on siduda traditsiooniline hooneautomaatika IP-põhiste võrkude ja kaasaegsete IoT-ökosüsteemidega. Erinevalt klassikalisest KNX-ist (TP/RF/PL) tugineb KNX IoT otse IPv6-le ja kasutab kergekaalulisi veebi- ja andmeformaate, võimaldades skaleeruvat, standardset ning turvalist sidet nii juhtmega kui ka juhtmevabade seadmete vahel [27].

Traditsioonilised hooneautomaatikasüsteemid seisavad üha enam silmitsi vajadusega integreeruda kaasaegsete IoT-ökosüsteemide ja pilveteenustega. IoT-tehnoloogiate rakendamine võimaldab tõhusamat andmekogumist, seadmete kaugjuhtimist, andmete analüüsi ja energiatõhususe parandamist.

4.1 Kontseptuaalne ülevaade

KNX IoT laiendab KNX-sõnumite semantikat IP-põhisesse võrku, rakendades KNX IoT Point API spetsifikatsiooni. Point API määratleb, kuidas KNX-i grupiaadressid, andmetüübid ja datapoint-id esitatakse CoAP/CBOR-vormingus, säilitades tagurpidi ühilduvuse klassikalise KNX-iga [28].

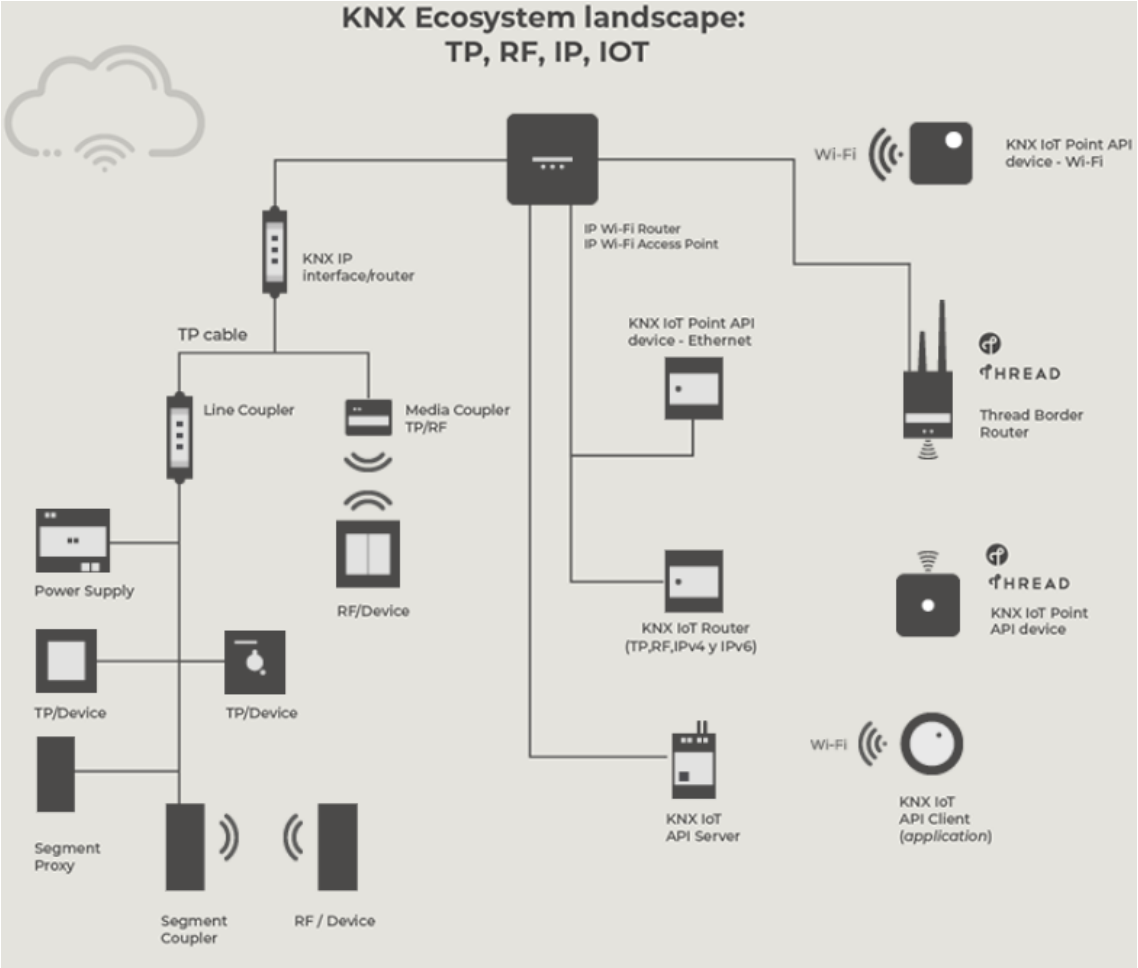
Üks KNX IoT kesksetest põhimõtetest on tsentraliseerimata arhitektuur: iga sõlm võib suhelda otse teistega IPv6 multicast'i kaudu, vältides vajadust keskse kontrollseadme järele. Selline lähenemine tagab parema töökindluse, skaleeritavuse ja vähendab süsteemi haavatavust üksikute punktide riketele.

4.2 Andmekandjad ja hübriidvõrgud

- Ethernet / Wi-Fi (IPv6) – kõrge läbilaskevõime serveri- ja seadmete ühendamiseks.
- Thread (IEEE 802.15.4 + IPv6) – madala energiatarbega silmusvõrk (*mesh*) akutoitel anduritele; Thread Border Router sildab Thread-domeeni ülejäänud IPv6-võrguga [29].
- KNX TP/RF ↔ IPv6 sildamine – KNX IoT-routerid laiendavad olemasoleva siini IP-seadmetega, võimaldades ühist grupiaadressiruumi.

- KNX IoT 3rd-Party API Server – REST/GraphQL-põhine vaheseade, mis kõrvutab KNX-i grupiaadressid veebiressurssideks ja pakub standardiseeritud liidest välistele rakendustele (KNX IoT API Client) [30].

Selline kombinatsioon loob ühtse võrgu, kus klassikalised TP/RF-sõlmed, uued IP-põhised seadmed ning pilve- või mobiilirakendused toimivad kooskõlas. Joonis 3 illustreerib tüüpilist topoloogiat, kus TP-kaabel, RF-seadmed, KNX IoT-routerid, Thread-võrk ja API Server moodustavad integreeritud arhitektuuri.



Joonis 3. KNX-i topoloogia [31]

4.3 Uus andmeedastusprotokoll

KNX IoT kasutab spetsiaalset protokolli CoAP, mis on mõeldud piiratud ressurssidega seadmetele, näiteks anduritele ja mikrokontrolleritele. CoAP sarnaneb struktuurilt veebiprotokollile HTTP, kuid on optimeeritud väiksemale energiatarbele ja ressursside säästmisele. CoAP toetab ka grupisõnumite edastamist, mis on oluline suurtes võrkudes, kus andmete samaaegne saatmine mitmele seadmele on vajalik.

Lisaks kasutatakse andmete kodeerimiseks CBOR-vormingut (*Concise Binary Object Representation*), mis on kompaktsem ja tõhusam võrreldes traditsiooniliste tekstipõhiste formaatidega nagu JSON (*JavaScript Object Notation*) või XML (*Extensible Markup Language*). CBOR vähendab andmete edastamisega seotud koormust võrgule ja seadmetele, võimaldades paremat jõudlust ja madalamat energiatarvet [32].

4.4 Turvalisus

KNX Secure tehnoloogia pakub tugevat kaitset klassikalistes KNX-süsteemides, rakendades krüpteerimismehhanisme, mis hõlmavad IP-side turvamist ja kaitset volitamata juurdepääsu eest. KNX Secure tagab, et ainult volitatud seadmed ja kasutajad saavad ligipääsu süsteemi juhtimisele ja andmetele, vähendades võimalusi manipuleerimiseks ja häkkimiseks.

KNX IoT puhul kasutatakse lisaks OSCORE (*Object Security for Constrained RESTful Environments*) protokolli, mis võimaldab turvalist ja krüpteeritud andmevahetust ka piiratud ressurssidega seadmete vahel, kasutades CoAP ja CBOR formaate. OSCORE krüpteerib kogu kommunikatsiooni otspunktist otspunktini, kaitstes mitte ainult edastatava teabe sisu, vaid ka selle metaandmeid. Selline lahendus muudab KNX IoT süsteemid vastupidavaks küberrünnakutele ning tagab andmete terviklikkuse ja konfidentsiaalsuse [33].

4.5 Tugevused, väljakutsed ja kokkuvõte

Alljärgnevalt on kokku võetud KNX IoT-lahenduse peamised tugevused ja olulisemad väljakutsed

Tugevused:

- IP-neutraalsus – sama protokoll toimib Etherneti, Wi-Fi ja Threadi peal;
- Standardne semantika – klassikalised KNX grupiaadressid säilivad;
- Avatud lähtekood ja laienev tootjate toetus.

Väljakutsed:

- Krüptograafia koormus vähema jõudlusega kontrolleritel;
- Sertifitseerimisprotsess alles väljatöötamisel;
- Turu varajane faas – kommertstoodete valik on hetkel piiratud.

Kokkuvõtteks võib öelda, et KNX IoT loob silla klassikalise hooneautomaatika ja IP-põhiste IoT-võrkude vahel. IPv6-kesksed protokollid ning standardiseeritud turvamehhanismid võimaldavad tulevikukindlat ja koostalitlusvõimelist lahendust, kuigi seadmete sertifitseerimise ja ökosüsteemi laienemisega seotud küsimused vajavad veel arendamist.

5 Tarkvarapõhised testkeskkonnad ja praktilised katsetused

Virtuaalsed tööriistad võimaldavad modelleerida KNX-põhiseid süsteeme enne füüsilise riistvara rakendamist, vähendades kulusid ja kiirendades arendustsüklit. Käesolevas peatükis kirjeldatakse magistritöö käigus läbi viidud katseid KNX Virtual-i, KNX IoT Virtual-i ja ETS 6-ga. Eesmärk oli luua realistlikud teststseenid, katsetada kaugkonfigureerimist ning valmistada ette platvorm ESP32-põhise prototüübi integreerimiseks.

5.1 KNX Virtual ja integratsioon ETS 6-ga

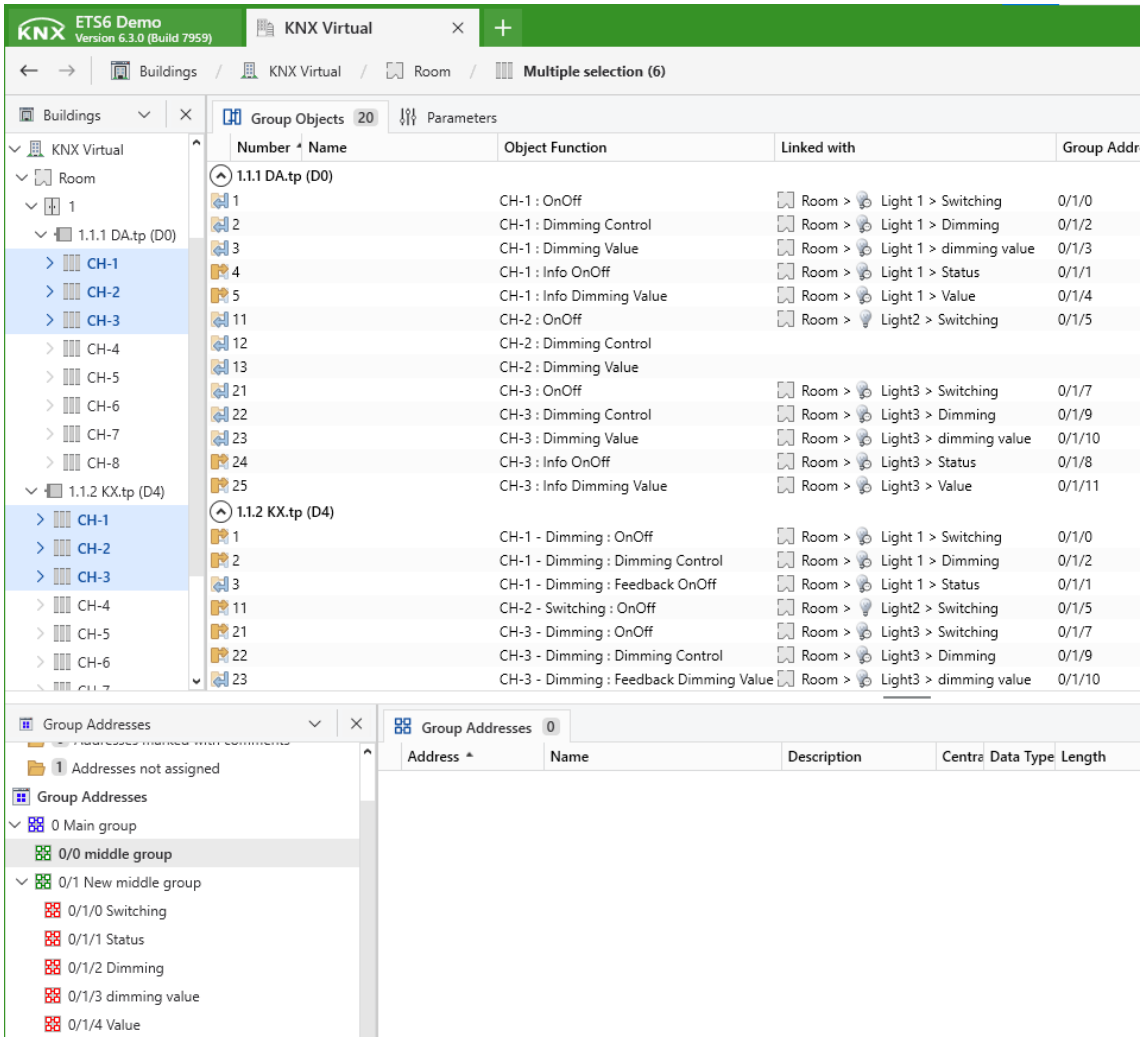
KNX Virtual on Windowsi-keskkonnas töötav KNX-süsteemi simulaator. Selle peamine eesmärk on võimaldada KNX-funktsionaalsuse õppimist ja testimist ilma füüsilisi seadmeid kasutamata. KNX Virtual sisaldab virtuaalseid seadmemudeleid, nagu lambi moodulid, lülitid, temperatuuri- ja liikumisandurid, samuti loogikamooduleid, mida saab konfigureerida ETS-tarkvara (Engineering Tool Software) abil täpselt samamoodi nagu reaalseid KNX-seadmeid [34].

