TAMPEREEN TEKNILLINEN YLIOPISTO

Juha Onkila

Käyttöjärjestelmän edut langattomassa anturiverkossa

Diplomityö

Tarkastajat: Professori Marko Hännikäinen ja professori Timo D. Hämäläinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 04.05.2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Sähkötekniikan koulutusohjelma
JUHA ONKILA: Käyttöjärjestelmän edut langattomassa anturiverkossa
Diplomityö, 57 sivua
Toukokuu 2011
Pääaine: Sulautetut järjestelmät
Tarkastajat: professori Marko Hännikäinen, professori Timo D. Hämäläinen
Avainsanat: Langattomat anturiverkot, reaaliaikaiset käyttöjärjestelmät, TUTWSN

Langattomat anturiverkot ovat kehittyneet viime vuosina nopeaa tahtia. Ominaisuuksien määrä on kasvanut, mutta toisaalta laitteistoresurssit ovat pysyneet lähes ennallaan. Lisääntyneen monimutkaisuuden hallintaan on kehitetty eritoten langattomia anturiverkkoja varten suunniteltuja reaaliaikaisia käyttöjärjestelmiä. Nämä käyttöjärjestelmät ottavat anturiverkkojen erityislaatuisuuden esimerkiksi energiankulutuksen ja laitteistoresurssien suhteen huomioon.

Tässä työssä otettiin SensorOS-niminen käyttöjärjestelmä käyttöön TUTWSN-anturiverkon reitittävän anturilaitteen sulautetussa ohjelmistossa. TUTWSN on mikrokontrolleripohjainen vähän energiaa kuluttava langaton anturiverkkoteknologia. Käyttöjärjestelmän käyttöönotolla pyrittiin parantamaan järjestelmän suorituskykyä sekä selkeyttämään järjestelmän rakennetta. Osana työtä pyrittiin selvittämään yleisesti käyttöjärjestelmän käyttöön liittyviä etuja langattomassa anturiverkossa. Käyttöönotto suoritettiin kahdessa vaiheessa. Näin saatiin paremmin toimiva järjestelmä aikaiseksi ja lisäksi ensimmäisen vaiheen järjestelmä toimi alkuperäisen TUTWSN-anturiverkon ohella hyvänä verokkijärjestelmänä.

Työssä luotua järjestelmää vertailtiin verokkijärjestelmiin usein eri tavoin. Suorituskykyä vertailtiin esimerkkijärjestelmän kannalta oleellisia parametrejä, kuten radion päälläoloaikaa ja keskeytysviivettä, käyttäen. Lisäksi vertailtiin järjestelmien resurssienkäyttöä. Rakennevertailu suoritettiin subjektiivisesti sekä mutkikkuusmittaa käyttäen.

Käyttöjärjestelmän havaittiin parantavan esimerkkijärjestelmän suorituskykyä ja reaaliaikavaatimusten toteuttaminen helpottui. Keskeytysviive oli käyttöjärjestelmää käytettäessä lähes vakio ja radion päälläoloaika parani 80 prosentista lähes 95 prosenttiin. Lisäksi käyttöjärjestelmä selkeytti järjestelmän rakennetta. Käyttöjärjestelmä helpotti energiankulutuksen hallintaa ja tehosti resurssien hallintaa. Toisaalta järjestelmän muistin kulutus lisääntyi huomattavasti. Datamuistin käyttö kasvoi 28 prosenttia alkuperäiseen järjestelmään verrattuna.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Electrical engineering
JUHA ONKILA: Benefits of operating system in wireless sensor networks
Master of Science Thesis, 57 pages
Mayl 2011
Major: Embedded systems
Examiners: Marko Hännikäinen, Timo D. Hämäläinen
Keywords: Wireless sensor networks, RTOS

Wireless sensor networks (WSN) have developed constantly during last years. Features have increased but on the other hand resources have remained at the same level. Operating systems designed just for WSNs have been developed to help increased complexity. These operating systems take into account WSNs speciality for example in energy consumption and hardware resources.

In this thesis SensorOS operating system was adopted in embedded software of TUTWSN wireless sensor network. As part of the thesis, general benefits of operating systems in wireless sensor networks are discoverd. OS deployment was done in two steps to gradually porting existing functionality to new OS environment and carry out performance comparison in pieces.

