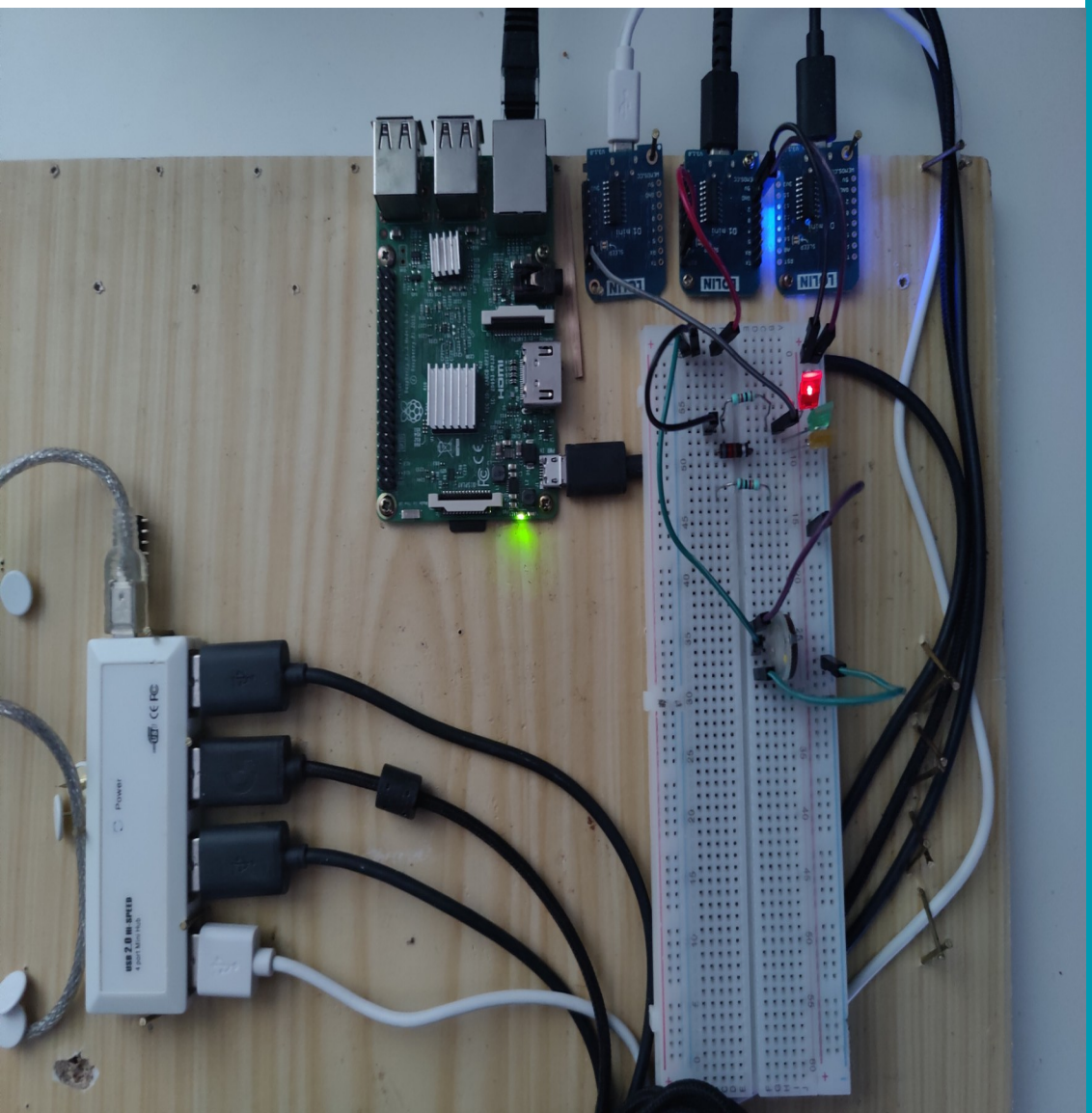
Samuli Piipponen

Datapalvelimen luominen mesh-verkkoon



Insinööri

Tieto- ja viestintätek-niikka

Kevät 2022

KAMK • University of Applied Sciences

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Piipponen Samuli

Työn nimi: Datapalvelimen luominen mesh-verkkoon

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: data, palvelin, mesh, verkko, synkronointi, tietoliikenne

Opinnäytetyö tehtiin KajaProlle käytettäväksi muihin projekteihin. Tarkoituksena oli tutkia, mitä haasteita ja mahdollisuuksia mesh-verkon rakentamisesta voisi yritykselle olla.

Opinnäytetyössä tutkittiin ensin hajautettujen verkkojen periaatetta teoriassa ja eri topologioiden eroja. Tämän jälkeen selvitettiin, mitä vaatii mesh-verkon rakentaminen ja pääpalvelimen pystyttäminen siihen. Samalla tutkittiin olemassa olevia ja mahdollisia mesh-teknologian käyttökohteita.

Työn mesh-verkko rakennettiin käyttämällä KajaPron hankkimia Arduino-pohjaisia piirilevyjä ja kyseiseen verkkoon liitettiin Raspberry Pi -palvelin. Lopputuloksena oli jokseenkin toimiva demonstraatioverkko, joka kuitenkin kaipaa vielä hiomista ennen oikeaa käyttöä.

Demonstraatioverkon rakentamisen aikana huomattiin, että vaikka mesh-verkot ovat melko joustavia yhteyksiltään, on palvelinta lisätessä otettava huomioon monia asioita, jotta yhteydet pääpalvelimeen muodostuvat dynaamisesti, sulavasti ja luotettavasti. Demonstraatioverkon päälle voidaan nykyisessäkin tilassa jo alkaa rakentaa toiminnallisuuksia. Demonstraatioverkon perusteella jatkokehitettävää olisi varsinkin yhteyksien asynkronisuudessa ja kirjastojen koodin tuomisessa lähemmäksi rautaa parempaa yhteydenhallintaa varten.

KAMK • University of Applied Sciences

Abstract

Author(s): Piipponen Samuli

Title of the Publication: Creating data server in mesh network

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: data, server, mesh, network, syncronization, telecommunications

This Bachelor's thesis was done for KajaPro for utilization in other projects. The purpose was to research the possible disadvantages and opportunities building a mesh network would give to the company.

At the beginning, the thesis goes through the research of the main ideas of and the differences between distributed and decentralized networks. After this, an attempt was made to figure out what building the mesh network and attaching the server to it actually requires. At the side, the existing and possible usage cases of mesh networks were considered.

The network of this work was built using Arduino based circuit boards KajaPro had acquired. A Raspberry Pi server was then attached to this network. The work resulted in rather functional demonstration network, which still needs some adjusting before it can be used.

During the building phase of the network, it occurred that even though mesh networks are quite flexible in terms of connectivity, there are many things to concider so that the connections to the main server can be made dynamically, smoothly and reliably. The demo network is now at the state that it can be used to build some sort of functionality on top of it already. Based on the experience with the demonstration network, the asynchronous behaviour of the connections must still to be further developed and the code of the now used high level libraries should be brought closer to the hardware for better control of the connections.

Sisällys

Symboliluettelo

WiFi – Myös Wi-Fi. Langaton lähiverkko.

Solmu – Yksittäinen mesh-verkon laite. Englanniksi node.

IEEE - The Institute of Electrical and Electronics Engineers. Kansainvälinen tekniikan alan standardeja kehittävä järjestö.

IoT – Internet of Things, suomeksi asioiden internet. Nykypäivän älylaitteet vaativat internet yhteyden ja muodostavat kokonaisuudessaan niin sanotun asioiden internetin.

Topologia – Verkon fyysinen rakenne.

JSON – JavaScript Object Notation, kielestä riippumaton, avoimen standardin tiedostomuoto tiedonvälitykseen.

IP – Internet Protocol. Määrittää, miten dataa siirretään koneelta toiselle.

HTTP – Hypertext Transfer Protocol. Protokolla, jota käytetään internetsisällön tiedonsiirtoon.

TCP – Transmission Control Protocol. Toimii yhdessä IP-protokollan kanssa määrittäen, kuinka paketteja siirretään.

1 Johdanto

Asioiden internet on tuonut mukanaan perinteiseen langattomaan viestintään uudemman hajautetun verkon ja etenkin näiden alle kuuluvan mesh-periaatteen, jota voidaan hyödyntää verkkojen laajentamiseen ja niiden luotettavuuden parantamiseen. Mesh-teknologiaa voidaan käyttää erityyppisissä verkoissa ja niiden sovellutuksissa. Tapoja luoda mesh-verkko on useita ja niillä on erilaiset päämäärät ja ongelmat. Monikäyttöisyys ja joustavuus tekevät mesh-verkosta kiinnostavan kohteen tulevaisuuden sovellutuksiin. Työn tarkoitus on löytää parhaat lähestymistavat datan synkronointiin yksittäiselle palvelimelle laajassakin verkossa.

Tässä työssä tutustutaan hajautettujen verkkojen perusteisiin ja selvitetään niiden erot perinteiseen verkkoon nähden. Yleisen hajautettujen verkkojen tarkastelun pohjustamana siirrytään käsittelemään tarkemmin mesh-verkon ominaisuuksia etenkin langattomassa käytössä. Lisäksi tarkastellaan mesh-teknologian olemassa olevia ja mahdollisia käyttökohteita, sekä hyötyjä ja hidasteita, jotka tulisi ottaa huomioon demonstraatioverkkoa rakentaessa.