Käesoleva töö raames viidi läbi uuring KNX Virtuali lisavõimaluste kohta, sealhulgas ETS6 kaugühenduse seadistamine selliselt, et ETS6 oleks installitud ühele arvutile ning KNX Virtual käivitatud teisel arvutil. Selline lähenemine loob oludele võimalikult sarnase keskkonna, nagu reaalses võrgus toimuvas seadmete suhtluses, jäljendades IoT-lahendusi kohalikus võrgus. Seejuures kasutati KNX-i ametlikku juhist, mis käsitleb IP-liidese seadistamist ETS-i ja KNX Virtuali vahel. Tulemuseks saadi kogemus KNX-projektide seadistamisel kaugtöökohtadest ning ülevaade sellest, kuidas IP-võrgu kaudu toimub telegrammide vahetus virtuaalseadmete ja ETS-i vahel.

Praktiline katse:

- Paigaldati KNX Virtual (v2.6) arvutisse A; konfiguratsioonikaustas loodi .txt-fail, kuhu kirjeti arvuti B IP-aadress ja port. See fail võimaldab hiljem ETS-il kaugühenduse luua [35].
- Arvutisse B paigaldati ETS 6 (Demo) ning lisati IP-interface, mis kasutab sama aadressi ja porti, mis määrati faili arvutis A.

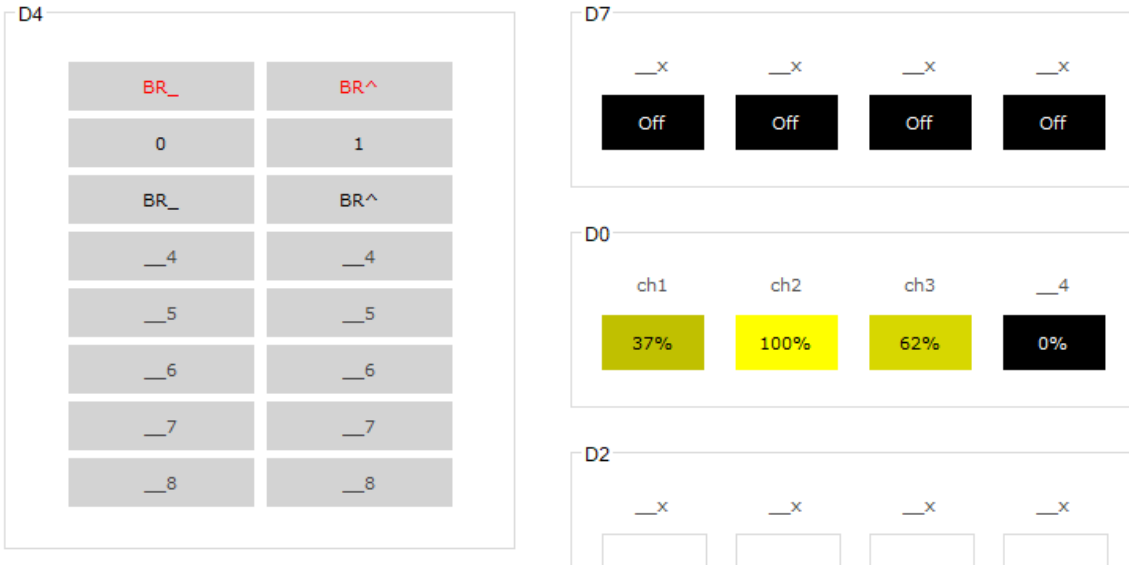
- Loodi projekt Demo – KNX Virtual, lisati 3 virtuaalset lampi ja 3 lülitit; gruppaadresside struktuur „0/1/x“ (vt Joonis 4).



Joonis 4. ETS 6 projektivaade – gruppobjektid ja aadressid

- Konfiguratsioon kirjutati KNX Virtuali.
- Testiti lampide (*channel* 1, 2 ja 3) CH1, CH2 ja CH3 sisselülitamist;

KNX Virtuali graafiline liides kuvab lampide oleku vastavalt ETS-is määratud loogikale (vt Joonis 4 ja Joonis 5). Viidatud joonistel on D4 klahvlülitite moodul ja D0 *dimmeri* moodul.



Joonis 5. KNX Virtual – lülitite ja lampide moodulite olek ETS 6 konfiguratsiooni alusel

Sellise protsessi abil kontrolliti edukalt seadmete kaugkonfigureerimist, testiti virtuaalse KNX-süsteemi ja ETS6 koostööd ning kinnitati simulaatori kasutusmugavust reaalsete KNX-installatsioonide häälestamisel ja silumisel.

5.2 KNX IoT Virtual

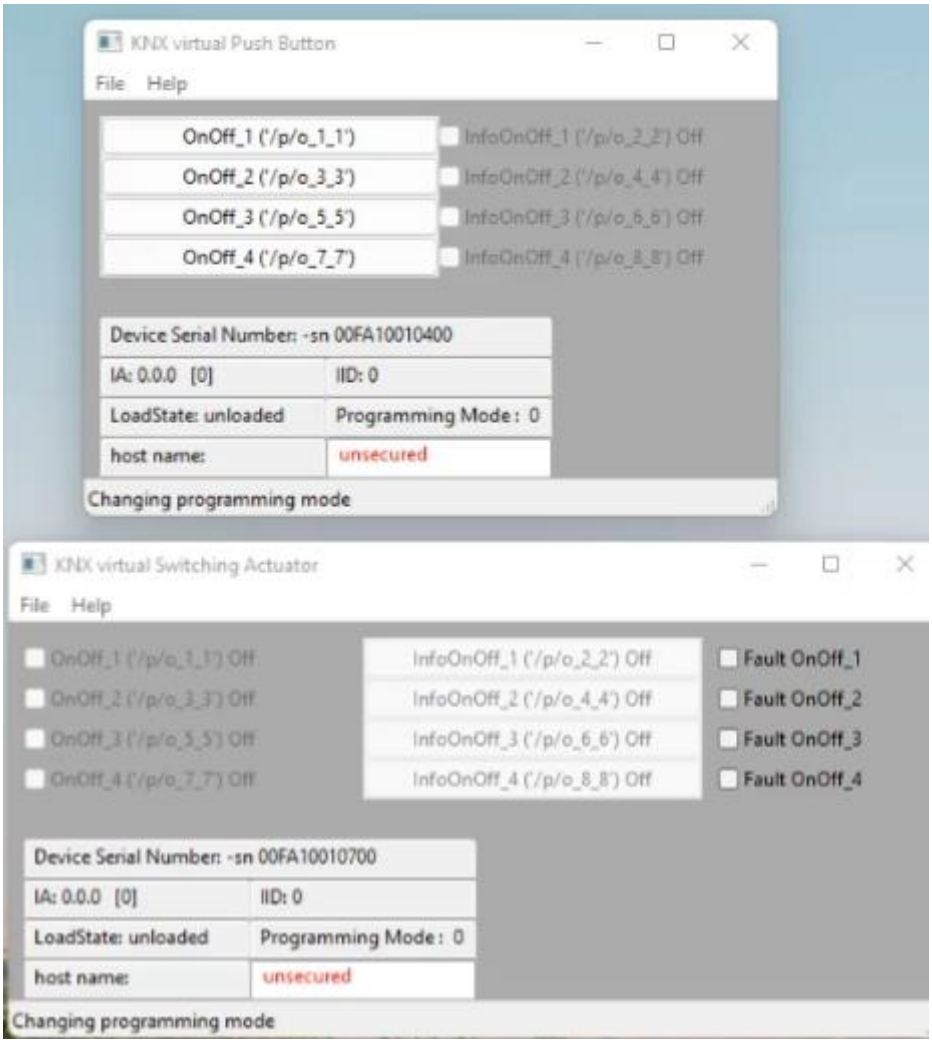
KNX IoT Virtual on uuem tööriist KNX IoT-seadmete simuleerimiseks ning kuulub KNX-i IoT-tehnoloogiate integreerimise algatuse juurde. Erinevalt klassikalisest KNX Virtualist modelleerib KNX IoT Virtual IP-võrgus töötavaid seadmeid, mis kasutavad CoAP-protokolli ning CBOR-andmevormingut. See võimaldab simuleerida nüüdisaegseid IP-keskseid KNX IoT-seadmeid ilma kuluka riistvara ostmiseta [36].

Käesoleva töö raames viidi läbi lihtsa teoreetilise näite väljatöötamine kahe virtuaalse seadme integreerimisest: neljakanalilise virtuaalse lüliti (*Push Button*) ning neljakanalilise aktuaatori (*Switch Actuator*) omavaheline suhtlus (vt Joonis 6). KNX IoT Virtuali kasutamine hõlmas järgmisi etappe:

- KNX IoT Virtuali lähtekoodi allalaadimine ametlikust GitHubi repositooriumist ning kompileerimine Windows-arvutis [37].
- Virtuaalse lüliti (Push Button) ja virtuaalse täiturseadme (Switch Actuator) käivitamine.
- ETS6-projekti loomine, kuhu lisatakse KNX IoT tüüpi seadmeid spetsiaalsest kataloogist.

- Valmis konfigureeritud projekti eksportimine ETS6-st KNX IoT Virtualiga ühilduvasse vormingusse (KNX IoT Information Model).

Antud eksperiment kinnitas KNX IoT-seadmete modelleerimise võimalikkust, mis annab arusaamist ja põhiteadmised sellest, kuidas tulevikus reaalsete seadmetega töötada.



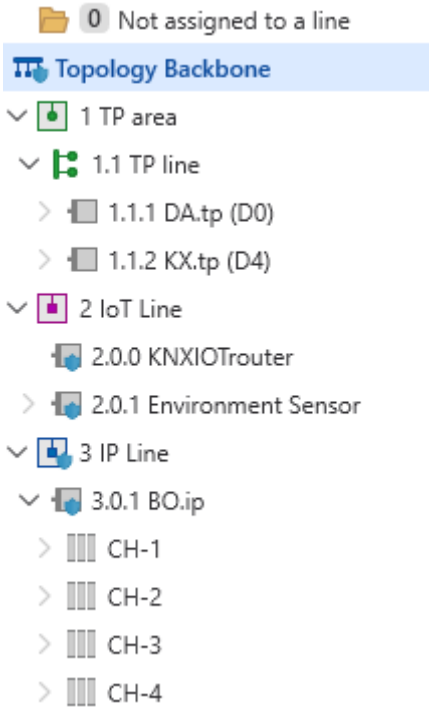
Joonis 6. Virtuaalsed KNX Iot Point Api seadmed [38].

5.3 ETS 6 – hübriidsete projektide keskne tööriist

ETS 6 on praktiline projekteerimislahendus, mis võimaldab ühes ja samas projektis hallata nii klassikalisi KNX TP/RF-seadmeid kui ka IP-põhiseid KNX IoT sõlmi. Selline "segaprojekt" lihtsustab üleminekut uutele IoT-komponentidele ilma olemasolevat siinijuhtmestikku muutmata.

Praktilises kasutuses uuriti järgnevaid ETS6 võimalusi:

- Hübriidtopoloogia loomine: TP-liin, IoT-liin ning eraldi IP-liin konfigureeriti ühes projektis (vt Joonis 7).



Joonis 7. Näide, kuidas ETS6 ühes projektis kasutada paralleelselt TP-liin, IoT-liin, IP-liin jne.

- Grupiaadresside ja objektide seadistamine: määrati ühtne gruppaadressi-skeem „0/1/x“, mis kehtib kõigile liinidele, et tagada loogiline ühtsus.

ETS6 kasutamise kogemus kinnitas selle kasutusmugavust ja ühilduvust uute IoT-komponentidega (näiteks CASCODA), muutes selle tööriistaks nii traditsiooniliste kui ka innovaatiliste KNX-projektide kavandamisel. Samuti on ETS6-s märgatavalt lihtsustatud eri tootjate seadmete integreerimise protseduuri.

Autori arvates tõestasid sellised tarkvarad nagu näiteks KNX Virtual ja ETS6 oma kasu nii õppimisel kui ka keerukate automaatikaprojektide praktilisel ettevalmistamisel. Käesoleva töö raames läbiviidud eksperimendid ja testid kinnitasid nende sobivust uute IoT-lahenduste kontrollimiseks ja katsetamiseks.

6 ESP32 baasil CoAP-serveri arendus ja integreerimine KNX IoT-ga

Tänapäevased IoT-süsteemid vajavad üha sagedamini kergeid ja energiatõhusaid protokolle. CoAP vastab nendele nõuetele, pakkudes lihtsat ja standardiseeritud viisi madala jõudlusega seadmete suhtlemiseks üle IPv4 ja IPv6 võrkude. Käesoleva töö praktilises osas on valitud ESP32 mikrokontroller, mis oma võimsuse, võrguühenduse võimaluste (Wi-Fi, Ethernet, Bluetooth), mitmekülgsuse ja madala hinna tõttu sobib IoT-sõlmede loomiseks. ESP32 ja CoAP-protokolli kombinatsioon võimaldab lihtsat ja efektiivset integreerimist KNX IoT-süsteemidega (vt Joonis 15), pakkudes paindlikku lahendust nutika hoone automatiseerimiseks ja energiatarbimise optimeerimiseks [39].

6.1 ESP32 ja CoAP-serveri tarkvara arendus

Projekti arendus algas ESP-IDF (*Espressif IoT Development Framework*) keskkonna põhjaliku seadistamisega Visual Studio Code'is. ESP-IDF on Espressifi ametlik arenduskeskkond, mis lihtsustab ja kiirendab tarkvara loomist ESP32 platvormile. Projekti struktuuri tuumaks valiti libcoap teek, mis vastab RFC 7252 spetsifikatsioonile ning toetab üheaegselt IPv4 ja IPv6 protokolle. Selline valik tagab ühildatavuse kaasaegsete IoT süsteemidega.