System with OS was compared to previous systems. Performance was analyzed with essential parameters of TUTWSN's point of view like interrupt delay and radio's on time. In addition, use of resources was compared. Structure comparison was done in subjective way and by analysing cyclomatic complexity.

Many benefits of operating system were discovered. Operating system increased system's performance and eased accomplishing of real-time requierments. Interrupt delay became almost constant and radio's on time increased from 80 % to 95 %. It clarified also the system structure. Resource management became more effective. At the same time the data memory footprint increased by 28 %.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston tietokonetekniikan laitoksella. Haluan kiittää työn ohjaajia ja tarkastajia prof. Marko Hännikäistä ja prof. Timo Hämäläistä. Haluan kiittää myös DI Ville Kasevaa, joka on ollut korvaamattomana apuna työtä suunniteltaessa ja tehtäessä. Lisäksi kiitän kaikkia tietokonetekniikan laitoksen työtovereita hyvän työympäristön luomisesta.

Erityiskiitokset perheelle ja tutuille tuesta ja kannustuksesta opiskelujen aikana.

Tampereella toukokuussa 2011,
Juha Onkila
Opiskelijankatu 4 A 30
33720 Tampere
Puh: +358 50 910 5784
E-mail: juha.onkila@iki.fi

SISÄLLYS

TERMIT JA SYMBOLIT

Termi	Selitys
DSR	Viivästetty palveluohjelma (engl. Deferred subroutine)
GPS	Maailmanlaajuinen satelliittipaikannusjärjestelmä (engl. Global positioning system)
ID	Tunniste (engl. identifier)
IPC	Prosessien välinen käyttöjärjestelmäviesti (engl. Inter process communication)
ISR	Keskeytyksen palveluohjelma (engl. Interrupt subroutine)
LAN	Lähiverkko (engl. Local area network)
LCD	Nestekidenäyttö (engl. Liquid crystal display)
MAC-kerros	Tiedonvälityskerros (engl. Medium access control)
OS	Käyttöjärjestelmä (engl. Operating system)
PDU	Protokollatietoyksikkö (engl. Protocol data unit)
PID	Prosessin tunnistenumero (engl. Process identifier)
POSIX	Unix-käyttöjärjestelmälle kehitetty käyttöjärjestelmärajapinta(engl. Portable operating system interface for unix)
RTOS	Reaaliaikainen käyttöjärjestelmä (engl. Real-time operating system)
TCP/IP	Internetin tietoliikenneprotokolla (engl. Transmission control protocol / internet protocol)
WLAN	Langaton lähiverkko (engl. Wireless local area network)
WPAN	Langaton henkilökohtainen verkko (engl. Wireless personal area network)
WMAN	Langaton kaupunkiverkko (engl. Wireless metropolitan area network)
WSN	langaton anturiverkko (engl. Wireless sensor network)
WWAN	Langaton laajaverkko (engl. Wireless wide area network)

1. JOHDANTO

Vielä 80-luvulla langattomat laitteet, kuten matkapuhelimet, olivat kooltaan suuria ja niiden käyttöaika oli suurikokoisista akuista huolimatta todella lyhyt. Tiedonsiirto oli tuolloin pääosin vielä analogista ja laitteissa käytettiin suuria määriä erilliskomponentteja. Elektroniikkalaitteet ovat ajan kuluessa digitalisoituneet ja samalla tiedonsiirto on muuttunut digitaaliseksi. Elektroniikan miniatyrisointi on kehittynyt ja tänä päivänä yhdellä mikropiirillä voi jo toteuttaa todella monimutkaisia kokonaisuuksia. Matkapuhelin itsessään sisältää enemmän laskentatehoa kuin eilispäivän tietokone ja aivan pienimmät ja virtapiheimmät mikroprosessoritkin ovat ominaisuuksiltaan vertailukelpoisia muutaman vuosikymmenen takaisten kotikoneiden prosessoreiden kanssa. Pienimpien prosessoreiden kehitys ja hintojen lasku on mahdollistanut elektroniikalle aivan uusia käyttökohteita ja samalla elektroniikkaa on tullut lähes kaikkialle ympärillemme. Lisäksi tiedonsiirto on muuttunut langattomaksi, mikä on tehnyt useista laitteista liikkuvia ja paristokäyttöisiä. Eräs langattoman tiedonsiirron ja pienimpien mikroprosessoreiden mahdollistama merkittävä sovellus on langaton anturiverkko. Langattoman anturiverkon merkittävyyttä kuvaa sen valinta yhdeksi kahdeksasta tulevaisuuden teknologiasta, jotka tulevat muuttamaan maailman [28].