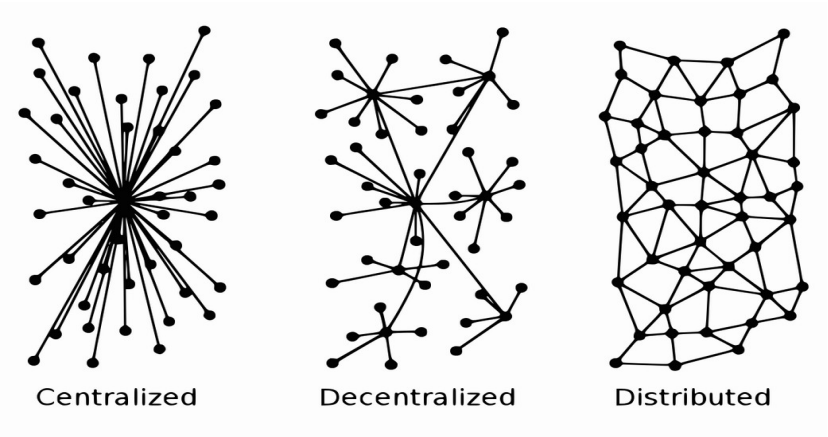
Työ on tehty toimeksiantona KajaProlle, tarkoituksena luoda muissa projekteissa käytettävä verkkototeutus ja pääpalvelin. Työhön kuuluu demonstraatioverkon rakentaminen, minkä avulla selvitetään vaatimukset tiedon luotettavaan synkronoimiseen mesh-verkon läpi ja tallentamiseen verkon sisällä toimivalle pääpalvelimelle. Demoverkon luomisen aikana testataan muutamia lähestymistapoja mesh-verkon tapahtumien tiedottamiseen pääpalvelimelle ja yhteyksien ylläpitämiseen. Tavoitteena on saada aikaan dynaamisesti muodostuvat ja luotettavat yhteydet.

Lopussa pohditaan kokonaisuutta siitä, mitkä asiat on tärkeää ottaa huomioon laitteistossa ja kehitysmenetelmissä, ennen kuin aletaan rakentaa verkkoa suuremmassa mittakaavassa. Rakennettu demonstraatioverkko antaa suuntaa, miten dataa voidaan synkronoida, mutta verkko on vielä kaukana valmiista. Demonstraatioverkkoa tarkastellaan erityisesti yhteyksien luotettavuuden kannalta. Samalla käydään läpi demonstraatioverkon rakentamisesta opittua käytäntöä ja jatkokehitystä tarvitsevia ideoita. Lähestymistapo-

jen joukosta verrataan mitkä menetelmistä toimivat ja mitkä tuottavat haasteita.

2 Tietoverkkojen peruskäsitteitä

Perinteisen keskitetyn verkkorakenteen rinnalle on noussut kaksi merkittävää modernimpaa rakennetta. Kuvassa 1 havainnollistetaan kolmen verkon yleistä profiilia. Ensimmäinen on perinteinen, keskitetty verkko. Decentralized-verkkoa kutsutaan suomeksi ei-keskitetyksi verkoksi ja distributed-verkko tunnetaan hajautettuna verkkona. Näitä verkkoja voidaan myös rakentaa eri verkkotopologioilla ja jopa yhdistää muuntyyppisiin verkkoihin [1], joten nämä kaksi ovat yleiskäsityksiä.

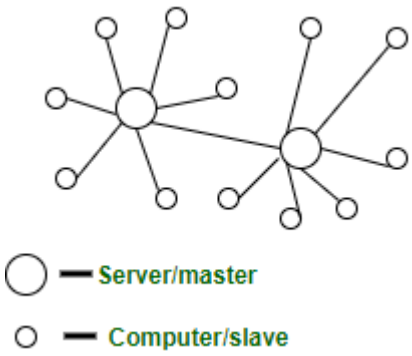


Kuva 1. Erityyppisiä verkkoja [2]

Yksittäistä verkon laitetta kutsutaan solmuksi. Solmut ovat verkkoon liitettyjä päätelaitteita kuten tietokoneita, mobiili- ja IoT-laitteita tai tulostimia [3]. Ei-keskitetyssä ja hajautetussa verkoissa solmut toimivat myös tukiasemina [3] muodostaen niin sanotun vertaisverkon [4]. Samalla jokainen solmu saattaa hoitaa itsenäisesti omat laskelmansa [3]. Tätä kutsutaan hajautetuksi laskennaksi. Hajautettu laskenta tarkoittaa usealle koneelle annettuja laskutehtäviä, jotka yhdessä muodostavan suuremman kokonaisuuden [5]. Näin saadaan laskettua nopeammin paljon laskentatehoa vaativia laskutoimituksia.

3 Hajauttamisen pääperiaate

On tärkeää kohdella ei-keskitettyä ja hajautettua verkkoa omina rakenteinaan, sillä ei-keskitetty verkko sisältää muutamia pääpalvelimia, kuten kuvan 2 verkossa, kun taas hajautettu verkko kykenee rakentumaan täysin tukilaitteidensa varaan ja olemaan kokonaan palvelimeton [6]. Kuten tässä työssä tullaan huomaamaan, hajautettuun verkkoon voidaan lisätä palvelimia. Se ei kuitenkaan vaadi niitä toimiakseen [6]. Molemmille verkoille on kuitenkin tyypillistä, että data kopioidaan useaan paikkaan [2]. Hajautetun verkon pääominaisuus on, että jos yksittäinen solmu katoaa verkosta, se ei katkaise samalla muita yhteyksiä, vaan solmut mukautuvat tilanteeseen ja yhdistyvät toisiinsa uutta reittiä pitkin [7, 8].



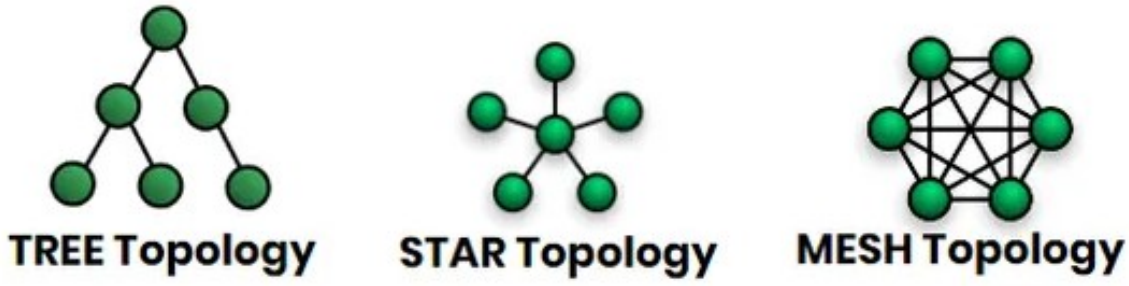
Kuva 2. Ei-keskitetty-verkko [3]

Hajautettu verkko on ei-keskitettyä verkkoa monipuolisempi ja se voidaan rakentaa eri tavoin. Hajautettuun verkkoon tarvitaan kuitenkin sitä tukeva verkon fyysinen infrastruktuuri [9], joten verkko pitää valmistella jo rakennusvaiheessa niin, että sen laitteet kykenevät yhdistymään toisiinsa automaattisesti ja jakamaan dataa ympäriinsä.

Hajautetussa verkossa on monta tukiasemaa tai -pistettä, jotka toimivat yhdessä toistensa kanssa siten, että kukin voi lähettää, muokata ja prosessoida verkossa liikkuvaa dataa itsenäisesti [10]. Riippuen verkon tarpeista ja laajuudesta, voivat myös verkon päätelaitteet toimia tukiasemana muille [11]. Näin verkkoja voidaan käyttää tiedon jakamiseen useampaa reittiä pitkin [9] ja laajentaa verkon kantamaa [7]. Tiedon jakaminen moneen paikkaan ja

yhteyksien määrä takaavat vakaat verkot, jotka pystyvät toimimaan rasituksen alla [11].

Hajautetun verkon alemman tason solmut voidaan kytkeä toisiinsa eri topologioilla. Erilaisista topologioista kolme tunnettua ovat tähti-, puu- ja mesh-topologia, jotka on havainnollistettu kuvassa 3. [12]. Tähtitopologiaa käytetään esimerkiksi kryptovaluuttojen lohkoketjuissa, jota voidaan jakaa ihmisille perinteisen verkon kautta. [9]. Data liikkuu käyttäjän ja yhden verkon palvelimista välillä ja palvelimet synkronoivat datan toisilleen [11]. Mesh-topologia sopii monen tyyppiseen, myös suuremman mittakaavan käyttötarkoitukseen monipuolisuutensa ja skaalautuvuutensa ansiosta [13]. Mesh-topologiassa kaikki yhteyttä jatkavat solmut yhdistyvät toisiinsa [14].



Kuva 3. Erilaisia topologioita [14]

3.1 Ero perinteiseen verkkoon

Keskitetty verkko koostuu yksittäisistä päätepisteistä, joihin laitteet ottavat yhteyden. Lähiverkkoon on yleensä liitetty reititin tai langattoman verkon tukiasema, jonka kautta kaikki tieto kulkee internetiin. [3, 7]. Maailmanlaajuisessa mittakaavassa päätepisteet ovat palvelimia, joista käyttäjä hakee varastoitua dataa [3]. Sulautetut järjestelmät saattavat jakaa prosessoitua tietoa yksittäisen palvelimen kanssa, jos ovat ollenkaan kytkettynä internetiin.

Myös hajautettu verkko voi sisältää näitä itsenäisiä päätepisteitä, mutta sen ei aina tarvitse, sillä laitteet ohjaavat dataa toistensa kautta ja korvaavat siten päätepisteen aseman [7]. Hajautetuilla verkoilla on monia hyviä puolia perinteiseen verkkoon nähden. Koska dataa haetaan monesta lähteestä, yhden pisteen rasitus mesh-verkossa ei ole niin suuri kuin perinteisessä verkos-

sa [9, 10]. Tämä näkyy suoraan verkon nopeuden kasvuna [7, 11]. Keskitetty verkko voi yhden tukiaseman periaattensa takia kaatua kokonaan, mikäli tukiasema jostain syystä sammuu tai ei yhdisty [3].