CoAP-server käivitatakse FreeRTOS (*Free Real-Time Operating System*) ülesandena. Server kuulab UDP-päringuid standardpordil 5683 ning suudab hallata mitut paralleelset ühendust. Serveri käivitamise ja ressursside loomise näide on järgmine (Joonis 8):

```
// Käivitame FreeRTOS-töö CoAP-serveriga
// Ülesan., nimi, suurus, param., prior., no_save
xTaskCreate(coap_server_task, "coap_server", 8*1024, NULL, 5, NULL);

static void coap_server_task(void *p)
{
// Loome CoAP-kontekst (vaikeseaded)
    coap_context_t *ctx = coap_new_context(NULL);
// Kui ei õnnestu
    if (!ctx) { ESP_LOGE(TAG, "No CoAP"); vTaskDelete(NULL); }
// IPv6-aadress ja port 5683 (kõik aadressid)
    coap_address_t a6;
    coap_address_init(&a6); //nullime
    a6.addr.sin6.sin6_family = AF_INET6; // IPv6
    a6.addr.sin6.sin6_port   = htons(COAP_DEFAULT_PORT);
// in6addr_any on vaikimisi seatud, pole vaja eraldi määrata

// UDP-endpoint loomine, et vastu võtta CoAP-päringuid IPv6 kaudu
    coap_new_endpoint(ctx, &a6, COAP_PROTO_UDP); }
```

Joonis 8. Koodinäide CoAP-serveri loomisest ESP32-l

Täiendavalt loodi ka IPv4 võimekus, võimaldades laiemat ühilduvust olemasolevate võrkudega.

6.2 CoAP-ressursside loomine ja haldamine

CoAP protokollis kasutatavad ressursid meenutavad URL-ide struktuuri, võimaldades klientidel päringuid ja käske hõlpsasti teha. Käesolevas projektis realiseeriti mitu põhiressurssi: erinevat tüüpi andurid (/*sensor*/), mis võimaldavad seadme mõõdetavaid väärtusi lugeda, ning täiturseadmed (/*actuator*/), mida saab kaugjuhtimise teel juhtida. Igal ressursil on vastavad *callback*-funktsioonid päringute (GET, PUT, DELETE) töötlemiseks.

Joonis 9 esitab näite GET callback-funktsioonist LED (*Light-Emitting Diode*)-i juhtimiseks loodud ressursile.

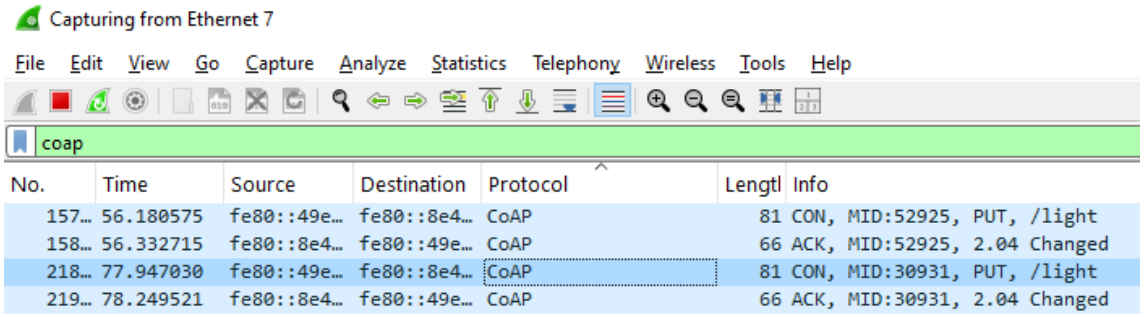
```
// GET-päring `/light` jaoks: saadab LED-i oleku CBOR-formaadis
static void hnd_light_get(coap_resource_t *r, coap_session_t *s,
                          const coap_pdu_t *req,
                          const coap_string_t *q,
                          coap_pdu_t *resp)
{
    uint8_t buf[8]; // puhver CBOR-i jaoks
    CborEncoder enc; // CBOR-enkooder
    cbor_encoder_init(&enc, buf, sizeof(buf), 0); // enkod. init.
    cbor_encode_boolean(&enc, led_on); // enkod Led-i olek
    size_t len = cbor_encoder_get_buffer_size(&enc, buf); //enkod.
Pikkus

    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CONTENT); //2.05 cont
    uint8_t opt[4]; // puhver Content-Format
    size_t ol = coap_encode_var_safe(opt, sizeof(opt), CF_CBOR);//cbor
    coap_add_option(resp, COAP_OPTION_CONTENT_FORMAT, ol, opt);
    coap_add_data(resp, len, buf);} // päringu andmed(CBOR-puhver)
```

Joonis 9. CoAP-ressurssi GET callback-funktsioonide näide

6.3 CoAP-serveri testimine ja analüüs

Serveri põhjalikuks testimiseks kasutati libcoap tööriista coap-client. Testid hõlmasid päringute edastamist ja töötlemist nii IPv4 kui IPv6 kaudu, kinnitades ESP32 serveri korrektset ja efektiivset toimimist erinevate võrgutingimuste korral. Lisaks testimisele teostati võrguliikluse süvitsi analüüs Wireshark-i abil, tagamaks täielikku vastavust CoAP protokolli spetsifikatsioonile ning tuvastamaks võimalikke anomaaliaid sõnumite töötlemisel (Joonis 10). Server näitas testide jooksul töökindlust ja korrektset CoAP-sõnumite töötlemise võimekust.



Joonis 10. Wireshark programmi liikluse analüüs

Wireshark'i kuvatõmmisel on näha CoAP-liiklust: iga real kuvatakse paketi number, ajatempli (saatmise/vastuvõtu aeg), kliendi ja ESP32 CoAP-serveri IPv6-aadressid,

protokoll (CoAP), sõnumi pikkus ning Info-s veerus tüüp (CON/ACK), sõnumi ID, PUT-päring ressursile /*light* ja vastuses standardne kood *2.04 Changed*.

6.4 CoAP-serveri ettevalmistamine integreerimiseks KNX IoT-süsteemi

ESP32 baasil loodud CoAP-serveri järgmiseks etapiks on selle ettevalmistamine integreerimiseks KNX IoT-keskkonda. See eeldab sõlme valmisolekut hõlpsaks suhtluseks Node-RED platvormiga, mis omakorda võimaldab sujuvat andmevahetust KNX-i TP-liiniga. Node-RED töötab vahendajana, edastades CoAP-serverilt saadud andmed ja käsud otse KNX seadmetele.

CoAP sõnumite kasutamine Node-RED platvormil võimaldab mitmekesiseid rakendussenaariume, näiteks valgustuse juhtimist, temperatuurinäitude jälgimist ja ohutuse tõhustamist nutika hoone keskkonnas. See lähenemine loob kindla aluse järgmiseks peatükiks, kus põhjalikumalt kirjeldatakse Node-RED serveri täpsemat seadistamist ning detailset integratsiooniprotsessi KNX IoT süsteemiga. Käesolev ESP32 ja CoAP protokolli põhine lahendus kinnitab oma sobivust nutikate IoT-süsteemide loomisel.

7 Node-RED platvormi kasutamine KNX IoT-serverina

Kaasaegsete IoT-süsteemide jaoks on oluline leida paindlik ja kasutajasõbralik platvorm, mis võimaldaks kiiresti ja mugavalt integreerida erinevaid seadmeid ja protokolle. Node-RED on selliseks ülesandeks sobiv keskkond, sest see pakub intuitiivset graafilist liidest ning mitmesuguseid lisamooduleid, sealhulgas toetust KNX- ja CoAP-protokollidele.

Node-RED võimaldab kasutajal luua kommunikatsioonisõlmesid („*node*“), mis ühendavad omavahel erinevaid võrguteenuseid ja riistvaraseadmeid. See teeb sellest tööriista ESP32 ja KNX IoT-seadmete integreerimiseks, pakkudes samal ajal efektiivset visuaalset arenduskeskkonda ja juhtimisliidest (*dashboard*). Tänu oma lihtsale, kuid võimsale struktuurile ja kasutajasõbralikkusele on Node-RED saanud populaarseks lahenduseks IoT ja tööstusautomaatika rakendustes [40].

7.1 Node-RED platvormi seadistamine ja põhimõtted

Node-RED eristub eelkõige lihtsuse ja kasutusmugavusega. Platvorm põhineb Node.js raamistil, mis tagab selle laialdase ühilduvuse erinevate operatsioonisüsteemidega nagu Linux, Windows ja macOS. Node-RED installatsioon on kiire ja lihtne, kasutades tavalisi käsurea tööriistu ja npm (*Node Package Manager*) paketihaldurit.

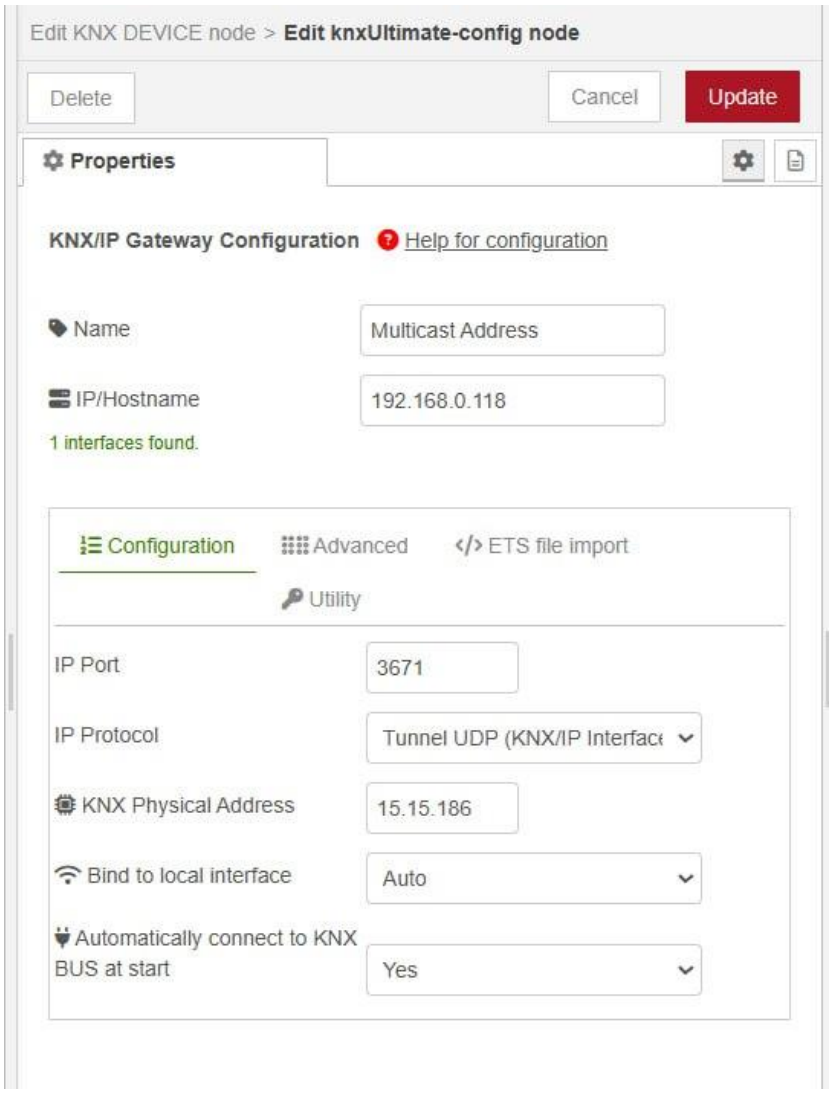
Node-RED üks peamisi eeliseid on lai valik valmismooduleid, mis võimaldavad integratsiooni erinevate seadmete ja protokollidega minimaalse programmeerimisvajadusega. Käesolevas töös kasutati laialdaselt kahte olulist moodulit: KNX Ultimate ja CoAP request. Need võimaldasid kiiresti ja mugavalt luua ühendusi KNX ja CoAP protokollidel põhinevate seadmetega.

7.2 CoAP ja KNX sõlmede seadistamine Node-RED'is

Käesoleva projekti raames seadistati kaks peamist sõlmetüüpi Node-RED keskkonnas:

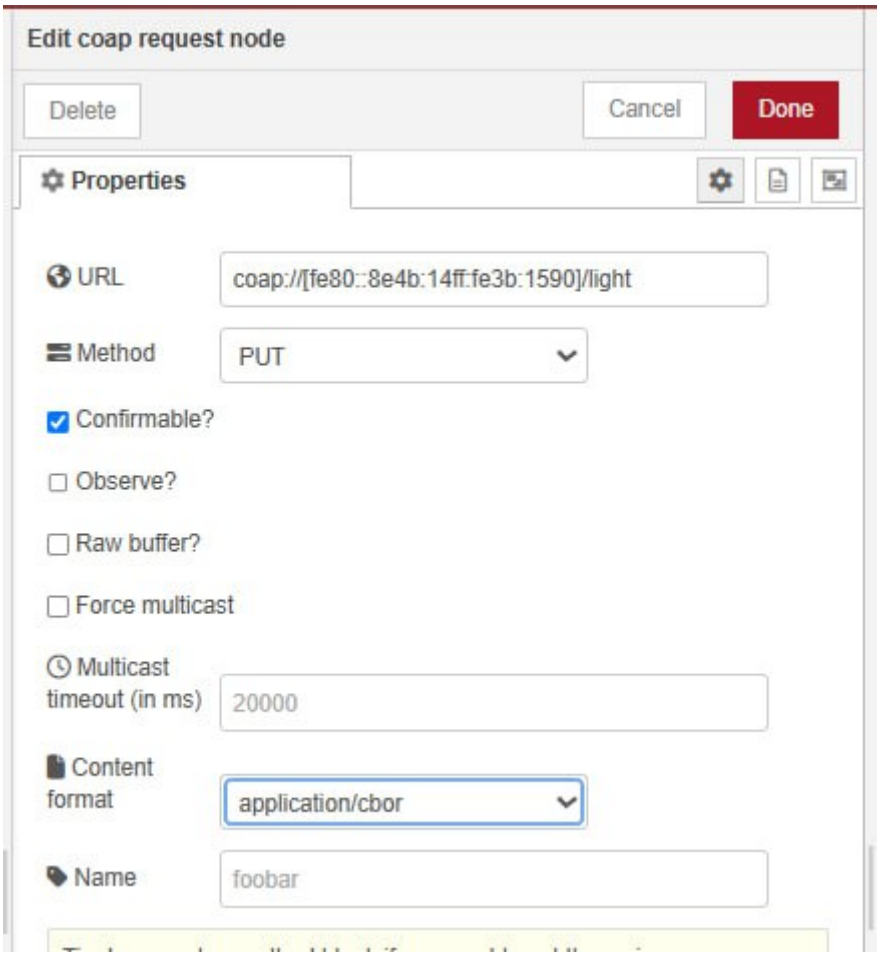
- KNX Ultimate – spetsialiseeritud sõlm KNX võrgule, mis võimaldab saata käske ja jälgida KNX-võrgu olekuid.
- CoAP request – sõlm, mis teeb päringuid ESP32 CoAP-serverile ja edastab sellele käske.