Käyttöjärjestelmä on sanana lähes kaikille tuttu. Useimmiten se vie ajatukset PC-tietokoneiden maailmaan ja mieleen tulevia käyttöjärjestelmiä ovat lähinnä Windows ja Linux. Usein käyttöjärjestelmä käsitetään maallikoiden keskuudessa pelkäksi tietokonetta ohjaavaksi käyttöliittymäksi. Toisaalta se myös joissain tapauksissa sitä on, mutta käyttöjärjestelmällä on myös monia muita tehtäviä, kuten tietokoneen resurssien hallinta. Käyttöjärjestelmiä on kehitetty lähes kaikkiin mahdollisiin tietokonejärjestelmiin. Käyttöjärjestelmä löytyy säätilaennustusta suorittavasta supertietokoneesta kuten myös sulautetun järjestelmän sisältävästä pullonpalautusautomaatista.

Puhuttaessa pullonpalautusautomaatista puhutaan myös sulautetusta järjestelmästä, joka ohjaa laitteen toimintaa. Sulautettu järjestelmä on tiettyyn tarkoitukseen tehty tietokonejärjestelmä, jonka olemassaolo voi olla käyttäjälle tuntematon. Pullonpalautusautomaatin ohjaus sisältää todennäköisesti joitain reaaliaikavaatimuksia. Reaaliaikavaatimukset tarkoittavat ennalta määrättyjä suoritusaikoja. Reaaliaikaisuus ei välttämättä tarkoita toiminnon suorittamista mahdollisimman

nopeasti vaan tietyn, ennalta määrätyn ajan, kuluessa. Sovelluksen reaaliaikavaatimukset yhdistettyä sulautettuun järjestelmään tekevät järjestelmästä sulautetun reaaliaikajärjestelmän.

Käyttöjärjestelmä auttaa sulautetun reaaliaikajärjestelmän toteutuksesta, koska se huolehtii ajoituksien hallinnasta. Ajoitusten hallinta on erittäin hankalaa eritoten monimutkaisissa järjestelmissä. Sulautetetuissa reaaliaikajärjestelmissä käytettäviä käyttöjärjestelmiä kutsutaan reaaliaikaisiksi käyttöjärjestelmiksi (RTOS, engl. real-time operating system).

1.1 Langattomat anturiverkot

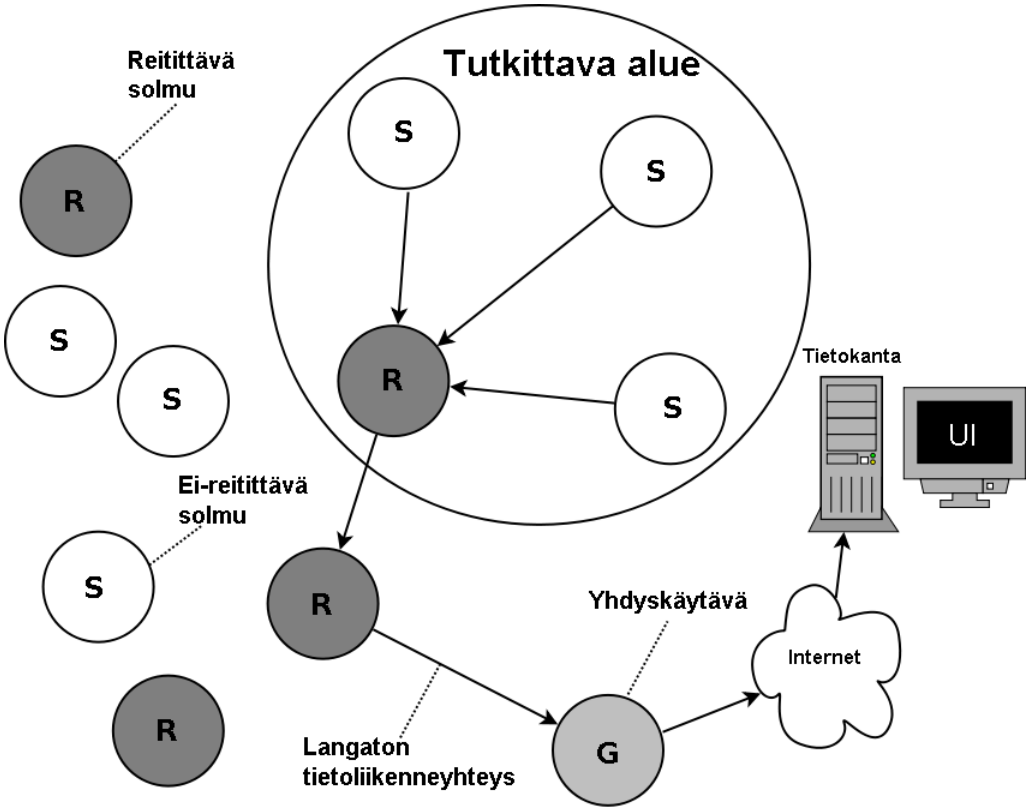
Langaton anturiverkko on nimensä mukaisesti antureista tai paremminkin anturilaitteista koostuva tietoverkko. Langattomat anturiverkot ovatkin olleet yksi suurimmista langattomaan teknologiaan suuntautuvan tutkimuksen kohteista viime vuosina. Elektroniikan miniatyrisoinnin kehitys, suorituskyvyn ja energiatehokkuuden kasvu ja protokollien kehitys on mahdollistanut tämä nopean kehityksen [23].