Hajautetussa tiedon ylläpitämisessä voi kuitenkin olla myös haittapuolensa. Koska kaikki voivat jakaa dataa, pitää luottaa, ettei yksikään osapuoli ole muokannut dataa väärin tai omien etujensa mukaisesti [9]. Tämä riippuu täysin verkon käyttökohteesta ja tiedon tyypistä, eikä välttämättä ole ongelma pienessä mittakaavassa ja suljetussa ympäristössä, esimerkiksi yrityksen sisällä, mutta voi nousta ongelmaksi julkisessa datanvaihdossa. Kryptovaluuttojen on pitänyt keksiä erillinen tapa todistaa datan todenmukaisuus, koska kryptovaluutan liikevaihdossa oikea raha vaihtaa omistajaa [9].

3.2 Hajautettu laskenta

Varsinkin IoT-laitteiden ympärillä hajautettu verkko suunnitellaan yleensä niin, että sen solmut laskevat yhdessä pienempiä osioita jostain suuremmasta kohteesta [15]. Näin suuressa verkossa voidaan käyttää paljon laskentatehoa laskennan nopeuttamiseksi, kun jokainen laite osallistuu laskentaan [16]. Tämä monimutkaistaa hieman laitteiden hallintaa, mutta paljon laskentatehoa vaativissa sovellutuksissa tämä on kannattavaa [15]. Toisin, kuin prosessorin säikeiden avulla tehdyssä rinnakkaislaskennassa, laskentatehoa voidaan helposti kasvattaa verkon mukana, koska jokainen uusi laite voi tulla mukaan laskentaan [17]. Koon kasvaminen kuitenkin johtaa useampiin muuttujiin verkossa ja siten taas verkon rakenteen monimutkaistumiseen [17].

Hajautetun laskennan pääohjelma lähettää laskijakoneille tehtäväpaketteja, joita koneet laskevat itsenäisesti ja lähettävät vastaukset takaisin pääohjelmalle. Tämän vuoksi ohjelmiston pitää olla rakennettu niin, että se kykenee jakamaan tehtävät moneen osaan, mikä voi olla hankalaa toteuttaa. [17.]

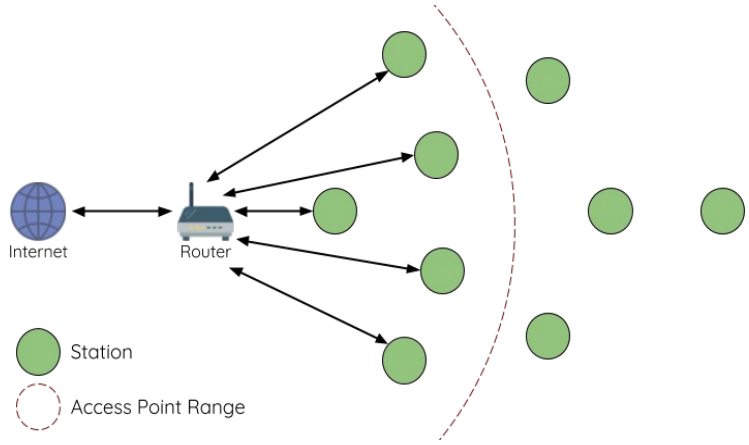
Esimerkiksi aiemmin mainittu kryptovaluuttojen tapa varmistaa lohkoketjun autenttisuus perustuu hajautettuun laskentaan, jossa jokaiselle koneelle annetaan hash-funktio laskettavaksi [16]. Myös Folding@home-projekti kokosi

hajautetusti yli eksaflopin [18] verran laskentatehoa ihmisiltä COVID-19 käyttäytymisen mallintamiseen [18, 19]. Laskennasta saatua tietoa voitiin käyttää rokotteiden kehittämiseen [19].

4 Mesh-verkko

4.1 Pääperiaate

Yksittäisellä tukiasemalla on oma rajallinen kantamansa. Kuvan 4 mukaisesti lähiverkon tukiaseman kantaman ulkopuolelle jääneet laitteet eivät kykene ottamaan yhteyttä verkkoon. Verkon kantamaan vaikuttaa signaalin taajuus ja erityisesti se, millaisia esteitä laitteen ja tukiaseman välissä on. Erityisesti kiviset materiaalit, kuten tiili ja betoni heikentävät signaalia huomattavasti. [20.]

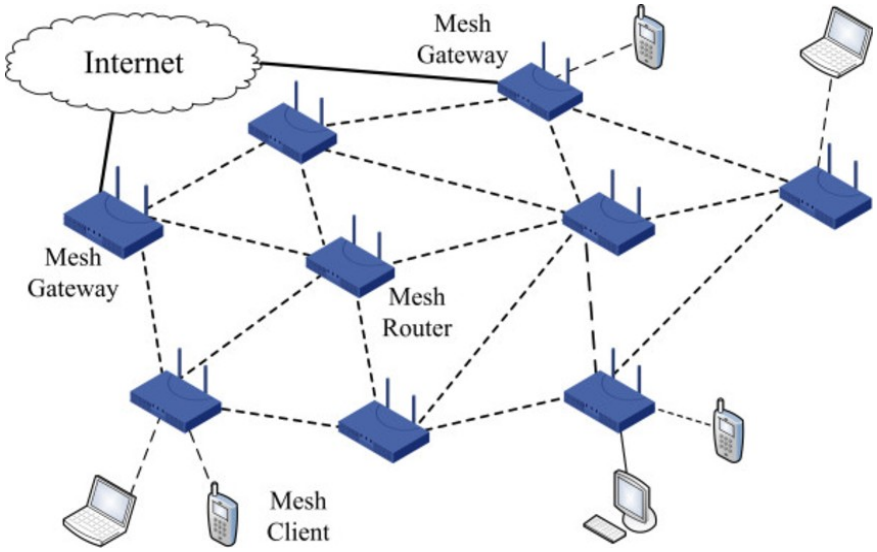


Kuva 4. Perinteisen langattoman verkon rakenne [8]

Mesh-verkko on hajautettujen verkkojen periaatteeseen pohjautuva verkon protokolla ja topologia [1]. Google kertoo Google Nest -ohjesivullaan, että ”Google Nest Wifin ja Google Wifin mesh-järjestelmillä saat kotiisi useita Wi-fi-pisteitä, jolloin jokin niistä on aina lähellä” [7]. Laitteen ei kuitenkaan tarvitse olla pelkkä tukiasema. Koska IoT-laitteissa on jo WiFi-kortti, tukiaseman ominaisuudet voidaan sijoittaa IoT-laitteen muun toiminnallisuuden rinnalle. Näin laitteet muodostavat itsenäisen, laajan alueen kattavan mesh-verkon [21]. Mesh-teknologia mahdollistaa laitteiden lisääminen verkkoon helposti ja mahdollisimman laajalle alueelle, kun laitteet yhdistyvät toisiinsa ja kykenevät siten jakamaan dataa kaikille verkon laitteille [1].

Mesh-verkot eivät välttämättä rakennu mesh-topologian varaan, vaan voivat muodostua käyttäen muita verkon rakenteita. Mesh-verkko voi olla joko ko-

konaan fyysisesti mesh-topologiaan pohjautuva, tai vain loogisesti mesh-toiminnallisuutta vastaava osittainen, jonkin toisen topologian päälle rakennettu verkko [1]. Esimerkiksi Espressif-kirjasto muodostaa solmuista puutopologian mukaisia väyliä, joiden välillä solmut eivät keskustele suoraan keskenään [8]. PainlessMesh-kirjasto taas käyttää tähtiverkon kaltaista rakennetta [22]. Kokonaan mesh-topolgian varaan rakennettu verkko olisi kuvan 5 mukaisesti hajautetun verkon kaltainen, jolloin solmut etsivät kaikki lähimmät ympäröivät solmut ja keskustelevat niiden kanssa. Kuitenkin tärkeintä on, että solmut muodostavat yhteyden dynaamisesti ja mukautuvat eri tilanteisiin ja nämä esimerkkikirjastot tekevät juuri sen. Mikään ei myöskään estä yhdistelemästä erityyppisiä verkkoja keskenään. Mesh-verkko voi hyvin olla ainoastaan osa suurempaa verkkoa [1].



Kuva 5. Mesh-topologian varaan rakennettu verkko [23]

Koska verkon laitteet ovat yhteydessä yhden päätukiaseman sijaan toisiinsa, on mesh-verkko luotettavampi kuin perinteinen verkko. Jos yksittäinen solmu poistuu verkosta, toiset solmut yksinkertaisesti vain etsivät uuden reitin. [8.] Yksi KajaPron käyttökohde-ehdotuksista on kulunvalvonta. Verkon yhteyksien luotettavuus on tämänkaltaiselle käyttökohteelle ehdoton ominaisuus.

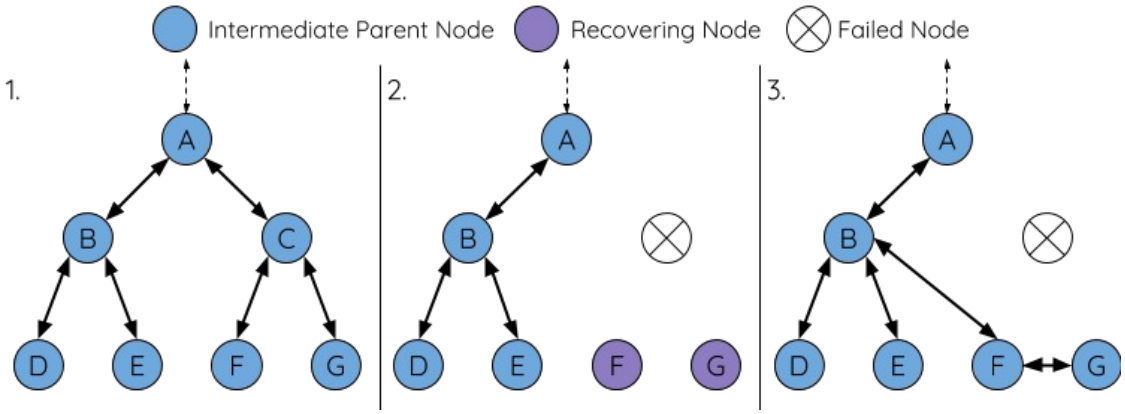
Riippuen siitä, minkä topologian päälle mesh-verkko on rakennettu, voi useamman solmun kautta kulkevan tiedon siirtämiseen tulla jonkin verran enemmän viivettä suoraan yhteyteen verrattuna. [1.] Se, haittaako tämä yli-

määräinen viive, riippuu tietenkin täysin käyttötarkoituksen tarpeista ja verkon päämääristä. Jotkut käyttötarkoitukset voivat asettaa luotettavuuden nopeuksien edelle ja viiveettömyys on uhrattavissa oleva ominaisuus. Esimerkiksi tämän työn verkossa viive ei haittaa toimivuutta mitenkään.