KNX Ultimate sõlm nõuab seadistamisel spetsiifilisi KNX-võrgu parameetreid, nagu grupiaadressid ja füüsilised seadme aadressid (vt Joonis 11). Sõlm ühendub KNX-võrguga KNX/IP gateway (tegelik seade on KNX/IP-ruuter) kaudu, mis tagab stabiilse kommunikatsiooni KNX seadmetega.



Joonis 11. KNX DEVICE sõlme seadistamine Node-RED serveril

CoAP request sõlm seadistatakse ESP32 CoAP-serveriga otseseks kommunikatsiooniks. Konfiguratsioon sisaldab ESP32 IP-aadressi ning soovitud ressursi URL-i, näiteks „*/light*“ (vt Joonis 12). Selline konfiguratsioon võimaldab süsteemil kiiresti reageerida kasutajapoolsetele päringutele ja käskudele.

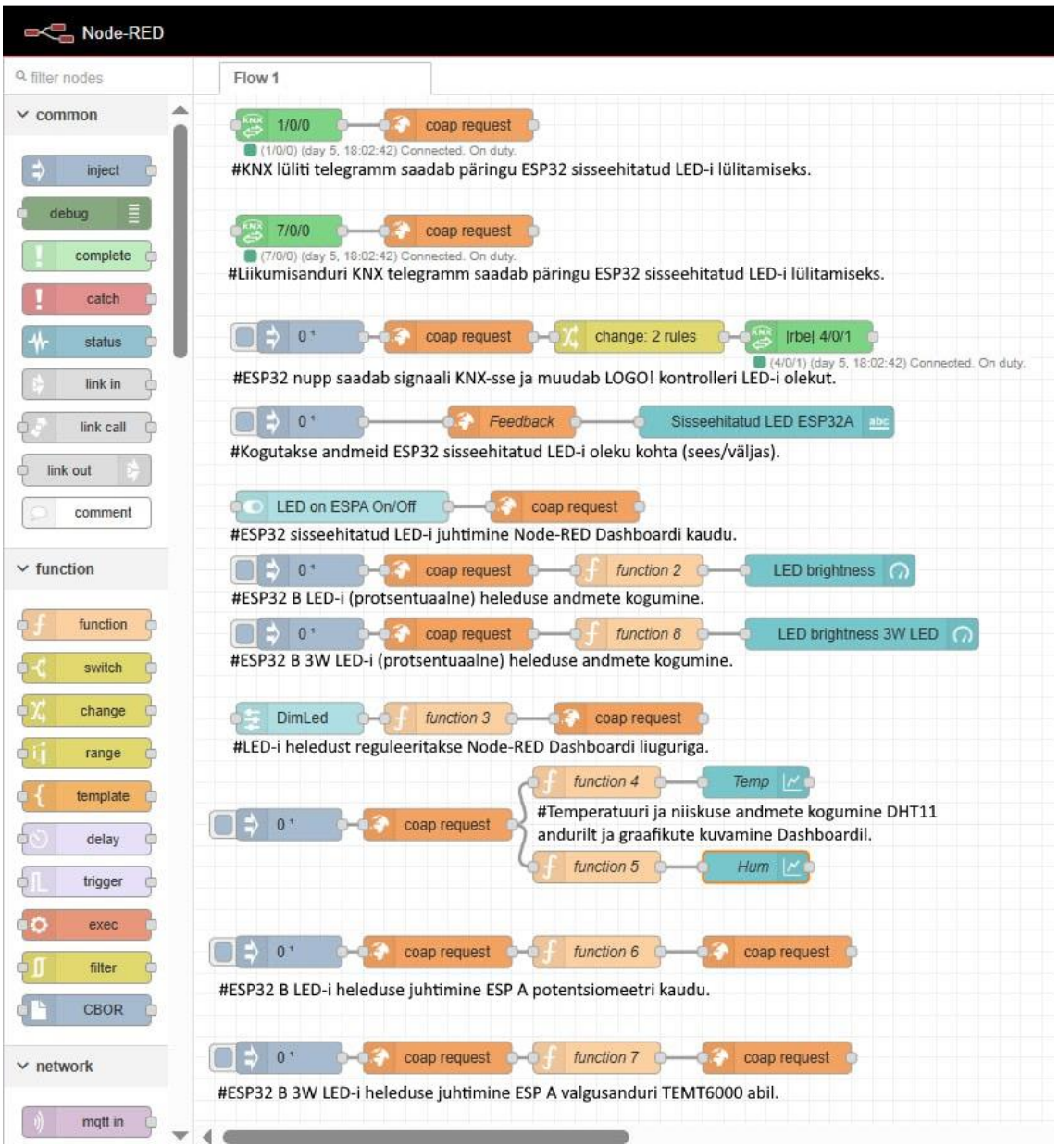


Joonis 12. coap request sõlme konfiguratsioon Node-RED serveril

7.3 Kahepoolse side realiseerimine ESP32 ja Node-RED vahel

Projekti üks võtmefunktsionaalsusi oli kahepoolse kommunikatsiooni loomine ESP32 seadme ja Node-RED platvormi vahel (vt üldskeem Joonisel 15). Node-RED on võimeline vastu võtma tagasisidet ESP32-lt, näiteks LED-i praeguse oleku kohta, mis võimaldab reaalajas jälgida seadme seisundit ja vajadusel kiiresti reageerida. Selle funktsionaalsuse tagamiseks on ESP32-l seadistatud väljund CoAP-sõnumid, mis edastatakse automaatselt Node-RED keskkonda iga kord, kui seadme olek muutub.

Joonis 13 esitab tervikliku vooskeemi, mis illustreerib sõlmede omavahelisi ühendusi ja andmevoogu ESP32, Node-REDi ja KNX süsteemide vahel. Selline visuaalne skeem aitab kasutajatel ja arendajatel paremini mõista süsteemi tööd ja hõlbustab tõrkeotsingut ning süsteemi laiendamist tulevikus.



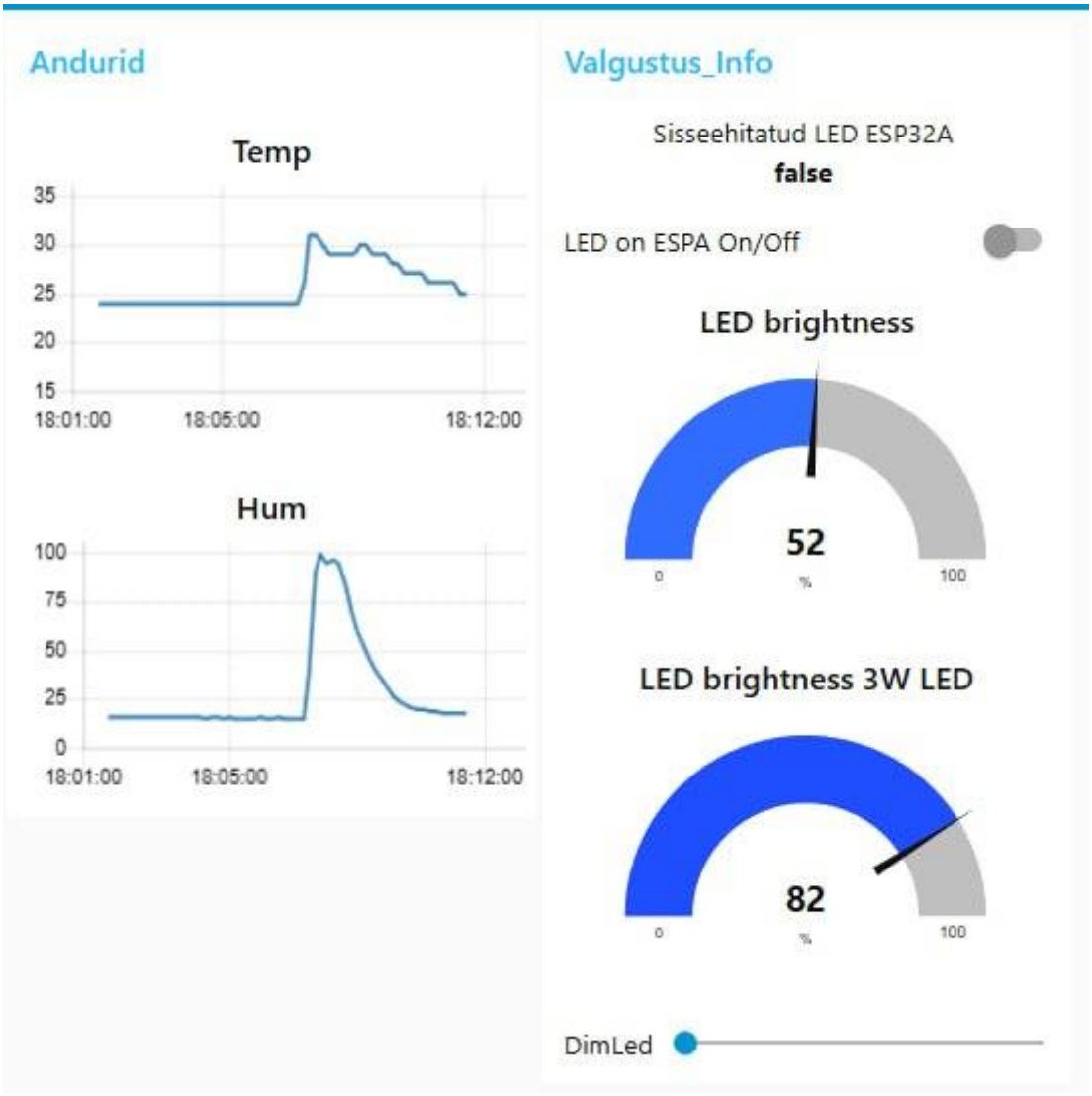
Joonis 13. Node-RED'i diagramm

7.4 Node-RED *dashboard*'i kasutamine ja võimalused

Node-RED sisaldab sisseehitatud veebipõhist juhtpaneeli (*dashboard*), mis võimaldab lõppkasutajatel jälgida ja juhtida süsteeme mugavalt otse veebilehitseja kaudu. *Dashboard* pakub graafilisi kasutajaliidese elemente, nagu lülitid, indikaatorid ja graafikud, mis aitavad selgelt ja arusaadavalt esitada süsteemi olekuid ja juhtida seadmeid.

Antud projektis võimaldab *dashboard* kasutajatel reaalajas jälgida ESP32 seadme LED-indikaatorite olekuid ning juhtida nende sisse- ja väljalülitamist. Selline lahendus pakub

kasutajatele mugavat ja intuitiivset juhtimisvõimalust ilma täiendava tarkvara paigaldamiseta (Joonis 14). Joonisel on kujutatud juhtpaneeli kasutajaliides. Vasakul pool kuvatakse andurite andmed – temperatuur ja õhuniiskus. Paremal pool on näha tagasiside sisseehitatud LED-i oleku kohta (väärtus „*false*“ tähendab, et LED on välja lülitatud). Selle all asub lüliti, millega saab LED-i otse paneelilt sisse või välja lülitada. Järgmisena on kaks näidikut, mis kuvavad kahe LED-i heledust vastavalt protsentides. Kõige all on liugur, millega saab reguleerida kolmanda LED-i heledust.



Joonis 14. *Dashboard*i näide

Node-RED *dashboard*'i kasutamine annab lisaväärtust kogu IoT-süsteemile, pakkudes lihtsat ja arusaadavat kasutajakogemust. Lisaks võimaldab *dashboard* tulevikus hõlpsasti lisada uusi funktsioone ja seadmeid vastavalt kasutajate vajadustele.

8 ESP32 baasil KNX IoT testseadme arendus ja praktiline katsetamine

Käesolevas peatükis kirjeldatakse seda praktilist osa magistritööst, mis hõlmab KNX IoT testseadmete arendamist ESP32 mikrokontrolleri platvormil, kasutades CoAP protokolli, ning nende seadmete integreerimist olemasoleva KNX süsteemiga. Peatüki raames antakse ülevaade seadmete ülesehitusest, tarkvara funktsionaalsusest, süsteemi praktilistest rakendustest ja selle potentsiaalsetest edasiarendusvõimalustest.