Termin langaton anturiverkon määrittäminen ei ole täysin yksiselitteinen asia. Esimerkiksi matkapuhelinverkot voidaan käsittää sanan laajimmassa merkityksessä jättimäisiksi langattomiksi anturiverkoksi. Nykyaikaiset matkapuhelimet mittaavat useita suureita ja tukiasemien sekä puhelimien GPS-vastaanottimien perusteella on lisäksi saatavilla paikannustietoa.

Pienemmässä mittakaavassa esimerkki langattomasta anturiverkosta on langattomia lämpötila-antureita sisältävä lämpömittari. Kuitenkin yleisesti ottaen puhuttaessa langattomista anturiverkoista tarkoitetaan pienistä, juuri anturiverkkoja varten suunnitelluista resurssirajoitteisista anturilaitteista muodostettuja langattomia verkkoja. Resurssirajoitteisuus tarkoittaa tässä tapauksessa rajoitteisuutta anturilaitteen energian ja samalla esimerkiksi laskentatehon ja muistin suhteen. Laitteet ovat yleensä paristokäyttöisiä ja niiden käyttöikä paristoilla voi olla jopa vuosia.

Langattomat anturiverkot ovat tyypillisesti topologialtaan mesh-tyyppisiä, mikä tarkoittaa, että verkko voi reitittää ilman ennalta määrättyä konfigurointia. Tässä työssä langattomilla anturiverkoilla tarkoitetaan nimenomaan edellä kuvattuja resurssirajoitteisia mesh-tyyppisiä anturiverkkoja.

Langaton anturiverkko on hajautettu järjestelmä, joka koostuu useasta itsessään sulautetun järjestelmän muodostavasta anturilaitteesta. Anturilaitteita kutsutaan myös anturisolmuiksi (engl. sensor node). Kuvassa 1.1 on esitetty tyypillinen anturiverkko, joka koostuu anturisolmuista ja reittien päätepisteistä, joita kutsutaan yhdyskäytäviksi. Anturilaitteet muodostavat ja ylläpitävät automaattisesti verkon, josta tietoa kerätään yhdyskäytävän avulla. Yhdyskäytävä muodostaa anturiverkon yhdyskäytävän muihin tietoverkkoihin. Yhdyskäytävä voi olla esimerkiksi internetin välityksellä yhteydessä tietokantaan, johon tallennetaan anturiverkon tuottamat tie-



Kuva 1.1: Langattoman anturiverkon toimintaperiaate. Tieto kulkee ei-reitittävältä solmulta reitittävälle, josta se kulkeutuu reitittävien solmujen kautta yhdyskäytävälle ja muihin tietoverkkoihin.