4.2 Yhteyksien muodostaminen langattomasti

Mesh-verkon voisi rakentaa myös langallisena viemällä datakaapelin jokaisesta solmusta jokaiseen ympäröivään solmuun. Esimerkiksi IEEE:n 802.1 aq -standardin mukainen lyhimmän polun silloittaminen on periaatteeltaan langallinen meshin-kaltainen verkkototeutus [24]. On kuitenkin paljon tarkoituksenmukaisempaa luoda se langattomasti. Langaton mesh-verkko on helppo asentaa ja ylläpitää: Riittää vain, että laite saa virtaa ja yhdistyy muuhun verkkoon WiFi-piirin kautta. Lisäksi se tulee halvemmaksi, koska riittää, että laitteissa on WiFi-kortti usean verkkoportin sijaan, kuten tässä työssä käytetyssä kortissa.

Uusien solmujen automaattinen yhdistyminen on osa langattoman mesh-verkon periaatetta [1, 8]. Yhteyksien valinnan tulee perustua mahdollisimman hyvään yhteydenlaatuun. Espressif-kirjaston sopivien solmujen yhteyksien valintaprosessi on etsiä ensin lähellä olevat solmut, joista valitaan haluttu määrä yhteyden heikkenemisen mukaan [8]. Mitä vähemmän yhteys heikkenee matkalla, sitä parempi solmuehdokas on. Tämän johtaa siihen, että saadaan mahdollisimman tiheä ja tasainen verkko. Ohjelmiston täytyy osata järjestää solmut uudelleen kuvan 6 mukaisella tavalla, mikäli väliin ilmestyy uusi laite tai jokin solmu katoaa. [8.]



Kuva 6. Uudelleenreititys C-solmun kadottua [8]

Langattomasti mesh-verkon yhdistämiseen on lähiverkon lisäksi muitakin tapoja. Bluetooth-protokolla kykenee muodostamaan mesh-verkon [25] ja myös tavallisia radiolähettimiä voi käyttää [26].

4.3 Käyttökohteita

Mesh-teknologian avulla saadaan siis aikaan luotettavampi ja laajempi verkko. Koska koteihin on nykyään tarjolla kaikenlaisia älylaitteita, jotka tarvitsevat internet-yhteyden tai ainakin yhteyden muihin laitteisiin integroituakseen osaksi kotia, tarvitaan nopeita ja luotettavia yhteyksiä. Tämän vuoksi mesh-verkko ja sen variantit kehittyivät. [27.] Kodin älylaitteet, esimerkiksi älyvalaisimet, käyttävät IEEE® 802.15.4 -standardin päällä toimivaa Zigbee-toteutusta. Yksi Zigbee-protokollan yhteydenmuodostustavoista on mesh-verkko. [28.]

IoT-laitteiden lisäksi mesh-periaate on otettu käyttöön kuudennen sukupolven kodin langattoman lähiverkon tukiasemissa. Uudemmat tukiasemat käyttävät mesh-periaatetta kantaman jatkamiseen koko talon läpi ilman, että tarvitsee muodostaa ylimääräisiä verkkoja. Näistä esimerkkinä toimivat Google Nest -älytuotteet [7].

Kun mietitään mahdollisia käyttökohteita, äkkiseltään mieleen juolahtaa, että periaatteessa minkä tahansa valvontajärjestelmän, koostui se sitten sensoridatasta tai valvontakameroista, voisi liittää osaksi mesh-verkkoa. Tämä edel-

lyttää äärimmäisen luotettavia yhteyksiä, mutta kaiken toimiessa oikein, olisi suuren alueen valvontakamerajärjestelmä halvempi rakentaa, kun data kulkisi itse laitteiden kautta.

Vaikka suurin osa mesh-verkoista on kooltaan juurikin kodin tai muun vastaavan pienehkön alueen kattavia, mesh-teknologian on skaalattavissa paljon suuremman mittakaavan verkoksi. Vuonna 2003 PanOULU-hanke toi Ouluun mesh-verkon [29], joka kattaa suurimman osan kaupungista [30]. PanOULUn verkko on loppukäyttäjälle täysin tavallinen avoin, ilmainen verkko. PanOULU on esimerkki siitä, kuinka mesh-verkko pystyy ylläpitämään yhteyksiä silloinkin, kun muut verkot pettävät. Yle uutisoi 2022 alkuvuodesta, kuinka verkot kaatuivat olympialaisten katsojien määrän vuoksi ja Oulussa kehotettiin ”käyttämään mobiiliverkon jakoa tai avointa ja langatonta PanOulu-verkkoa laiteyhteyksissä, ne toimivat normaalisti” [31]. Vastaavanlainen kaupungin kattava verkko löytyy myös Oklahomasta [32].

Samantapaista, koko kaupungin kattavaa verkkoa voisi ajatella varsinkin alueille, jossa vaihtoehtoja ei juuri ole. Ollessani työharjoittelussa Espanjan Furteventura-saarella asensimme WiMAX-antenneja kohteisiin. WiMAX on saaren pääasiallinen yhteys internetiin, sillä kiviperäiseen maastoon ei voi kaivaa kaapelia. WiMAX-yhteys muodostuu korkeasta tukiasemasta ja vastaanottimista, jotka voivat olla kohteeseen asennettavia antenneja tai mukana kuljetettavia USB-modeemeja [33]. Yhteys vastaa meidän mobiililaajakaistaamme, mutta on hitaampi [33]. Asennettuani WiMAX-yhteyksiä huomasin nopeasti, kuinka vaivalloista se on. Antennien tulee olla juuri tietyssä kulmassa tukiasemaa vasten ja välissä ei saa olla fyysisiä esteitä, jotta saadaan minkäänlaista yhteyttä. Tämä on hankalaa varsinkin keskellä kaupunkia kerrostalojen ympäröimänä. Mikäli jo luodun yhteyden väliin rakennetaan talo, yhteyden nopeus puolittuu tai yhteys voi kadota jopa kokonaan.

Fuerteventuralle tämä yhteysmuoto on kuitenkin yleisesti jokseenkin toimiva, koska saarella ei luonnostaan kasva puita ja välimatkat kaupungista toiseen ovat melko lyhyitä. Verkon voisi kuitenkin luoda myös koko saaren kattavana mesh-verkkona, jolloin siitä saattaisi saada nopeamman ja yhteyksistä luotettavampia. Koska kaupunkien välillä ei ole esteitä, voisi mesh-verkkoa käyttää yhdessä WiMAX-teknologian kanssa niin, että yksittäiset kaupungit

olisi katettu mesh-verkoilla PanOULUn tapaan ja kaupunkien välillä data kulkisi WiMAX-yhteydellä.

Mesh-verkon joustavuus on apuna erityisesti haastavissa olosuhteissa, kun tavalliset verkot pettävät. Venäjän hyökkäys Ukrainaan aiheutti internet-yhteyksien heikentymistä ja osittain jopa koko internet-infrastruktuurin kaatumista [34, 35]. Bitcoin Magazine uutisoi, kuinka Ukrainan verkon epävakauden takia ihmiset pystyttivät sen rinnalle mesh-verkkoa päästäkseen käsiksi finanssipalveluihin [37].

5 Demonstraatioverkon toteutus

5.1 Työn tavoitteet

Työn tarkoituksena on tutustua mesh-verkon periaatteeseen, hyötyihin ja mahdollisiin haittoihin, sekä rakentaa demonstraatioverkko ja sen palvelimet, joita KajaPro voisi hyödyntää tulevissa projekteissaan. Demonstraatioverkon avulla voidaan myös mahdollisesti tutkia uusia käyttökohteita ja tuotteita, joihin verkkoa voidaan hyödyntää. Lisäksi tutkitaan, mitä haasteita tulee eteen mesh-verkon kanssa.

Tärkein päämäärä on saada aikaan luotettavat ja dynaamisesti muodostuvat yhteydet. Nopeuksilla ei todennäköisesti ole monessakaan IoT-laitteessa niin väliä, kunhan data liikkuu tarpeeksi sulavasti. Koska viestit ovat melko lyhyitä, ei yksittäinen viesti vie kovinkaan paljon kaistaa. Jos lähetetään kaksikymmentä viestiä sekunnissa, ei datamäärä kasva kovinkaan suureksi. Char-tyyppinen muuttuja vie yhden bitin tilaa [36], joten vaikka lähetetään kaksikymmentä kahdensadan merkin viestiä sekunnissa, on käytetty kaista silti vain noin neljä megabittiä sekunnissa.