8.1 Testseadmete üldkirjeldus

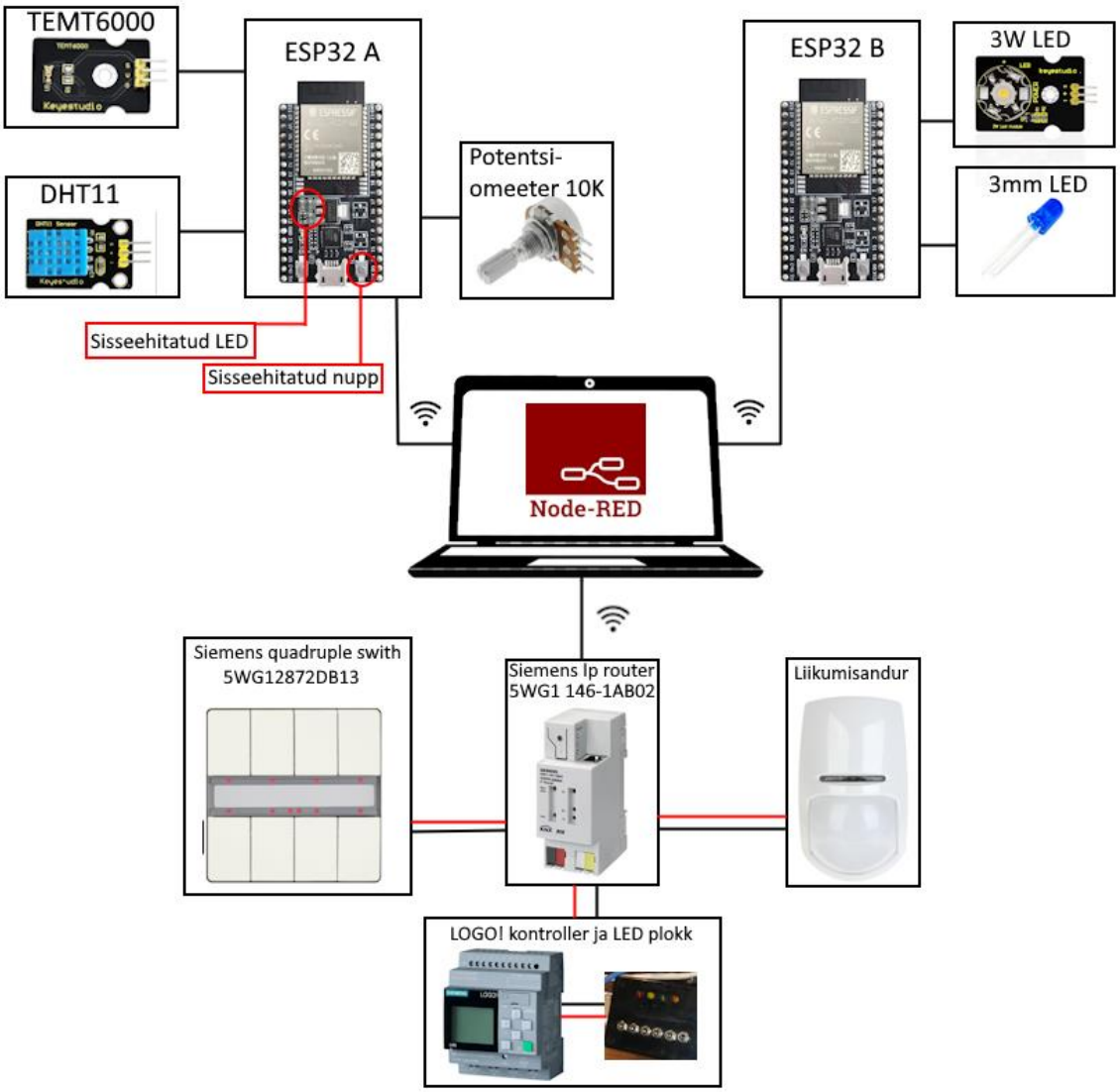
Projekti käigus arendati välja kaks IoT seadet ESP32 platvormil, nimetusega ESP32A ja ESP32B, mis omavahel suhtlevad CoAP-protokolli vahendusel. ESP32 mikrokontrolleri valik oli tingitud selle suurest funktsionaalsusest, sisseehitatud Wi-Fi ja Bluetooth ühendustest ning piisavast arvutusvõimekusest IoT ja KNX integratsiooniks [41]. Mõlemad seadmed on ühendatud Node-RED serveriga, võimaldades suhelda teiste KNX seadmetega (vt Joonis 15).

ESP32A seadmesse on integreeritud mitmed sensorid ja komponendid, sealhulgas:

- DHT11 temperatuuri- ja niiskusandur keskkonna jälgimiseks;
- TEMT6000 valgusandur valgusintensiivsuse mõõtmiseks;
- Potentsiomeeter, mille abil saadetakse analoogsignaale valgusintensiivsuse juhtimiseks.

ESP32B seadme põhiülesandeks on LED-valgustuse juhtimine PWM-signaali abil, pakkudes võimalust dünaamiliseks valgusintensiivsuse reguleerimiseks vastavalt saadud käskudele ja sensorite andmetele. LED-e saab juhtida nii käsitsi KNX nupuvajutuste abil kui ka automaatselt vastavalt sensorite väärtustele.

Joonisel 15 on KNX andmeside TP liin kujutatud kahe paralleelse joonega (punane ja must). Tavaline must joon koos WiFi sümboliga näitab juhtmevaba ühendust serveriga WiFi kaudu. Andurite ja LED-ide ühendus ESP32 mikrokontrolleritega on kujutatud ühe musta joonega.



Joonis 15. Seadmete ühendusskeem plokkdiagrammina

8.2 KNX ja CoAP integratsioon

Arendatud süsteem võimaldab kahepoolset andmevahetust KNX liini ja CoAP-põhiste ESP32 seadmete vahel. Näiteks KNX 8-nupulise lüliti abil saab juhtida nii ESP32A kui ka ESP32B seadmete sisseehitatud LED-e. Lisaks on ESP32A seadme füüsiline nupp konfigureeritud juhtima KNX süsteemis asuvat valgustit. Süsteemi kaudu on võimalik LED-e juhtida ka Node-RED dashboard'i kaudu. Ühte valgusallikat saab lülitada nii KNX TP liinilt kui ka IoT osast, näiteks ESP32 seadme nupust või Node-RED kasutajaliidesest.

Andmevahetuses kasutatakse CBOR andmeformaati. Kõik ESP32 seadmetest saadetavad ja vastu võetavad sõnumid on CBOR formaadis, mis lihtsustab nende töötlemist.

Lisaks juhitavale käsklusele edastatakse süsteemis ka seadmete hetkeseisundit. Näiteks kui LED süttib või kustub kas nupu vajutuse või liikumisanduri signaali tõttu, saadab vastav ESP32 seade oma uue seisundi CoAP kaudu Node-RED serverile. Node-RED platvorm saab selle kaudu reaalajas informatsiooni seadmete hetkeseisust ning saab seda *Dashboard*'il kasutajale kuvada. Kasutaja näeb näiteks kohe, kas tuli on hetkel sisse lülitatud või välja lülitatud, ilma et peaks päringut käsitsi saatma.

Selline lähenemine muudab süsteemi oluliselt kasutajasõbralikumaks ja tagab pideva andmesünkroonsuse kõikide KNX ja IoT võrgus osalevate seadmete vahel

8.3 Dimmeri funktsionaalsus

Dimmeri ehk hämardi funktsionaalsus on projekti oluline komponent, võimaldades valgusintensiivsuse dünaamilist muutmist. ESP32A seadmel on ühendatud potentsiomeeter, mis saadab analoogsignaali väärtuses 0–4095 Node-RED serverile. Serveris töödeldakse see signaal ning saadetakse ESP32B seadmele PWM juhtsignaalina, reguleerides LED valgusintensiivsust sujuvalt. Lisaks võimaldavad KNX liinil olevad nupud muuta sama LED-valgustuse heledust, pakkudes kasutajale mitmeid juhtimisvõimalusi.

PWM juhtimine ESP32B seadmes realiseeriti kasutades ESP32 LEDC draiverit, mis võimaldab täpset ja sujuvat dimmerdamist. Joonis 16 esitab PWM-signaali konfigureerimiseks kasutatud seaded.

```
// PWM-seadistus ESP32 LEDC draiveriga
ledc_timer_config_t tcfg = {
    .speed_mode      = LEDC_MODE,        // madalsagedusrežiim
    .timer_num       = LEDC_TIMER,       // valitud timer
    .duty_resolution = LEDC_DUTY_RES,    // 12-bitine täpsus
    .freq_hz         = LEDC_FREQUENCY,   // 5 kHz
    .clk_cfg         = LEDC_AUTO_CLK     // automaatne kell
};
ledc_timer_config(&tcfg);                // rakenda timer-seaded

// Kanal 0 ehk LED1 PWM väljund GPIO25 peal
ledc_channel_config_t ch1 = {
    .speed_mode = LEDC_MODE,
    .channel    = LED1_CHANNEL,
    .timer_sel  = LEDC_TIMER,
    .intr_type  = LEDC_INTR_DISABLE,
    .gpio_num   = LED1_PWM_GPIO,
    .duty       = 0,                      // algne tähistus 0
    .hpoint     = 0
};
ledc_channel_config(&ch1);               // rakenda kanal-seaded
```

Joonis 16. PWM juhtimiskoodi fragment

Ülaltoodud koodilõik näitab, kuidas määratakse PWM-signaali sagedus (5000 Hz) ja resolutsioon (12 bitti), võimaldades valguse sujuvat reguleerimist 0–4095 vahemikus. Selline täpsus tagab visuaalselt meeldiva ja stabiilse valguse muutumise, ilma märgatavate hüpeteta heleduses.

8.4 Sensoriandmete kogumine ja visualiseerimine

ESP32A seade kogub reaalajas keskkonna andmeid DHT11 temperatuuri- ja niiskusandurite ning TEMT6000 valgustusanduri abil. Need andmed edastatakse CoAP protokolli kaudu Node-RED platvormile, kus need dekodeeritakse ja visualiseeritakse *Dashboard*-il. Kasutaja saab jälgida temperatuuri, niiskuse ja valgusintensiivsuse muutumist reaalajas graafiliste diagrammide abil (Joonis 14). Lisaks kasutatakse TEMT6000 sensorilt saadavat valguse taseme informatsiooni, et automaatselt reguleerida ESP32B seadmel LED-i heledust vastavalt ümbritsevale valgusele, suurendades süsteemi autonoomsust ja energiatõhusust.

Joonis 17 esitab koodinäite spetsiaalsest perioodilisest ülesandest, mida kasutatakse andmete automaatseks edastamiseks.

```
// Perioodiline ülesanne CoAP-/env-ressursi observer’itele
// andmete saatmiseks
static void env_notify_task(void *pvParameters) {
    for (;;) {
        vTaskDelay(pdMS_TO_TICKS(10000));          // 10 sek wait
        coap_resource_notify_observers(env_resource, NULL);
    }
}

//serveri käivitamisel
xTaskCreate(env_notify_task, "env_notify", 2048, NULL, 5, NULL);
```

Joonis 17. Perioodiline andmete saatmine

Selline lahendus võimaldab temperatuuri, niiskuse ja valgustugevuse andmete automaatset saatmist Node-RED serverile iga määratud aja tagant (näiteks iga 10 sekundi järel), mis omakorda võimaldab reaalajas visualiseerimist ilma, et kasutaja peaks käsitsi päringuid saatma. See muudab süsteemi töö oluliselt mugavamaks ja tagab sujuva kasutajakogemuse.

8.5 Süsteemi praktiline rakendamine ja edasised arendusvõimalused

Katsetuste käigus näitas arendatud süsteem täpsust ja kasutusmugavust. Süsteemi abil õnnestus kombineerida traditsioonilise KNX protokolli tugevused ja kaasaegse IoT tehnoloogia paindlikkus. Kasutajakogemus oli sujuv ja intuitiivne, võimaldades süsteemi juhtimist erinevatest punktidest: KNX nuppudelt, ESP32 seadmete füüsilistest nuppudest ja Node-RED veebiliidesest.

Edasiste arenduste raames on võimalik lisada täiendavaid sensoreid (näiteks CO2, suitsu või liikumisandurid), täiustada turvafunktsioone (sh autentimine ja krüptitud side) ning ühendada süsteem pilvepõhiste teenustega nagu Home Assistant või Google Cloud IoT.

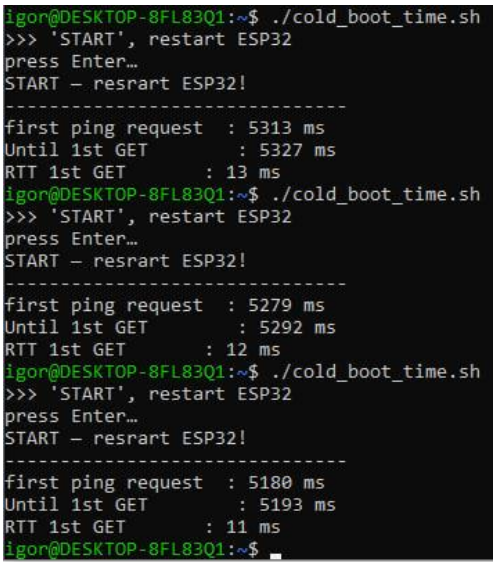
9 Testide läbiviimine ja tulemuste analüüs

Selles peatükis käsitletakse käesolevas magistritöös arendatud ESP32-põhise CoAP- ja OSCORE-protokolle rakendava seadme testimist. Testide peamiseks eesmärgiks oli hinnata süsteemi jõudlust erinevate võrguühenduse ja turbeprotokollide kasutamisel. Täpsemalt keskenduti reaktsiooniaja ehk *Round Trip Time*'i (RTT) mõõtmisele, seadme külmkäivituse ajale (*cold boot time*) ning protsessori koormuse analüüsile.

9.1 Testikeskkonna kirjeldus

Testid viidi läbi Windows Subsystem for Linux (WSL) keskkonnas, mis võimaldas kasutada Linuxi rakendusi Windowsi platvormil. Testimisel kasutati järgmisi tööriistu ja spetsiaalselt loodud skripte:

- libcoap – CoAP kliendi rakendus (*coap-client*), mida kasutati CoAP-päringute loomiseks ja analüüsimiseks.
- run_rtt_plain.sh ja run_rtt_oscore.sh – skriptid, mille abil mõõdeti seadme reaktsiooniaegu (RTT) CoAP- ja OSCORE-protokollide korral.
- cold_boot_time.sh – skript seadme külmkäivitusaegade mõõtmiseks, mis võimaldas hinnata seadme valmisolekut pärast taaskäivitamist (Joonis 18).
- poll_metrics.sh – skript protsessori koormuse ja süsteemi ressursikasutuse jälgimiseks ja salvestamiseks.



Joonis 18. Külmkäivitusaegade mõõtmise tulemused

9.2 Reaktsiooniaja mõõtmised

Reaktsiooniaja ehk RTT mõõtmised teostati CoAP- ja OSCORE-päringute puhul, kusjuures igale testile tehti suur arv päringuid, et saavutada statistiline usaldusväärsus. Tulemused on koondatud tabelisse 3.

Tabel 3. RTT mõõtmiste tulemused CoAP- ja OSCORE-protokollide puhul

Päringu tüüp	**Keskmine RTT (ms)**	**95%-ne RTT (ms)**	**Edukate päringute arv**
Plain GET	11,1	18,0	999/1000
Plain PUT	12,2	18,0	1000/1000
OSCORE GET	18,6	24,0	500/500
OSCORE PUT	23,9	25,0	500/500

Tulemused näitavad selgelt, et OSCORE protokolliga seotud turvameetmete kasutamine tõstab reaktsiooniaega võrreldes tavapäraste CoAP-päringutega. Selle põhjuseks on OSCORE'i lisanduv krüptograafiline töötlus.

9.3 Seadme külmkäivituse aja mõõtmised

Seadme külmkäivituse aja mõõtmise eesmärgiks oli hinnata, kui kiiresti ESP32 taastab funktsionaalse seisundi pärast täielikku taaskäivitamist. Test viidi läbi kolmekordselt ja tulemused on esitatud tabelis 4.