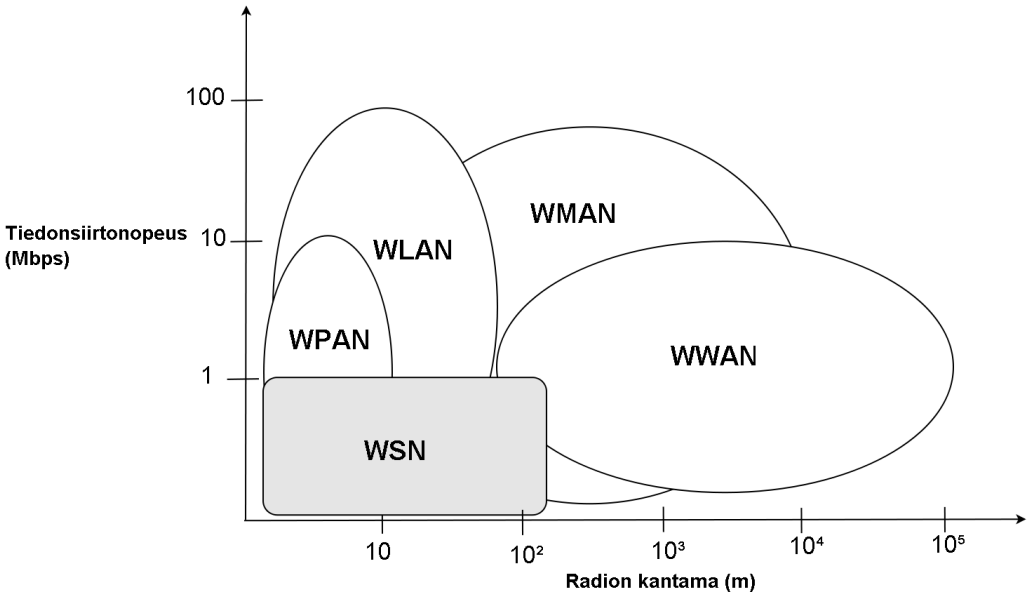
dot. Langattoman anturiverkon toimintaa voi olla mahdollista tarkkailla ja ohjata etänä esimerkiksi käyttöliittymäohjelman avulla.

Kuvassa 1.2 on kuvattu langattomien anturiverkkoja suhteessa muihin langattomiin tietoliikenneteknologioihin [30]. Kuvassa on luokiteltu eri verkkoteknologioita niiden radion kantaman ja tiedonsiirtonopeuden mukaan.

Langaton kaupunkiverkko (WMAN, engl. Wireless Metropolitan Area Network) ja langaton laajaverkko (WWAN, engl. Wireless Wide Area Network) omaavat laajimman radiokantaman yhdellä lähettimellä. WWAN-verkkoja käytetään pääasiassa digitaalisissa matkaviestinverkoissa langattomaan äänen ja datan siirtoon [36]. WMAN-verkkoa, kuten WiMAX:ia, käytetään laajakaistaisiin tietoliikenneyhteyksiin kodeissa ja pienissä toimistoissa [19].

Langatonta lähiverkkoa (WLAN, engl. Wireless Local Area Network) käytetään laajentamaan ja korvaamaan langallista lähiverkkoa (LAN, engl. Local Area Network) [17]. Langatonta henkilökohtaista verkkoa (WPAN, engl. Wireless Personal Area Network) käytetään lähinnä johtojen korvaamiseen kulutuselektroniikassa. Esimerkki tunnetusta WPAN-teknologiasta on IEEE 802.15.1-2002, joka tunnetaan paremmin nimellä Bluetooth v1.1 [5].

Langattomat anturiverkot (WSN, engl. Wireless Sensor Network) on tarkoitet-



Kuva 1.2: Langattomien anturiverkkojen (WSN) sijoittuminen langattomien tietoliikenneteknologioiden joukossa.

tu sovelluksiin, joissa tarvitaan todella pientä energiankulutusta, mutta pieni tiedonsiirtonopeus on riittävä [24]. Radion kantama voi kuitenkin eri verkkojen välillä vaihdella muutamista metreistä jopa satoihin metreihin. Lisäksi verkko voi koostua jopa tuhansista anturilaitteista, mikä mahdollistaa pinta-alaltaan suuret verkot. Langattomat anturiverkot eroavat esimerkiksi WLAN- ja WPAN-tekniikoista sovelluskeskeisyyden, itseorganisoitavuuden ja laajuuden perusteella. Lisäksi langattomassa anturiverkossa keskitytään sovelluksen toteuttamiseen pelkän datayhteyden sijaan.