5.2 Yleistä

Pääpalvelimen Message-luokka toimii viestin pohjana. Se sisältää viestin headerin eli otsikon, solmun IDn ja mistä solmun anturista data on peräisin. Alun perin viestin oli tarkoitus myös sisältää unix-aikaleima siitä, kun solmu on luonut viestin, mutta internet-yhteyden puutteellisuuden takia aikaa ei voida noutaa mistään. Vaihtoehtoisesti voidaan käyttää PainlessMeshin ylläpitämää mesh-verkon elossaoloaikaa. Itse data on kokonaislukujen joukko, joten dataa voi laittaa enemmänkin kuin vain yhden luvun. Otsikko voi olla mikä tahansa tekstinpätkä, joten sitä voidaan käyttää vertailuun ja lukijaystävälliseen tarkasteluun. Otsikon kautta voidaan myös lähettää muita kuin anturidataa sisältäviä viestejä. Viestin messageType-muuttujalla voidaan selventää, millaista dataa viesti sisältää ja ottaa se huomioon palvelimen jäsen-

täessä viestiä. Viesti lähetetään solmun päässä JSON-muodossa ja puretaan taas pääpalvelimella Message-objektiksi, jota on helpompi käsitellä. Kuvassa 7 on tulostettu viestistä Message-objektiin jäsennettyjä muuttujia.

```
Processing connection: 0
Header: Potentiometer value
nodeId: 3202234485
sensorId: 1
Data: 753,
creationTime: 1645095762
```

Kuva 7. Pääpalvelimen vastaanottama dataviesti jäsennettynä

Serverin ja meshin yhdistämisessä käytetään TCP-protokollan päällä toimivaa [38] WebSocket-protokollaa. Tämä tapa tuntuu järkevältä, sillä se on yleisesti käytössä monessa paikassa ja helppo toteuttaa. Lisäksi se on minulle henkilökohtaisesti jo ennestään tuttu periaate.

5.2.1 Monisäikeisyys

Vaikka asynkronisuus ja monisäikeistäminen eivät varsinaisesti kuulukaan työn aihepiiriin, on molemmissa ohjelmissa kuitenkin otettava huomioon niiden sujuvuuden kannalta, että ne eivät jää odottamaan yksittäisen yhteyden dataa ja siten pysähdy kokonaan tapauksissa, joissa dataa ei tulekaan. Pääpalvelimelle annetaan tämän vuoksi yksi säie jokaista WebSocket-yhteyttä kohden. Näin säie voi odotella taustalla silloin, kun solmulta ei saada dataa.

WeMos-korttien tapauksessa ongelmaa oli hankalampi lähestyä. Työn aikana testattiin kahta tapaa, ajastinta ja säikeistämistä. Koska ESP8266-mikropiirissä ei ole kuin yksi prosessointiyksikkö, jouduttiin säikeistaminen luomaan kokonaan ohjelmallisesti. Ohjelmallisesti luotua säikeistämistä kutsutaan protosäikeistämiseksi ja sen avulla yksi suorittimen fyysinen ydin voidaan puolittaa kahdeksi virtuaaliseksi ytimeksi [39]. Toimenpide vastaa Intelin kehittämää Hyperthreading-teknologiaa [39] ja on lähempänä asynkronisuutta

kuin oikeaa säikeistämistä. Lopulta, molempien tapojen testaamisen jälkeen, ajastin vaikutti paremmalta vaihtoehdolta.

5.2.2 Ulkopuoliset kirjastot

Aikataulutuksen vuoksi työn mesh-verkko rakennettiin suoraan käyttämällä PainlessMesh-kirjastoa. Kirjasto muodostaa verkon tukiaseman ja ylläpitää yhteyksiä solmujen välillä sekä tarjoaa helpot funktiot verkon päivittämiseen ja datan synkronointiin.

WebSocket-yhteydet on rakennettu ArduinoWebsockets-kirjaston funktioilla. Kirjasto käyttää pohjana myös Linux-alustalle saatavilla olevaa TinyWebsockets-kirjastoa, jolla pääpalvelimen yhteydenmuodostus voitiin rakentaa yhteensopivaksi.

Viestit lähetetään solmun ohjelmistossa suoraan JSON-muodossa. Viestin purkamisen helpottamiseksi pääpalvelimen puolella käytetään nlohmann json kirjastoa, jolla saadaan jäsenet suoraan muuttujiin ja sitä kautta Message-objektiksi.

5.3 Solmut

5.3.1 Laitteisto

Solmut rakennettiin Arduino-pohjaisille WeMos D1 Mini -korteille, jollainen näkyy kuvassa 8. D1 Minin ESP8266-mikropiiri sisältää WiFi-moduulin ja kortti on erittäin pienikokoinen ja halpa. Lisäksi siinä on sarjaportti erinäköisten anturien yhdistämiseen.

Kuva 8. WeMos D1 Mini

D1 Minissä ei kuitenkaan ole kuin yksi analoginen pinni, A0, joten jos halutaan enemmän kuin yksi analoginen anturi, tarvitsee väliin rakentaa oma, signaalin analogisesta digitaaliseen kääntävä laite. Digitaalisia pinnejä sen sijaan riittää, joten jos anturi antaa ulos digitaalista dataa, voi sen lisätä sellaisenaan. Tässä työssä anturidataa simuloitiin A0-pinniin kiinnitetyllä potentiometrilla. Tarvittaessa ylimääräisiä antureita voitiin testata ohjelmallisesti luoduilla viesteillä.

Solmujen ohjelmiston pitää toimia yhtä aikaa tukiasemana muille solmuille sekä yhdistää toiseen solmuun kohti pääpalvelinta. Koska WeMos D1 Mini perustuu Arduino-laiteeseen, tapahtuu ohjelmointi Arduinon omalla sovelluksella ja Arduinon omalla, C-kieleen perustuvalla kielellä.

5.3.2 Ohjelmisto

Koska mesh-verkon luomisessa ja tietoturvassa sekä solmujen yhteyksien ylläpitämisessä on niin paljon vaiheita, ei tämän työn aikataulun puitteissa ole mahdollista rakentaa mesh-verkkoa itse. Mesh-verkko voidaan luoda helposti PainlessMesh-kirjaston avulla. Se hallitsee solmujen yhteyksien muodostamisen ja datan synkronoinnin luoden yhden langattoman verkon ja tukiaseman,

johon päätelaitteet voivat kytkeytyä. PainlessMesh käyttää WPA-salausta [40], joten tietoturva on sama kun kaupallisilla langattomilla tukiasemilla.

Solmut itse yhdistyvät toisiinsa ja saattavat vaihtaa yhteyksiään eri solmujen välillä nopeaankin tahtiin riippuen siitä, minkä yhteyden näkevät optimaalisimpana, vaikka edellinen yhteys ei olisikaan välttämättä katkennut. [41.]

PainlessMesh-verkon synkronointi tapahtuu kutsumalla mesh-objektin update-funktiota. Kuvassa 9 on solmun logiikka säikeittäin ja säikeiden ajamat funktiot näkyvät kuvassa 10. Funktio käskee solmua hakemaan kaikki senhetkiset viestit muilta solmuilta, joihin se on sillä hetkellä yhteydessä. Helposti käytettävää yksittäistä funktiota voidaan kutsua missä kohtaa koodia tahansa. Tämä antaa vapaat kädet tehdä mitä tahansa muuta koodissa ja päivittää verkko, kun siihen on hyvä hetki. Tässä vaiheessa mesh-verkon ylläpito tuntuu kuitenkin tärkeimmiltä, joten niitä päivitetään tiheimmin. Säikeille annettiin ajat, joilla verkkoa itseään pyritään päivittämään 10 millisekunnin välein ja pääpalvelimelle data synkronoidaan 100 millisekunnin välein. Vaikka loop-funktiossa kutsutaan säikeitä melko nopeaan tahtiin, shouldRun-funktio pitää huolen ajoituksesta. Funktioita ei ajeta, ellei shouldRun-funktion palautusarvo ole tosi. Mikäli yhteyttä palvelimelle ei ole, yritetään yhdistää uudelleen kerran sekunnissa.

```
void loop() {
        // shouldRun function checks if the interval has been waited
        if(meshThread.shouldRun()) {
                meshThread.run();
        }

        if(websocketThread.shouldRun() && connected) {
                websocketThread.run();
        }

        if(websocketConnectingThread.shouldRun() && !connected) {
                websocketConnectingThread.run();
        }
}
```

Kuva 9. Solmun ohjelmistosilmukka

```
void connectToWebSocket() {
        Serial.println("Trying to connect to the websocket");
        connected = ws.connect(SERVER_IP, SERVER_PORT, "ws://");
        if(connected) {
                Serial.println("Connected");
        }
        else {
                Serial.println("Didn't connect");
        }
}

void updateWebSocket(){
        if(ws.available()) {
                Serial.println("Syncing with websocket...");
                // Let the websockets ws check for incoming messages
                ws.poll();
                WSSendSensorData();
        }
}

// Gather information from the nodes and send your own
// it will run the user scheduler as well
void updateMesh() {
        mesh.update();
}
```

Kuva 10. Säikeiden ajamat funktiot

Pääpalvelimeen yhdistämiseen solmun mesh-ohjelmiston sekaan pystyttiin lisäämään ArduinoWebsockets-kirjaston funktioita, jotka yrittävät tietyn väliajoin yhdistää pääpalvelimeen. WebSocket-objektin connect-funktio tosin odottaa vähän aikaa yhteyttä, mikä taas pysäyttää muun ohjelman pyörimisen hetkellisesti, joten monisäikeistäminen on tarpeellinen. Koska säikeistämistä ei voida suorittaa oikeilla prosessorin säikeillä, se ei eliminoi odotusaikaa kokonaan. Kuten kuvan 11 lokista voidaan huomata, yhdistämisen ja virheilmoituksen väliin ei ikinä saada mesh-verkon viestejä. Ohjelma kuitenkin vaikuttaa pyörivän hieman sulavammin protosäikeistämisen myötä.