Tabel 4. ESP32 seadme külmkäivituse aja mõõtmised

Katse nr	**Esimese pingi aeg (ms)**	**Aeg esimese GET päringuni (ms)**	**RTT esimese päringu jaoks (ms)**
1	5313	5327	13
2	5279	5292	12
3	5180	5193	11

Tabelis toodud andmed kinnitavad seadme kiiret taaskäivitusvõimet, mille puhul esimene võrguühendus toimub ligikaudu 5 sekundi jooksul, mis sobib hästi reaalajas töötavate IoT rakenduste jaoks.

9.4 CPU koormuse analüüs

CPU koormust hinnati erinevates koormustingimustes kasutades poll_metrics.sh skripti, mis jälgis ja salvestas protsessori koormuse näitajaid. Mõõtmised toimusid seadme

jõudeolekus ning nii CoAP kui ka OSCORE päringute ajal. Tulemused on esitatud tabelis 5.

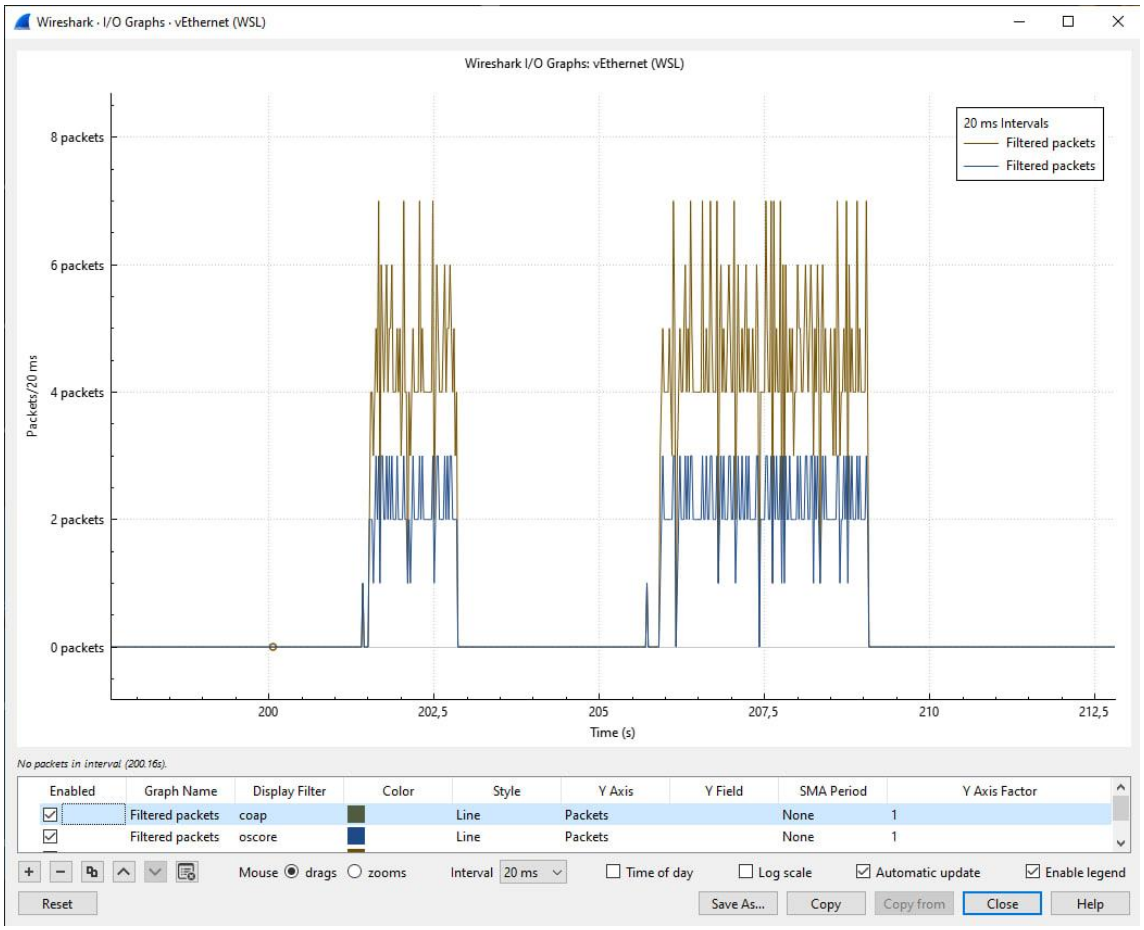
Tabel 5. CPU koormuse analüüsi tulemused

Töörežiim	**Keskmine CPU koormus (%)**	**95%-ne CPU koormus (%)**
Ilma koormuseta	1,8	2,3
Plain CoAP koormus	14,5	58,9
OSCORE koormus	20,2	78,5

Tabeli 5 andmetest ilmneb ühemõtteliselt, et OSCORE protokolli kasutamine suurendab CPU koormust võrreldes traditsioonilise CoAP protokolliga. CPU koormuse suurenemine tuleneb peamiselt protokollile omasest krüpteerimise ja dekrüpteerimise protsessist, mis nõuab täiendavaid arvutusressursse. Sellest tulenevalt on loogiline, et OSCORE rakendamine eeldab täiendavat riistvaralist jõudlust ning suuremat energiakulu võrreldes lihtsa, turvamata CoAP lahendusega. Samas on oluline rõhutada, et OSCORE-i suurenenud ressursikasutuse kõrval on selle peamiseks eeliseks ja õigustuseks turvalisuse oluline paranemine, mis muudab selle protokolli eriti sobivaks kasutamiseks turvakriitilistes rakendustes, nagu näiteks tööstuslikud juhtimissüsteemid või hooneautomaatika lahendused.

9.5 Võrguanalüüs Wireshark abil

Võrguanalüüsi läbiviimiseks kasutati Wireshark tarkvara, mis võimaldab detailset võrguliikluse analüüsi, sealhulgas CoAP- ja OSCORE-protokollide päringute ja vastuste struktuuri ning liiklusmustrite uurimist. Antud uurimuses keskenduti eelkõige pakettide edastusintensiivsuse ja andmevoogude erinevuste väljaselgitamisele protokollide vahel. Joonis 19 esitab Wireshark abil saadud võrguliikluse intensiivsuse graafik, mis illustreerib selgelt CoAP protokolli kõrgemat pakettide edastusaktiivsust võrreldes OSCORE protokolliga.



Joonis 19. Võrguliikluse graafik Wiresharkis

Graafikult nähtub, et CoAP protokolli kasutades toimub märgatavalt intensiivsem andmevahetus võrreldes OSCORE -ga. See kinnitab eelnevaid RTT ja CPU mõõtmiste tulemusi, näidates OSCORE-i suuremat ressursivajadust.

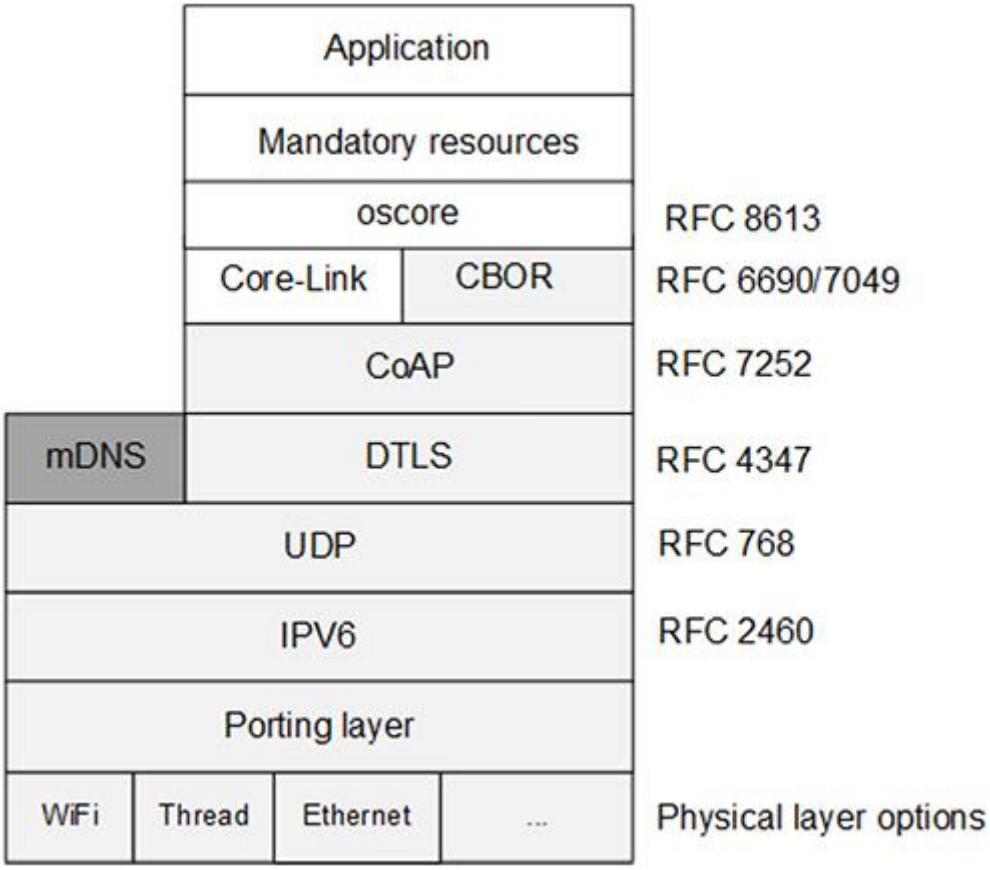
9.6 Testide kokkuvõte ja analüüs

Läbiviidud testide tulemused kinnitavad selgelt, et OSCORE protokoll esitab süsteemile suuremaid nõudeid nii arvutusvõimsuse kui ka energiatõhususe seisukohast. Siiski on selle kasutamine põhjendatud ja sageli vältimatu olukordades, kus süsteemi turvalisus ja andmete kaitse on ülioluline. Sellisteks valdkondadeks võivad olla näiteks automatiseeritud juhtimissüsteemid tööstuslikes rakendustes või hooneautomaatika, kus turvamata andmeside võib põhjustada tõsiseid turvariske ja süsteemirikkeid. Testide käigus kogutud andmed ja läbiviidud analüüs kinnitavad ühtlasi, et ESP32 platvorm on piisavalt võimas ja töökindel KNX IoT süsteemide praktiliseks realiseerimiseks, pakkudes head jõudlust ning stabiilsust ka nõudlikes rakendustes ja erinevates töötingimustes.

10 KNX IoT Point API seadme prototüüp

Üheks töö eesmärgiks oli luua lihtne, kuid võimalikult hästi nõuetele vastav KNX IoT seade, mis põhineks KNX IoT Point API spetsifikatsioonil (ver. 3.10.5) [42].

Point API määrab ära, milliseid protokolle, turvalisuse mehhanisme ja ressursikirjeldusi peab KNX IoT seade toetama, et saavutada koostalitlusvõime, grupisuhtlus ja turvaline sidumine teiste KNX-seadmetega IP-võrgus. Spetsifikatsioon viitab mitmele IETF standardile nagu CoAP, CBOR, OSCORE, SPAKE2+, mis kõik peavad olema rakendatud seadmes, mis järgib KNX IoT määratlust. Joonisel 20 on kujutatud põhikomponendid, mis on vajalikud KNX IoT seadme loomiseks.



Joonis 20. Stack-i komponendid [43]

Arendustöö hõlmas ka tutvumist KNX-i ametliku GitLabi [44] avatud lähtekoodiga, kus asuvad mitmed viiteteostused erinevatele operatsioonisüsteemidele – näiteks Linuxile, Windowsile või FreeRTOS-põhistele seadmetele. Kuigi need koodibaasid on põhjalikud ja standardsed, võib nende kasutamine praktikas osutuda keerukaks – projekt areneb pidevalt, kõik komponendid ei pruugi olla ajakohased ning mitmete moodulite

integreerimine võib olla keeruline, kuna need on kirjutatud spetsiifilisteks platvormideks või eesmärkideks. Siiski võib GitLabi projektist leida mitmeid kasulikke komponente, nagu *mbedtls* ja *tinycbor*, samuti FreeRTOS-i jaoks mõeldud *Makefile*'i. Kuigi need ei pruugi olla kohe kasutusvalmid, saab neid vajadusel kohandada ja kasutada arenduse alusena, pakkudes head lähtekohta süsteemi ülesehituseks.

10.1 Rakendatud funktsionaalsus ja vastavus KNX IoT nõuetele

Töö käigus arendatud tarkvara põhineb ESP-IDF arenduskeskkonnal ning konfigureerib ESP32 mikrokontrolleri ühenduma IPv6-võrguga Wi-Fi kaudu. Seade toimib CoAP-serverina, kasutades libcoap v4 teeki. Kaitstud ressurss */led_secure* võimaldab CBOR-vormingus PUT-päringute kaudu juhtida mikrokontrolleri GPIO-väljundit (LED), võimaldades kahepoolset suhtlust KNX võrgus.

Rakendus toetab OSCORE turvastandardit, mis tagab andmeside krüpteerimise CoAP-protokolli tasemel. OSCORE konfiguratsioon toimub staatiliselt, määrates käsitsi eelnevalt defineeritud *master_secret* ja osapoolte identifikaatorid (*sender*_id, *recipient*_id). Lisaks ühineb seade automaatselt IPv6 multisaategrupiga FF02::158, mis on defineeritud KNX IoT Point API spetsifikatsioonis kui kohustuslik komponent grupipõhiste sõnumite jaoks.

CoAP-ressurssidele on lisatud KNX-ile vastavad metaandmed: rt (*resource type*) atribuudina kasutatakse näiteks väärtust "knx.dpt.1.001" ning if (*interface type*) atribuudina "knx.p". Lisaks on rakendatud ressursid /oic/d, /oic/p ja /oic/res, mis esindavad seadme identiteeti, konfiguratsiooni ja ressursiloendit. Need põhinevad OCF tuumressursside spetsifikatsioonil ning on loetletud ka KNX IoT Point API kohustuslike komponentidena.

10.2 Teostamata osad ja võimalikud täiustused

Kuigi enamik KNX IoT Point API funktsionaalsusest on prototüüpis realiseeritud, jäid mõned funktsioonid töö raames rakendamata. Näiteks ei sisalda kood SPAKE2+ turvamehhanismi, mis on mõeldud seadme turvaliseks ühendamiseks võrku PIN-koodi põhise autentimise kaudu. Selle funktsiooni realiseerimine eeldaks dünaamilist

krüptograafilise konteksti vahetust ja keerukamat osapoolte autentimisprotsessi, mida antud prototüübis ei rakendatud.