Kuva 11. Lokia solmun ohjelmistosta

Myöhemmin protosäikeistäminen vaihdettiin itse tehtyyn ajastimeen, sillä säikeistämisen kanssakaan WebSocket-yhteyttä ei saatu pidettyä yllä. Koodi ei muuttunut paljon. Kuvassa 12 näkyy ajastimen toiminta ja se, kuinka kuvan 10 funktioita käytetään nyt suoraan. Uudelleenyhdistämisen ajoitusta on myös muutettu niin, että pienimmillään yritetään kymmenen sekunnin välein ja joka yrityskerran jälkeen aikaa kasvatetaan sekunnilla. Tämä auttoi jonkin verran yhteyden muodostamiseen. Myöhemmin havaittiin, että yhteydenmuodostus toimi hyvin, kunhan mesh-verkossa ei aluksi ollut kuin yksi solmu. Solmuja voitiin yhdistää yhteydenmuodostuksen jälkeen ja ne kaikki yhdistyivät oitis tästä eteenpäin. Tietenkään tämä ei ole ideaalista, kun kyseessä pitäisi olla dynaamisesti yhteyksiä luova järjestelmä, mutta kohti paremmin toimivaa ohjelmaa joka tapauksessa. Ajastin sai jäädä.

```
// Timers
int prevMillis = 0;
int timerWS = 0;
// To make more failed rerties result longer waiting between
int wsTries = 0;
void loop() {
        updateMesh();

        // WebScoket timer
        timerWS += (millis() - prevMillis);
        prevMillis = millis();

        if(connected && timerWS > 100) {
                timerWS = 0;
                updateWebSocket();
        }
        else if (!connected && timerWS > 10000 + wsTries * 1000) {
                timerWS = 0;
                wsTries++;
                if( connectToWebSocket() ) {
                        prevMillis = millis();
                }
        }
}
```

Kuva 12. Säikeiden ajamat funktiot

Tavoitteena olisi, että mesh-verkon välistä datan synkronointia tapahtuisi koko ajan, kun taas WebSockettiin data lähetettäisiin aina, kun voidaan. Solmujen yhteyksien vaihtelu toistensa välillä tuntuu hankaloittavan WebSockettiin yhdistämistä ja yhteyden ylläpitämistä.

5.4 Pääpalvelin

5.4.1 Laitteisto

Pääpalvelimeksi valittiin Raspberry Pi, joka on tarpeeksi kompakti ja halpa, mutta mahdollistaa silti monimutkaisemman ohjelmiston käytön. Tässä tapauksessa käytettiin tarkemmin 3b-mallia, koska sellainen oli valmiiksi saatavilla. Ohjelmisto on kuitenkin pyritty toteuttamaan niin, että se toimii mahdollisimman yhteensopivasti kaikella laitteistolla Linux-ympäristössä.

Linux pääpalvelimen alustana mahdollistaa palvelinohjelmiston luomisen matalan tason ohjelmointikielillä, sekä järjestelmäkirjastoja apuna käyttäen. Tämä parantaa pienten yksityiskohtien säätämistä mutta pitää valmiiden kirjastojen avulla kehityksen nopeana. Lisäksi Linux-koodin pitäisi kääntyä muille Linux-laitteille helposti.

5.4.2 Ohjelmisto

Palvelinohjelmisto toteutettiin C++-kielellä. Yhteydenottoon käytettiin alun perin järjestelmän WebSocket-kirjastoja, mutta myöhemmin vaihdettiin TinyWebsockets-kirjastoon yhteensopivuusongelmien takia. Solmujen ohjelmistossa käytetty ArduinoWebsockets kirjasto vaatii vastauksena tietynlaisella tavalla muotoillun hyväksyntäviestin, kun WebWockettiin on saatu yhteys. Kirjaston dokumentaatio on puutteellinen, joten itse tehdyillä yhteyden kättelyillä ei ollut mahdollista vastata oikealla tavalla. Pääpalvelin kykenee ylläpitämään yhteyksiä useampaan solmuun kerralla ja prosessoimaan niiden dataa asynkronisesti. Rapsberry Pi voitiin yhdistää mesh-verkkoon ja sen verkkoasetukset määrittää täysin samalla lailla kuin mihin tahansa langattomaan tukiasemaan.

Solmujen yhteyksille annetut säikeet prosessoivat itsenäisesti solmujen lähettämän datan. Toisin kuin solmuissa, pääpalvelimella voidaan käyttää oikeita säikeitä, sillä Raspberry Pi 3B:ssä on 4 ydintä [42].

Kuolleiden yhteyksien siivoamiseen tarvittiin yksi säie. Mikäli tarpeeksi monta tyhjää viestiä vastaanotetaan peräkkäin, yhteys katsotaan kuolleeksi. Solmun prosessointisäie tulkitsee oman yhteytensä poistamisen tarpeen ja asettaa Connection-objektin shouldJoin-lipun todeksi. Siivoojasäie käy tietyin väliajoin kaikki muut säikeet ja niiden yhteydet läpi ja vapauttaa poistettavaksi merkityt objektit muistista. Tämä tyhjien viestien mukaan laskettava periaate toimi alussa, mutta myöhemmin TinyWebSockets-kirjastoon vaihtamisen jälkeen viestejä jäädään odottamaan ikuisesti, joten tyhjiä viestejä ei tullut. Täytyi siis luoda ajastin, joka lisättiin siivoojasäikeelle.

Internet-yhteyden mahdollisen puuttumisen takia solmuilla on ainoastaan tieto siitä, miten kauan mesh-verkko on ollut pystyssä. Tämän vuoksi ainoa tapa on lähettää viestin mukana verkon elossaoloaika ja verrata jokaista viestiä keskenään. PainlessMeshin ajansynkronointi on hyvä ja tapahtuu kuvassa 13 näkyvän lokin mukaan useammankin kerran sekunnissa. Kunhan palvelimella on tarpeeksi yhteyksiä muihin solmuihin, tämä periaate todennäköisesti riittää. On tarpeellista tietää vain, kumpi tapahtuma on tapahtunut aiemmin. Mikäli uusia viestejä — ja sitä kautta uusinta aikaa — ei tule pääpalvelimelle ollenkaan, yhteydet jäävät roikkumaan. Tämän vuoksi alkuperäinen tyhjien viestien tarkkailumenetelmä oli hyvä säilyttää ajastimen rinnalla.

```
18:50:21.476 -> Received from 3202234485 msg={"creationTime":1645095762,"
18:50:21.575 -> Adjusted time 213676508. Offset = -13005
18:50:21.675 -> Received from 3202234485 msg={"creationTime":1645095762,"
18:50:22.105 -> Adjusted time 214175400. Offset = -7132
18:50:22.205 -> Send text-only message to server
18:50:22.437 -> Trying to connect to the websocket on: 10.52.171.200
18:50:27.573 -> Didn't connect
18:50:27.573 -> Send text-only message to server
18:50:27.573 -> Adjusted time 217175674. Offset = -2483146
18:50:27.871 -> Received from 3202234485 msg={"creationTime":1645095762,"
18:50:27.971 -> Received from 3202233515 msg={"creationTime":1645095762,"
18:50:28.103 -> Adjusted time 220170535. Offset = 2479269
18:50:28.203 -> Received from 3202233515 msg={"creationTime":1645095762,"
18:50:28.401 -> Received from 3202234485 msg={"creationTime":1645095762,"
18:50:28.501 -> Received from 3202234826 msg={"creationTime":1645095762,"
18:50:28.567 -> scandone
18:50:28.700 -> Adjusted time 220750539. Offset = -38771
18:50:28.998 -> Send text-only message to server
18:50:29.031 -> Received from 3202234826 msg={"creationTime":1645095762,"
18:50:29.130 -> Received from 3202234826 msg={"creationTime":1645095762,"
18:50:29.230 -> Adjusted time 221305392. Offset = 36807
18:50:29.329 -> Received from 3202234826 msg={"creationTime":1645095762,"
18:50:29.727 -> Adjusted time 221812373. Offset = 825
18:50:30.555 -> Trying to connect to the websocket on: 10.52.171.200
18:50:35.723 -> Didn't connect
18:50:35.723 -> Send text-only message to server
```

Kuva 13. Ajansynkronoinnit solmun ohjelmistolokissa

6 Yhteenveto

Loppujen lopuksi yhteysongelmat saatiin sen verran kuntoon, että yhdistäminen onnistuu ja yhteydet pysyvät päällä. Nopeus on enemmän kuin riittävä ja kaikkien mesh-verkon solmujen viestit saadaan pääpalvelimelle asti. Kuvassa 14 on katkelma pääpalvelimen lokista, jossa näkyy tulostettuna kahden verkon laitteen viestejä.

Kuva 14. Lokia pääpalvelimelta

Mitä luotettavuuteen tulee, on siinä vielä parantamisen varaa. Koska mesh-verkon periaatteena olisi yhdistää laitteita missä tahansa järjestyksessä milloin tahansa, ei tämä ohjelmisto vielä kata kaikkia tarpeita. Demonstraatioverkko kuitenkin on nyt siinä tilanteessa, että sen päälle voi alkaa rakentaa jo jotain toiminnallisuuttakin. Kun verkkoa oikeasti käytetään, voi tulla ilmi parempia tapoja yhteyksien ylläpitoon.

6.1 Yhteyksien luotettavuus

Pelkän mesh-verkon luominen PainlessMesh-kirjaston avulla on pikku juttu, mutta kun siihen halutaan lisätä pääpalvelin, pitää ottaa huomioon monenlaisia asioita. Esimerkiksi tämän työn perustana on saada mahdollisimman luotettavat yhteydet laitteiden välillä. Yhteyden ylläpitäminen pääpalvelimeen oli alusta asti haasteellista. Suuntaa antavasti yhteyden luotettavuus parani säikeistämisen myötä, mutta ei silti ollut tarpeeksi luotettava varsinkaan yhteyden muodostamisessa. Tämän perusteella ongelma saattaa olla nimenomaan siinä, että ohjelma pysähtyy hetkeksi silloin tällöin. Toinen vaihtoehto ongelman syyksi voi myöhempien havaintojen perusteella olla se, että solmut yrittävät yhdistää samaan porttiin samaan aikaan ja joutuvat tappelemaan siitä, mikä solmu saa yhdistää. Tällöin yksikään solmu ei saa yhdistettyä palvelimeen.