Samuti jäid realiseerimata *Group Object Table* (GOT) ehk grupiaadresside sidumine ja .knxprod-faili genereerimine, mis on vajalik seadme integreerimiseks ametlikku ETS projekteerimiskeskkonda. Tasuta ETS-i versioon ei võimalda KNX IoT seadmete lisamist ning täisfunktsionaalsed kommertsversioonid maksavad alates 200 kuni 1000 eurot. Veelgi enam, KNX-i poolt pakutav The KNX Manufacturer Tool, mis on vajalik sertifitseeritud seadmete kirjeldamiseks ja registreerimiseks, maksab täiendavalt 2050 eurot. Seetõttu jäi vastav osa väljapoole käesoleva töö võimalusi [45].

10.3 KNX IoT 3rd Party API serveri võimalused

Lisaks KNX IoT Point API spetsifikatsiooni põhjal töötavatele seadmetele on KNX ühenduse loomiseks olemas ka alternatiivne lähenemine, kasutades *3rd Party API* serverit. Tegemist on serveripoolse lahendusega, mis toimib vahendajana klassikalise KNX paigalduse ja väliste IP-põhiste seadmete vahel. Selline server võimaldab REST-tüüpi HTTP-liidese kaudu ligipääsu KNX võrgus toimuvatele sündmustele ning nende juhtimist, ilma et klientseade peaks ise realiseerima kogu KNX protokollipakki. See lihtsustab selliste platvormide nagu näiteks ESP32 integreerimist KNX süsteemi.

KNX IoT 3rd Party API server põhineb semantilisel KNX Information Model'il ning eksporditud ETS projektide kirjel, kus määratletakse ruumid, funktsioonid ja grupiaadressid. Server on tavaliselt rakendatud Docker-konteinerina.Välised seadmed saavad lugeda või kirjutada KNX grupiadresse läbi serveri, kasutades standardiseeritud URL-põhist liidest. Server ise edastab vastavad CoAP või KNXnet/IP telegrammid füüsilisele KNX võrgule.

Serveri kasutamiseks tuleb alla laadida vastav docker-i fail, mis defineerib API konteineri ja PostgreSQL andmebaasi. Pärast konteinerite käivitamist muutub server kättesaadavaks läbi brauseri defineeritud aadressil. Kahjuks oli töö kirjutamise ajal varasem serveriversioon eemaldatud avalikust repositooriumist ning uus versioon arendamisel. Seetõttu ei olnud võimalik antud lähenemist täielikult katsetada ega rakendada töö raames [46].

11 Kokkuvõte

Käesoleva magistritöö põhieesmärk oli KNX IoT raamistikul põhineva testseadme arendamine ja sidumine KNX-võrkudega, uurides samal ajal KNX IoT spetsifikatsiooni praktilist rakendamist ja selle ühilduvust klassikalise KNX-infrastruktuuriga. Töö käigus tehti mitmeid tehnoloogilisi valikuid – platvormiks valiti ESP32 mikrokontroller ja visuaalseks arendus- ning ühendusliideseks Node-RED, mis võimaldas kiiret seadistamist ja süsteemi paindlikku laiendamist. Lisaks viidi läbi KNX ja KNX IoT standardite võrdlus teiste levinud hooneautomaatika protokollidega.

Oli realiseeritud süsteemi prototüüp ja selle keskseks tulemuseks oli kahepoolse kommunikatsiooni saavutamine reaalse KNX-liiniga, kasutades KNX/IP-ruuterit ja klassikalisi KNX-seadmeid. ESP32 platvormil arendati CoAP-server, mis võimaldas andurite ja täiturite juhtimist ning andmete kogumist ja edastamist läbi CBOR-andmeformaadi. Samuti implementeeriti ESP32 serveri tasemel ka OSCORE tugi, mis võimaldas turvalise CoAP-liikluse testimist piiratud ressurssidega seadmes.

Node-RED platvorm võimaldas CoAP- ja KNX-seadmete integreerimist ning kasutajaliidese loomist, mille kaudu sai süsteemi reaalajas jälgida ja juhtida. Edukalt realiseeriti näiteks LED-i juhtimine, sensoriandmete kogumine, dimmeri funktsionaalsus ja kasutajapõhine juhtimine läbi veebiliidese.

Töö käigus analüüsiti ka ametlikke KNX IoT lahendusi, nagu KNX IoT Point API ja 3rd Party API Server. Kuigi nende täielik praktiline rakendamine jäi piiratud ressurside tõttu teostamata, toob töö välja nende potentsiaali ja vajalikkuse professionaalsetes hooneautomaatikaprojektides.

Kokkuvõttes näitas töö, et ka piiratud ressurssidega platvormid võivad toetada KNX IoT spetsifikatsiooni põhifunktsioone ning sobivad hästi selle esmaste testide ja katsetuste läbiviimiseks. Edasine arendus võiks keskenduda kas prototüübi funktsionaalsuse laiendamisele või ametlike KNX IoT komponentide kasutuselevõtule professionaalsetes lahendustes.

Kasutatud kirjandus

[1] KNX association, „A brief introduction to KNX“. Vaadatud: 5. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/for-professionals/What-is-KNX/A-brief-introduction/

[2] KNX Association International, „A History of KNX“. Vaadatud: 5. veebruar 2025. [Võrgumaterjal]. Saadaval: https://crelectrics.com.au/wp-content/uploads/2015/05/a_history_of_KNX.pdf

[3] Neuron Team, „KNX Protocol: The Basics and Its Possibilities with IoT“. Vaadatud: 7. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.emqx.com/en/blog/knx-protocol

[4] „Introducing the knx standard in smart building (BMS)“, Vaadatud: 12. veebruar 2025. [Võrgumaterjal]. Saadaval: https://bmskaren.ir/knx-smart-home-en/introducing-the-knx-standard-in-smart-building-bms/

[5] Mark Warburton, „The New KNX TP1-256 Topology: more devices and fewer line repeaters“. Vaadatud: 12. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knxtoday.com/2019/07/13941/the-new-knx-tp1-256-topology-more-devices-and-fewer-line-repeaters.html

[6] KNX association, „KNX Cable : Detailed Specification Guide“. Vaadatud: 13. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knxhub.com/detailed-guide-to-knx-cable/

[7] KNX association, „KNX Basics“. Vaadatud: 13. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knx.fi/doc/esitteet/KNX-Basics_en.pdf

[8] KNX association, „The main features of ETS6“. Vaadatud: 15. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/newsroom/news/news/20210916-The-main-features-of-ETS6/

[9] KNX association, „The future of sustainable living: How KNX is leading the way“. Vaadatud: 15. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/newsroom/news/news/20240327-The-future-of-sustainable-living-How-KNX-is-leading-the-way/

[10] „KNX“. Vaadatud: 29. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.gvssmart.com/article/856.html

[11] KNX association, „The benefits of a distributed versus a centralized system“. Vaadatud: 21. veebruar 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/newsroom/news/news/20231030-The-benefits-of-a-distributed-versus-a-centralized-system/

[12] KNX association, „KNX and smart device manufacturers: The need for open standards in smart homes“. Vaadatud: 15. märts 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/newsroom/news/news/20250305-KNX-and-smart-device-manufacturers-The-need-for-open-standards-in-smart-homes/

[13] „KNX : Definition“. Vaadatud: 15. märts 2025. [Võrgumaterjal]. Saadaval: https://www.wattsense.com/resources/glossary/knx/

[14] Meghan Kelley, „The Difference Between BACnet, Modbus and LonWorks“. Vaadatud: 15. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.setra.com/blog/the-difference-between-bacnet-modbus-and-lonworks

[15] KNX association, „KNX and Matter: Position Paper“. Vaadatud: 15. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.knxtoday.com/2024/03/50281/knx-and-matter-position-paper.html

[16] Valerii Haidarzhy, „Zigbee vs. Z-Wave: Comparison, Pros & Cons, How Do They Work?“ Vaadatud: 1. aprill 2025. [Võrgumaterjal]. Saadaval: https://sirinsoftware.com/blog/zigbee-vs-z-wave-comparison-pros-cons-how-do-they-work

[17] Anand Ajith Kumar, „The Thread Protocol: Redefining IoT Connectivity for a Smarter World“, nov 2024, Vaadatud: 15. aprill 2025. [Võrgumaterjal]. Saadaval: https://thinkpalm.com/blogs/the_thread_protocol_redefining_iot_connectivity_for_a_smarter_world/

[18] Mikołaj Skawiński, „Bluetooth vs WiFi Comparison For the IoT Solutions“. Vaadatud: 15. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.netguru.com/blog/bluetooth-vs-wifi-comparison-for-the-iot-solutions

[19] „Zigbee FAQ“. Vaadatud: 16. aprill 2025. [Võrgumaterjal]. Saadaval: https://csa-iot.org/all-solutions/zigbee/zigbee-faq/

[20] „What is Z-Wave Long Range and How Does it Differ from Z-Wave?“ Vaadatud: 16. aprill 2025. [Võrgumaterjal]. Saadaval: https://z-wavealliance.org/what-is-z-wave-long-range-how-does-it-differ-from-z-wave/

[21] SILICON LABS, „Thread Fundamentals“. Vaadatud: 17. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.silabs.com/documents/public/user-guides/ug103-11-fundamentals-thread.pdf

[22] „Bluetooth Vs WiFi – What’s the difference?“, juuni 2024, Vaadatud: 17. aprill 2025. [Võrgumaterjal]. Saadaval: https://accsoon.com/explore/bluetooth-vs-wifi-whats-the-difference/

[23] KNX association, „KNX Technology Overview“. Vaadatud: 19. aprill 2025. [Võrgumaterjal]. Saadaval: https://radiocrafts.com/technologies/knx-technology-overview/

[24] „MQTT Essentials“. Vaadatud: 21. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.hivemq.com/mqtt/

[25] Z. Shelby, ARM, K. Hartke, ja C. Bormann, „The Constrained Application Protocol (CoAP)“. Vaadatud: 19. aprill 2025. [Võrgumaterjal]. Saadaval: https://datatracker.ietf.org/doc/html/rfc7252

[26] Ian Craggs, „MQTT Vs. HTTP for IoT“. Vaadatud: 15. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/

[27] Jesus Arias, „KNX IoT Roll-out 2024“. Vaadatud: 21. märts 2025. [Võrgumaterjal]. Saadaval: https://support.knx.org/hc/en-us/articles/11224992993426-KNX-IoT-Roll-out-2024

[28] Joost Demarest, „KNX Development including IoT: the full picture“. Vaadatud: 21. märts 2025. [Võrgumaterjal]. Saadaval: https://support.knx.org/hc/en-us/articles/4402060394770-KNX-Development-including-IoT-the-full-picture

[29] THREAD GROUP, „Thread 1.3.0 Features White Paper“. Vaadatud: 23. märts 2025. [Võrgumaterjal]. Saadaval: https://www.threadgroup.org/Portals/0/documents/support/Thread1.3.0WhitePaper_07192022_3990_1.pdf

[30] Jesus Arias, „KNX IoT API Server development: Implementation example for KNX PoC 2.x version“. Vaadatud: 27. märts 2025. [Võrgumaterjal]. Saadaval: https://support.knx.org/hc/en-us/articles/23995369446162-KNX-IoT-API-Server-development-Implementation-example-for-KNX-PoC-2-x-version

[31] KNX association, „KNX IoT“, Vaadatud: 24. märts 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/for-professionals/benefits/knx-iot/

[32] „CoAP“. Vaadatud: 29. märts 2025. [Võrgumaterjal]. Saadaval: https://coap.space/

[33] KNX association, „Highest security for smart buildings“. Vaadatud: 30. märts 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/newsroom/news/press/20240125-2-new-knx-security-check-tools/

[34] „Create your digital smart home with KNX Virtual“. Vaadatud: 31. märts 2025. [Võrgumaterjal]. Saadaval: https://www.knx.org/knx-en/for-professionals/get-started/knx-virtual/

[35] Christophe Parthoens, „Interface: connect to ETS“. Vaadatud: 3. aprill 2025. [Võrgumaterjal]. Saadaval: https://support.knx.org/hc/en-us/articles/4502392375314-Interface-connect-to-ETS

[36] KNX association, „KNX IoT Virtual“. Vaadatud: 8. märts 2025. [Võrgumaterjal]. Saadaval: https://www.knx-iotech.org/

[37] KNX association, „KNX-IOT-STACK“. Vaadatud: 8. märts 2025. [Võrgumaterjal]. Saadaval: https://github.com/KNX-IOT/KNX-IOT-STACK

[38] CASCODA, „KNX IoT“. Vaadatud: 26. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.cascoda.com/solutions/knxiot/

[39] Daniel Silva, Liliana I. Carvalho, José Soares, ja Rute C. Sofia, „A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA“, juuni 2021, Vaadatud: 2. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.mdpi.com/2076-3417/11/11/4879

[40] „Node-RED“. Vaadatud: 28. märts 2025. [Võrgumaterjal]. Saadaval: https://nodered.org/

[41] ESPRESSIF, „ESP32 Series“. Vaadatud: 16. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[42] KNX association, „KNX IoT Point API“. Vaadatud: 17. aprill 2025. [Võrgumaterjal]. Saadaval: https://support.knx.org/hc/en-us/articles/10386532582930-Downloads

[43] „KNX IoT: Part 7 – the open-source KNX IoT stack“. Vaadatud: 7. aprill 2025. [Võrgumaterjal]. Saadaval: https://www.knxtoday.com/2023/11/48537/knx-iot-part-7-the-open-source-knx-iot-stack.html

[44] KNX association, „knx-iot-point-api-stack“. Vaadatud: 23. aprill 2025. [Võrgumaterjal]. Saadaval: https://gitlab.knx.org/public-projects/knx-iot-point-api-stack/-/tree/work_in_progress?ref_type=heads

[45] Erind Dedja, „Licensing Options ETS6 & Price“. Vaadatud: 17. aprill 2025. [Võrgumaterjal]. Saadaval: https://support.knx.org/hc/en-us/articles/21546945829010-Licensing-Options-ETS6-Price

[46] Stefan Heinloth, „KNX IoT 3rd Party API Demo“. Vaadatud: 5. mai 2025. [Võrgumaterjal]. Saadaval: https://gitlab.knx.org/public-projects/knx-iot-3rd-party-api-demo/-/wikis/home/

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Igor Andrejev

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "KNX IoT raamistikul põhineva testseadme arendamine ja sidumine KNX-võrkudega", mille juhendaja on Andres Rähni
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

12.05.2025

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Tehisintelekti kasutamine töös

Generatiivseid tehisintellekti tööriistu (ChatGPT) kasutati selle töö koostamise käigus teksti tõlkimise, selguse ja loetavuse parandamise ning arendusprotsessi toetamise eesmärgil, sealhulgas sobivate koodistruktuuride, teekide ja sõltuvuste soovitamiseks.