Mesh-osiossa ongelmia tulee, koska solmut tuntuvat hyppivän eri yhteyksien välillä nopeaan tahtiin. Päätelaitteen kytkeminen ja sen yhteyden säilyttäminen onnistuu, sillä päätelaitteet eivät vaadi kokoaikaista tiedon kulkua, mutta WebSocket-palvelin tarvitsee kokoaikaisen yhteyden. Tämän vuoksi yhteys katkeaa herkemmin kokonaan pienien yhteysongelmien aikana. Tämä vaikuttaa tietovirtaan ja pääpalvelimen tulee kyetä ottamaan nämä mahdolliset katkokset huomioon. Solmut kuitenkin aina palaavat takaisin yhteyden saatuaan, joten tämä ei välttämättä ole iso ongelma.

Suurimmaksi ongelmaksi jäi se, että WebSocket-protokolla ei oikein tunnu toimivan mesh-verkon päällä, kun useampi solmu on jo yhdistetty. Yhteys pääpalvelimeen toimii täydellisesti, kun yhdistetään tavalliseen verkkoon tai jos mesh-verkko koostuu vain yhdestä solmusta, mutta sama koodi ei suostu millään yhdistymään mesh-verkossa useamman solmun ollessa jo valmiiksi mukana. Kun solmu yhdistyy, se toimii hyvin ja sen jälkeen solmuja voidaan alkaa lisätä. Kun yksi solmu on saatu yhdistettyä pääpalvelimeen, alkaa se tuoda muidenkin verkon laitteiden viestejä. Ongelmia tosin vielä tulee, jos tämä ensimmäisen solmun yhteys katkeaa, eivät muut solmut voi enää yhdistyä.

6.2 Laitteiston huomiointi

Arduinon vajavaisuus alemman tason ohjelmoinnissa tuotti päänvaivaa, kun kirjastot tekivät itse asioita, joita ei ole dokumentoitu. Tämä taas aiheutti sen, että jo tehdystä pääpalvelinohjelmistosta jouduttiin vaihtamaan osioita ylemmän tason kirjastojen funktioihin. Mikäli Arduino-alustaa halutaan käyttää muiden laitteiden kanssa, kannattaa ohjelmiston suunnittelu aloittaa nimenomaan Arduino-laitteen puolelta ja etenkin sen puutteista. Tämän työn alkuvaiheessa tosin WeMos-kortit olivat vielä tilauksessa, joten ajan säästämisen vuoksi oli järkevintä aloittaa pääpalvelimesta.

Pillä käytettävä RaspbianOS ei halua pysyä verkossa ilman internet-yhteyttä. Siksi, jos aiemmin oli yhdistetty toiseen, internet-yhteydelliseen verkkoon, Pi palasi siihen automaattisesti havaittuaan, että mesh-verkossa ei ole yhteyttä. Asia korjautui, kun poistettiin muut verkot muistista. Koska pääpalvelin toimii millä tahansa Linux-alustalla, voi eroja olla paljonkin eri Linux-jakeluiden välillä riippuen asetuksista. Nämä ovat muotoseikkoja, kunhan käyttäjä osaa asettaa käyttöjärjestelmänsä kuntoon, mutta hyvä ottaa huomioon.

6.3 Yrityskäyttö

Koska tässä työssä jouduttiin käyttämään paljon kolmannen osapuolen kirjastoja, joilla on erilaisia lisenssejä, ei työ välttämättä sovi sellaisenaan yrityksen käyttöön. Varsinkin GPL v3.0 -lisensoitujen kirjastojen muokkauksen tarve aiheuttaa sen, että lisenssin mukaan muokattu lähdekoodi joudutaan julkaisemaan. Riippuu yrityksen tarpeista, haittaako ohjelmiston koodin julkaiseminen tuotteen valmistusta ja myyntiä.

TinyWebsocketin koodi vaati muokkaamista, että Raspberry Pin C-kääntäjä sai sen käännettyä. Tämä tarkoittaa GPL v3.0 -lisenssin mukaan, että muokattu lähdekoodi tulee julkaista avoimesti. Tämä ei välttämättä ole ongelma, mikäli koko ohjelmaa ei tarvitse julkaista, vaan pelkkä kirjasto riittää. Tämä on kuitenkin otettava huomioon erityisesti yrityskäytössä.

6.4 Jatkokehitys

Kuten aiemmin mainittu, varsinkin yhteyksien luotettavuus pääpalvelimelle on tässä vaiheessa vasta haussa. Mesh-verkko ja WebSocket toimivat täydellisesti erikseen, mutta eivät suostu toimimaan kunnolla, kun ne laitetaan yhteen. Tämä on ironista, sillä juuri se on koko demonstraatioverkon idea. Erityisesti pääpalvelimen lisääminen useamman solmun verkkoon tulisi toimia moitteetta. Yhteyksien muodostuttua verkko vaikuttaa kuitenkin toimivan varsin hyvin, eikä mahdollinen ylimääräinen viive vaikuta verkon sujuvuuteen.

Jos mahdollista, tulisi WebSocket-koodi tuoda mahdollisimman lähelle rautaa. Nykyiset kirjastot ovat käteviä nopeaan demokoodin rakentamiseen, mutta niistä ei näe, mitä ne taustalla tekevät. Koska suurimmaksi yhteysongelmaksi paljastui nimenomaan TinyWebsockets ja ArduinoWebsockets -kirjastoilla toteutettu pääserveriyhteys, tulisi tähän yhteyteen käyttää aikaa, että saadaan oma WebSocket-kirjasto tarkempia hienosäätöjä varten. Ajastus on tärkeä osa toimivaa yhteyttä.

Mikäli jatketaan valmiiden kirjastojen avulla, protosäikeistämisen ja ajastimen sijaan solmun ohjelmistossa voitaisiin kokeilla asynkronisuutta, sillä WebSockettiin yhdistäminen vie tällä hetkellä aivan liikaa aikaa silmukasta. WeMos-kortin suurin ongelma on, että sillä on vain yksi prosessointiyksikkö ja -ydin. Kahdella ytimellä ajettaessa voitaisiin säikeistää mesh- ja WebSocket-prosessoinnit kahdeksi erilliseksi ohjelmaksi, jolloin ne eivät joutuisi jakamaan prosessorin syklejä.

Vaikka nykyinen ajastin lisättynä tyhjien viestien määrään toimii melko hyvin, yhteyksien tarkkailua tulee kehittää, mikäli verkkoa aiotaan käyttää sellaiseen tarkoitukseen, jossa yhteyksien tilojen pitää olla koko ajan tiedossa. Tällä hetkellä ei voida määrittää, milloin yhteys oikeastaan katkesi, sillä aikaleimat saadaan ainoastaan solmujen viestien yhteydessä. Jos solmuja on vain yksi, ei sen tilasta saada mitään tietoa, mikäli se ei kykene lähettämään viestejä pääpalvelimelle. Niille solmuille, jotka eivät ole suorassa yhteydessä pääpalvelimeen, voitaisiin kehittää tarkkailufunktioita, jotka selvittävät muiden solmujen viesteistä, ovatko nämä epäsuorat yhteydet vielä pystyssä.

Tämä lisää luotettavuutta, mutta ei vieläkään kokonaan poista sitä rajoitusta, että viestejä täytyy tulla koko ajan.

6.5 Oppimisprosessi

Työn aikana olen oppinut lisää siitä, mitä WebSocket-ohjelmoinnissa tulee ottaa huomioon sekä pintapuolisesti yhteyksien tietoturvasta. Mesh-periaate ja etenkin sen joustavuus ja yhdisteltävyys on tullut tutuksi pohdiskellessani demonstraatioverkon toteutusta ja mahdollisia käyttökohteita.

Lisäksi olen huomannut, että asynkronisuus ja yhtäaikainen ohjelman ajaminen on kriittinen osa IoT-laitteiden toimintaa ja toteuttamattomana tai huonosti toteutettuna ajaa koko ohjelman toimimattomaksi. Asynkronista ohjelmointia täytyy opiskella vielä lisää, sillä en saanut sitä tämän työn puitteissa ollenkaan toimimaan WeMos-kortilla.

Arduino alustana on minulle vielä uusi ja olen tämänkin työn jälkeen tutustunut siihen vasta pintapuolisesti. Ennen työtä minulla oli sellainen käsitys Arduinosta, että sille voidaan kirjoittaa lähempänä rautaa olevaa koodia, mutta tosiasiassa Arduino-kirjastoista ja ohjeista oli todella vaikeaa löytää mitään, mikä mahdollistaisi alemman tason koodin. Ainoa toinen vaihtoehto näyttää olevan kirjoittaa ajurimaista C-koodia, joka vaatii enemmän aikaa. Esimerkiksi pääpalvelimen WebSocket-kutsut jouduttiin korvaamaan TinyWebsockets-kirjaston funktioilla juurikin siksi, ettei Arduinossa voitu käyttää funktioita, jotka olisivat samalla tasolla Linuxin järjestelmä-kirjastojen kanssa. Solmun ohjelmiston virheenkorjaus vaatii paljon lokin kirjoittamista, sillä Arduinon kehitysympäristössä ei voi lisätä ohjelman keskeytyskohtia.

Kaiken kaikkiaan työ on ollut maistiainen laiteohjelmoinnista ja innostanut hakemaan lisää tietoa. Arduino alustana vaatii vielä hieman opiskelua, mutta tunne jäi, että sen tarjoamat kirjastot voisivat olla enemmän Linux-järjestelmäkirjastojen kaltaisia: Alemmalla tasolla, mutta kuitenkin aikaa säästäviä.

Mesh-verkko on joustavarakenteinen ja helppo pystyttää haastavillakin alueilla. Se on jo tullut osaksi normaalia langattoman tiedonvälityksen tekno-

logiaa ja kaikkia käyttökohteita ei olla vielä varmasti löydetty. Vielä hieman asemaansa hakeva demonstraatioverkko on kuitenkin käytettävä ja ensimmäinen askelma, josta voi lähteä jatkokehittämään parempaa.

7 Lähteet

1 Chai, Wesley. wireless mesh network (WMN). (2021). TechTarget. Saatavilla: https://www.techtarget.com/searchnetworking/definition/wireless-mesh-network. Viitattu 15.2.2022

2 Karibasic, Arvin. How Will the Internet of Things Use Blockchain?. (2016). Perficient. Saatavilla: https://blogs.perficient.com/2016/12/15/how-will-internet-of-things-use-blockchain/. Viitattu 15.2.2022

3 Comparison – Centralized, Decentralized and Distributed Systems. (2021). GeeksforGeeks. Saatavilla: https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems. Viitattu 19.4.2022

4 Vertaisverkko. Wikipedia. Saatavilla: https://fi.wikipedia.org/wiki/Vertaisverkko. Viitattu 19.4.2022

5 Mohanan, Remya. What Are Distributed Systems? Architecture Types, Key Components, and Examples. (2022). Toolbox Tech. Saatavilla: https://www.toolbox.com/tech/cloud/articles/what-is-distributed-computing. Viitattu 19.4.2022

6 Antonenko, Dimitri. Centralized vs Decentralized vs Distributed Networking Explained. (2022). BusinessTechWeekly. Saatavilla: https://www.businesstechweekly.com/operational-efficiency/computer-networking/centralized-vs-decentralized-vs-distributed-networking-explained. Viitattu 19.4.2022

7 Mikä mesh-verkko on?. Google Nest Help. Saatavilla: https://support.google.com/googlenest/answer/7182746?hl=fi. Viitattu 11.2.2022

8 ESP-WIFI-MESH. (2016). Espressif. Saatavilla: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/esp-wifi-mesh.html. Viitattu 11.2.2022

9 Decentralized Networks. (2018). Coinbundle. Saatavilla: https://medium.com/coinbundle/decentralized-networks-1d5e2e92953b. Viitattu 11.2.2022

10 Distributed Network. (2012). Technopedia. Saatavilla: https://www.techopedia.com/definition/27788/distributed-network. Viitattu 15.2.2022

11 Centralized, Decentralized, & Distributed Networks. (2021). Cryptopedia. Saatavilla: https://www.gemini.com/cryptopedia/blockchain-network-decentralized-distributed-centralized. Viitattu: 22.3.2022

12 Basics of Wireless Sensor Networks (WSN) | Classification, Topologies, Applications. (2019). Electronics Hub. Saatavilla: https://www.electronicshub.org/wireless-sensor-networks-wsn/#Network_Topologies_in_WSN. Viitattu 4.4.2022

13 IoT Connectivity – Network Topology. Infinite Information & Technology. Saatavilla: http://www.infiniteinformationtechnology.com/internet-things-network-topology. Viitattu 4.4.2022

14 Verkkotopologian tyypit. Education-wiki. Saatavilla: https://education-wiki.com/7299355-types-of-network-topology. Viitattu 4.4.2022

15 Mocnej, Jozef; Seah, Windown K.G.; Pekar, Adrian; Zolotova, Iveta. Decentralised IoT Architecture for Efficient Resources Utilisation. (2018). IFAC PapersOnLine. Saatavilla: https://www.sciencedirect.com/science/article/pii/S2405896318308942. Viitattu 1.4.2022

16 Blockchain 2.0: Hajautettu laskenta ja hajautetut sovellukset selitetty [Osa 11]. CodePre. Saatavilla: https://codepre.com/fi/blockchain-2-0-explicacion-de-la-computacion-distribuida-y-las-aplicaciones-distribuidas-part-11.html. Viitattu: 4.4.2022

17 Ero rinnakkais- ja hajautetun tietojenkäsittelyn välillä. Sawakinome. Saatavilla: https://fi.sawakinome.com/articles/technology/difference-between-parallel-and-distributed-computing-2.html. Viitattu 5.4.2022

18 Leppälä, Samuli. Oletko mukana? Miljoonan laitteen välille hajautettu laskenta ylitti yhden eksaflopin laskentatehon – tavoitteena koronaviruksen selättäminen. (2020). Tivi. Saatavilla: https://www.tivi.fi/uutiset/oletko-mukana-miljoonan-laitteen-valille-hajautettu-laskenta-ylitti-yhden-eksaflopin-laskentatehon-tavoitteena-koronaviruksen-selattaminen/78a8c6ed-83bf-42bc-a479-32f66b0dfd28. Viitattu 5.4.2022

19 Folding@home. Folding@home. Saatavilla: https://foldingathome.org/. Viitattu 5.4.2022

20 How Much & Which Building Materials Block Cellular & WiFi Signals?. (2021). Signalbooster. Saatavilla: https://www.signalbooster.com/blogs/news/how-much-which-building-materials-block-cellular-wifi-signals. Viitattu 16.3.2022

21 Creating NodeMCU Mesh Network using ESP12 and Arduino IDE. (2020). IoTDesignPro. Saatavilla: https://iotdesignpro.com/projects/creating-nodemcu-mesh-networking-using-esp12-and-arduino-ide. Viitattu 11.2.2022

22 PainlessMesh. Network layout. Saatavilla: https://gitlab.com/painlessMesh/painlessMesh/-/wikis/home#network-layout. Viitattu 18.2.2022

23 Bai, Yong. Kuva mesh verkon rakenteesta. (2012). ReseachGate. Saatavilla: https://www.researchgate.net/figure/Wireless-mesh-network-architecture-with-mesh-gateway-mesh-routers-and-mesh-clients_fig2_257877841. Viitattu 16.2.2022

24 Ashwood-Smith, Peter. Shortest Path Bridging IEEE 802.1aq Tutorial and Demo. (2010). Huawei. Saatavilla: https://archive.nanog.org/meetings/nanog50/presentations/Sunday/IEEE_8021aqShortest_Path.pdf. Viitattu 21.3.2022

25 Kolderup, Kevin. Introducing Bluetooth Mesh Networking. (2017). Bluetooth. Saatavilla: https://www.bluetooth.com/blog/introducing-bluetooth-mesh-networking/. Viitattu 16.2.2022

26 van de Velde, Sander. Cheap Arduino mesh using RF24 radio modules. (2016). Saatavilla: https://sandervandevelde.wordpress.com/2016/05/30/cheap-arduino-mesh-using-rf24-radio-modules/. Viitattu 16.2.2022

27 Cotrim, Jeferson Rodrigues. (2020). LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication. PMC. Saatavilla: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7435450/. Viitattu 25.2.2022

28 Zigbee Wireless Mesh Networking. Digi. Saatavilla: https://www.digi.com/solutions/by-technology/zigbee-wireless-standard. Viitattu 11.2.2022

29 panOULU. Wikiwand. Saatavilla: https://www.wikiwand.com/fi/PanOULU. Viitattu 2.3.2022

30 Panoulu tukiasemakartta. OUKA. Saatavilla: https://kartta.ouka.fi/IMS/fi?REQUEST=Search&q=category:%20panoulu%20tukiasemat&Municipality=564&LAYERS=Opaskartta. Viitattu 2.3.2022

31 Koutonen, Jouni; Kauta, Jasmina; Holopainen, Hanna. Olympialaiset kiellettiin monilla työpaikoilla, kun Suomen mitalisade alkoi – verkko uhkasi kaatua: "Kyllä se suhteellisen isona yllätyksenä tuli". (2022). Yle Uutiset. Saatavilla: https://yle.fi/uutiset/3-12313008. Viitattu 8.3.2022

32 Nokso-Koivisto, Hannu. Maailman suurin kunnallinen langaton silmukkaverkko Oklahoma Cityssä. (2008). Ilta-Sanomat. Saatavilla: https://www.is.fi/digitoday/art-2000001573385.html. Viitattu 22.3.2022

33 Mitchell, Bradley. What Does WiMAX Internet Mean?. (2020). Lifewire. Saatavilla: https://www.lifewire.com/wimax-wireless-networking-818321. Viitattu 8.3.2022

34 Cooney, Michael. Ukraine internet battered but not out. (2022). Networkworld. Saatavilla: https://www.networkworld.com/article/3652516/ukraine-internet-battered-but-not-out.html. Viitattu 21.3.2022

35 Collier, Kevin; Talmazan, Yuliya. Ukraine facing major regional internet outages as Russian invasion continues. (2022). BBC News. Saatavilla:

https://www.nbcnews.com/tech/tech-news/ukraine-facing-major-regional-internet-outages-russian-invasion-contin-rcna18973. Viitattu 21.3.2022

36 C - Data Types. Tutorialspoint. Saatavilla: https://www.tutorialspoint.com/cprogramming/c_data_types.htm. Viitattu 25.3.2022

37 Bent, Marty. Bitcoiners Are Building Mesh Networks In Ukraine. (2022). Bitcoin Magazine. Saatavilla: https://bitcoinmagazine.com/culture/bitcoiners-building-mesh-networks-in-ukraine. Viitattu 21.3.2022

38 Hickson, I. The WebSocket protocol draft-hixie-thewebsocketprotocol-76. (2010). Google Inc. Saatavilla: https://datatracker.ietf.org/doc/html/draft-hixie-thewebsocketprotocol-76. Viitattu 22.3.2022

39 Alden, Dew. How to "Multithread" an Arduino (Protothreading Tutorial). (2016). Arduino ProjectHub. Saatavilla: https://create.arduino.cc/projecthub/reanimationxp/how-to-multithread-an-arduino-protothreading-tutorial-dd2c37. Viitattu 4.3.2022

40 Security. (2019). PainlessMesh Gitlab issue. Saatavilla: https://gitlab.com/painlessMesh/painlessMesh/-/issues/161. Viitattu 16.2.2022

41 PainlessMesh Technical Documentation. (2020). PainlessMesh Gitlab. Saatavilla: https://gitlab.com/painlessMesh/painlessMesh/-/wikis/home. Viitattu 22.3.2022

42 Raspberry Pi 3 Model B. Raspberry Pi. Saatavilla: https://www.raspberrypi.com/products/raspberry-pi-3-model-b. Viitattu 21.3.2022