Lisa 3 – Kood ESP A jaoks

```
// main/coap_server_for_ESP_A.c

#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "protocol_examples_common.h"
#include "driver/gpio.h"
#include "driver/adc.h"
#include "coap3/coap.h"
#include "cbor.h"
#include "dht.h" // DHT component for ESP-IDF

// === Pins and Constants ===
#define DHT_GPIO              18                  // DHT11 DATA pin
#define DHT_TYPE              DHT_TYPE_DHT11

#define LED_GPIO              2                   // built-in LED
#define BUTTON_GPIO           0                   // BOOT button (GPIO0)
#define DIMMER_ADC_CH         ADC1_CHANNEL_6      // GPIO34, potentiometer

#define LIGHTSENSOR_ADC_CH    ADC1_CHANNEL_7      // GPIO35, TEMT6000

// CBOR / text formats
#define CF_CBOR               60
#define CF_TEXT_PLAIN         COAP_MEDIATYPE_TEXT_PLAIN

static const char *TAG = "CoAP_CBOR";

// CoAP resources
static bool             led_on           = false;
static coap_resource_t *light_resource   = NULL;

static bool             toggle_state     = false;
static coap_resource_t *button_resource  = NULL;
static QueueHandle_t    button_queue     = NULL;

static int              dimmer_value     = 0;
static coap_resource_t *dimmer_resource  = NULL;

static coap_resource_t *env_resource     = NULL; // /env
```

```
static coap_resource_t *lux_resource     = NULL; // /lux

// ———————— GET /light → CBOR boolean led_on
static void hnd_light_get(coap_resource_t *r, coap_session_t *s,
                          const coap_pdu_t *req, const coap_string_t *q,
                          coap_pdu_t *resp) {
    uint8_t buf[8];
    CborEncoder enc;
    cbor_encoder_init(&enc, buf, sizeof(buf), 0);
    cbor_encode_boolean(&enc, led_on);
    size_t len = cbor_encoder_get_buffer_size(&enc, buf);

    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CONTENT);
    uint8_t opt[4];
    size_t ol = coap_encode_var_safe(opt, sizeof(opt), CF_CBOR);
    coap_add_option(resp, COAP_OPTION_CONTENT_FORMAT, ol, opt);
    coap_add_data(resp, len, buf);
}

// ———————— PUT /light
static void hnd_light_put(coap_resource_t *r, coap_session_t *s,
                          const coap_pdu_t *req, const coap_string_t *q,
                          coap_pdu_t *resp) {
    unsigned fmt = CF_TEXT_PLAIN;
    coap_opt_t *cf = coap_check_option(req, COAP_OPTION_CONTENT_FORMAT,
NULL);
    if (cf) fmt = coap_decode_var_bytes(coap_opt_value(cf),
coap_opt_length(cf));

    size_t size; const uint8_t *data;
    if (!coap_get_data(req, &size, &data) || size == 0) {
        coap_pdu_set_code(resp, COAP_RESPONSE_CODE_BAD_REQUEST);
        return;
    }

    bool new_state = led_on;
    if (fmt == CF_TEXT_PLAIN) {
        new_state = (data[0] == '1');
    } else if (fmt == CF_CBOR) {
        CborParser p; CborValue v;
        if (cbor_parser_init(data, size, 0, &p, &v) == CborNoError) {
            if (cbor_value_is_boolean(&v)) {
                cbor_value_get_boolean(&v, &new_state);
            } else if (cbor_value_is_integer(&v)) {
                int64_t x;
                cbor_value_get_int64(&v, &x);
                new_state = x != 0;
            }
        }
    } else {
```

```
        coap_pdu_set_code(resp,
COAP_RESPONSE_CODE_UNSUPPORTED_CONTENT_FORMAT);
        return;
    }

    if (new_state != led_on) {
        led_on = new_state;
        gpio_set_level(LED_GPIO, led_on);
    }
    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CHANGED);
    coap_resource_notify_observers(light_resource, NULL);
}

// ————————— DELETE /light
static void hnd_light_delete(coap_resource_t *r, coap_session_t *s,
                             const coap_pdu_t *req, const coap_string_t *q,
                             coap_pdu_t *resp) {
    led_on = false;
    gpio_set_level(LED_GPIO, 0);
    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_DELETED);
}

// ————————— GET /button
static void hnd_button_get(coap_resource_t *r, coap_session_t *s,
                           const coap_pdu_t *req, const coap_string_t *q,
                           coap_pdu_t *resp) {
    uint8_t buf[8]; CborEncoder enc;
    cbor_encoder_init(&enc, buf, sizeof(buf), 0);
    cbor_encode_boolean(&enc, toggle_state);
    size_t len = cbor_encoder_get_buffer_size(&enc, buf);

    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CONTENT);
    uint8_t opt[4]; size_t ol = coap_encode_var_safe(opt, 4, CF_CBOR);
    coap_add_option(resp, COAP_OPTION_CONTENT_FORMAT, ol, opt);
    coap_add_data(resp, len, buf);
}

// ————————— GET /dimmer
static void hnd_dimmer_get(coap_resource_t *r, coap_session_t *s,
                           const coap_pdu_t *req, const coap_string_t *q,
                           coap_pdu_t *resp) {
    uint8_t buf[8]; CborEncoder enc;
    cbor_encoder_init(&enc, buf, sizeof(buf), 0);
    cbor_encode_int(&enc, dimmer_value);
    size_t len = cbor_encoder_get_buffer_size(&enc, buf);

    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CONTENT);
    uint8_t opt[4]; size_t ol = coap_encode_var_safe(opt, 4, CF_CBOR);
    coap_add_option(resp, COAP_OPTION_CONTENT_FORMAT, ol, opt);
    coap_add_data(resp, len, buf);
}
```

```
// ————————— GET /env
static void hnd_env_get(coap_resource_t *r, coap_session_t *s,
                        const coap_pdu_t *req, const coap_string_t *q,
                        coap_pdu_t *resp) {
    float t = 0, h = 0;
    if (dht_read_float_data(DHT_TYPE, DHT_GPIO, &h, &t) != ESP_OK) {
        coap_pdu_set_code(resp, COAP_RESPONSE_CODE_INTERNAL_ERROR);
        return;
    }
    uint8_t buf[64]; CborEncoder enc, map;
    cbor_encoder_init(&enc, buf, sizeof(buf), 0);
    cbor_encoder_create_map(&enc, &map, 2);
    cbor_encode_text_stringz(&map, "temperature"); cbor_encode_double(&map,
t);
    cbor_encode_text_stringz(&map, "humidity");    cbor_encode_double(&map,
h);
    cbor_encoder_close_container(&enc, &map);
    size_t len = cbor_encoder_get_buffer_size(&enc, buf);

    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CONTENT);
    uint8_t opt[4]; size_t ol = coap_encode_var_safe(opt, 4, CF_CBOR);
    coap_add_option(resp, COAP_OPTION_CONTENT_FORMAT, ol, opt);
    coap_add_data(resp, len, buf);
}

// ————————— GET /lux (TEMT6000)
static void hnd_lux_get(coap_resource_t *r, coap_session_t *s,
                        const coap_pdu_t *req, const coap_string_t *q,
                        coap_pdu_t *resp) {
    int raw = adc1_get_raw(LIGHTSENSOR_ADC_CH);
    uint8_t buf[4]; CborEncoder enc;
    cbor_encoder_init(&enc, buf, sizeof(buf), 0);
    cbor_encode_int(&enc, raw);
    size_t len = cbor_encoder_get_buffer_size(&enc, buf);

    coap_pdu_set_code(resp, COAP_RESPONSE_CODE_CONTENT);
    uint8_t opt[4]; size_t ol = coap_encode_var_safe(opt, 4, CF_CBOR);
    coap_add_option(resp, COAP_OPTION_CONTENT_FORMAT, ol, opt);
    coap_add_data(resp, len, buf);
}

// ————————— Button ISR
static void IRAM_ATTR button_isr_handler(void *arg) {
    bool cur = (gpio_get_level(BUTTON_GPIO) == 0);
    BaseType_t hpw = pdFALSE;
    xQueueSendFromISR(button_queue, &cur, &hpw);
    if (hpw) portYIELD_FROM_ISR();
}

// ————————— Button Task
```

```
static void button_task(void *arg) {
    bool cur, last = false;
    for (;;) {
        if (xQueueReceive(button_queue, &cur, portMAX_DELAY)) {
            if (cur && !last) {
                toggle_state = !toggle_state;
                ESP_LOGI(TAG, "Button→%s", toggle_state ? "ON" : "OFF");
                coap_resource_notify_observers(button_resource, NULL);
            }
            last = cur;
        }
    }
}

// ————————— Potentiometer Task
static void dimmer_task(void *arg) {
    int last = -1;
    for (;;) {
        int raw = adc1_get_raw(DIMMER_ADC_CH);
        if (abs(raw - last) > 10) {
            last = raw;
            dimmer_value = raw;
            ESP_LOGI(TAG, "Dimmer raw=%d", raw);
            coap_resource_notify_observers(dimmer_resource, NULL);
        }
        vTaskDelay(pdMS_TO_TICKS(200));
    }
}

// ————————— Notifications /env
static void env_notify_task(void *pv) {
    for (;;) {
        vTaskDelay(pdMS_TO_TICKS(10000));
        coap_resource_notify_observers(env_resource, NULL);
    }
}

static void lux_notify_task(void *pv) {
    for (;;) {
        vTaskDelay(pdMS_TO_TICKS(5000)); // for example, every 5 seconds
        coap_resource_notify_observers(lux_resource, NULL);
    }
}

// ————————— Main CoAP Task
static void coap_server_task(void *p) {
    coap_set_log_level(LOG_INFO);
    coap_context_t *ctx = coap_new_context(NULL);
    if (!ctx) {
        ESP_LOGE(TAG, "No CoAP ctx");
        vTaskDelete(NULL);
```

```
    }

    coap_address_t a6, a4;
    coap_address_init(&a6);
    a6.addr.sin6.sin6_family = AF_INET6;
    a6.addr.sin6.sin6_port = htons(COAP_DEFAULT_PORT);
    coap_new_endpoint(ctx, &a6, COAP_PROTO_UDP);

    coap_address_init(&a4);
    a4.addr.sin.sin_family = AF_INET;
    a4.addr.sin.sin_port = htons(COAP_DEFAULT_PORT);
    a4.addr.sin.sin_addr.s_addr = INADDR_ANY;
    coap_new_endpoint(ctx, &a4, COAP_PROTO_UDP);

    // light
    light_resource = coap_resource_init(coap_make_str_const("light"), 0);
    coap_register_handler(light_resource, COAP_REQUEST_GET, hnd_light_get);
    coap_register_handler(light_resource, COAP_REQUEST_PUT, hnd_light_put);
    coap_register_handler(light_resource, COAP_REQUEST_DELETE,
hnd_light_delete);
    coap_resource_set_get_observable(light_resource, 1);
    coap_add_resource(ctx, light_resource);

    // button
    button_resource = coap_resource_init(coap_make_str_const("button"), 0);
    coap_register_handler(button_resource, COAP_REQUEST_GET, hnd_button_get);
    coap_resource_set_get_observable(button_resource, 1);
    coap_add_resource(ctx, button_resource);

    // dimmer
    dimmer_resource = coap_resource_init(coap_make_str_const("dimmer"), 0);
    coap_register_handler(dimmer_resource, COAP_REQUEST_GET, hnd_dimmer_get);
    coap_resource_set_get_observable(dimmer_resource, 1);
    coap_add_resource(ctx, dimmer_resource);

    // env
    env_resource = coap_resource_init(coap_make_str_const("env"), 0);
    coap_register_handler(env_resource, COAP_REQUEST_GET, hnd_env_get);
    coap_resource_set_get_observable(env_resource, 1);
    coap_add_resource(ctx, env_resource);
    xTaskCreate(env_notify_task, "env_notify", 2048, NULL, 5, NULL);

    // lux
    lux_resource = coap_resource_init(coap_make_str_const("lux"), 0);
    coap_register_handler(lux_resource, COAP_REQUEST_GET, hnd_lux_get);
    coap_resource_set_get_observable(lux_resource, 1);
    coap_add_resource(ctx, lux_resource);
    xTaskCreate(lux_notify_task, "lux_ntf", 2048, NULL, 5, NULL);

    // button setup
    button_queue = xQueueCreate(4, sizeof(bool));
```

```
    gpio_config_t bcfg = {
        .pin_bit_mask = 1ULL << BUTTON_GPIO,
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = GPIO_PULLUP_ENABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_ANYEDGE
    };
    gpio_config(&bcfg);
    gpio_install_isr_service(0);
    gpio_isr_handler_add(BUTTON_GPIO, button_isr_handler, NULL);
    xTaskCreate(button_task, "button_task", 2048, NULL, 5, NULL);

    // ADC setup
    adc1_config_width(ADC_WIDTH_BIT_12);
    adc1_config_channel_atten(DIMMER_ADC_CH, ADC_ATTEN_DB_11);
    adc1_config_channel_atten(LIGHTSENSOR_ADC_CH, ADC_ATTEN_DB_11);
    xTaskCreate(dimmer_task, "dimmer_task", 4096, NULL, 5, NULL);

    // CoAP loop
    while (1) {
        coap_io_process(ctx, 1000);
    }
    coap_free_context(ctx);
    coap_cleanup();
    vTaskDelete(NULL);
}

void app_main(void) {
    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    ESP_ERROR_CHECK(example_connect());
    coap_startup();

    // LED
    gpio_config_t lg = {
        .pin_bit_mask = 1ULL << LED_GPIO,
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE
    };
    gpio_config(&lg);
    gpio_set_level(LED_GPIO, 0);

    xTaskCreate(coap_server_task, "coap_server", 8 * 1024, NULL, 5, NULL);
}
```