1.2 Langattomien anturiverkkojen ominaisuudet

Langaton anturiverkko on **sovelluskeskeinen** tietoliikenneverkko [30]. Sovelluskeskeisyys näkyy siinä, että langaton anturiverkko on rakennettu tietyn tehtävän toteuttamiseksi. Tehtävä voi koostua useista sovelluksista, joita anturiverkko suorittaa. Sovelluskeskeisyys tekee jokaisesta anturiverkosta monella tapaa itsessään uniikin. Verkon komponentit kuten tietoliikenneprotokollat ja laitealustat palvelevat sovellusta. Laitealustat voivat vaihdella jopa saman verkon sisällä. Yleiskäyttöisen anturiverkon luominen on mahdotonta käyttötapojen ja kohteiden monimuotoisuuden vuoksi.

Langaton anturiverkko on toisin kuin perinteiset tietoliikenneverkot perusluonteeltaan **datakeskeinen** [30]. Tämä tarkoittaa sitä, että anturilaitteiden tunnistenumeroa tärkeämpiä seikkoja ovat laitteen sijainti ja sen tuottama mittaustieto. Esimerkiksi kerättäessä ilmankosteustietoja maastosta on yhdentekevää, mikä mittaustiedon tuottaneen anturilaitteen tunnistenumero on. Tärkeämpää on tietää, mis-

tä sijainnista kyseinen mittaustieto tulee ja milloin se on mitattu.

Langattomat anturiverkot ovat perusluonteeltaan **dynaamisia** [30]. Jopa kiinteästi asennetun anturiverkon toiminta voi olla ennalta-arvaamatonta. Ympäristön muuttumisen ja esimerkiksi muiden anturilaitteiden virheellisen toiminnan takia nodejen välinen tietoliikenneyhteys ei ole stabiili, mikä vaatii dynaamista toimintaa. Toisaalta verkot itsessään ovat jo dynaamisia. Niiden koko voi vaihdella ja esimerkiksi anturilaitteet voivat olla liikkuvia. Langaton anturiverkko voi koostua jopa tuhansista erittäin tiheään sijoitetusta anturilaitteesta, mikä eroaa muista langattomista verkkoteknologioista.

Anturilaitteet ovat tyypillisesti paristokäyttöisiä ja pienikokoisia, ja niiden toimintaa ohjaa vain vähän laskentatehoa sekä data- ja ohjelmamuistia omaava mikrokontrolleri. Anturilaitteen eniten energiaa kuluttava osa on tyypillisesti radio, joten radioliikenteen määrän minimoimisella pystytään elinikä maksimoimaan [23]. Anturilaitteet ja siten langattomat anturiverkot ovatkin **resurssirajoitteisia** energia-, muisti-, tietoliikenne- ja laskentaresurssien suhteen [30]. Laitteen koko rajoittaa lisäksi paristojen tai energian keräimen, kuten aurinkopaneelin, kokoa.

Anturinodet voidaan sijoittaa satunnaisesti hyvinkin ankariin olosuhteisiin. Anturilaite voi olla kiinteänä osana jotain toista laitetta tai sijoitettuna paikkoihin, joissa mittaaminen perinteisillä tavoilla on vaikeaa. Niiden ylläpito tai laitteiden vaihtaminen voi olla hyvin vaikeaa ellei jopa lähes mahdotonta. Toisaalta esimerkiksi sovellusvaatimukset voivat muuttua, joten ylläpitoa kuitenkin tarvitaan. Langattomien anturiverkkojen **käyttöönotto** eroaakin ratkaisevasti muista tietokonejärjestelmistä.

1.3 Langattomien anturiverkkojen hyödyntäminen

Langattomien anturiverkkojen on spekuloitu tuovan esille uusia ilmiöitä, joista ei muilla mittausjärjestelmillä ole aikaisemmin ollut havaintoa [33]. Langaton anturiverkko tarjoaa helpon mittaamisen lisäksi täysin uuden tavan mitata.

Langattomilla anturiverkoilla on lukuisia sovelluksia mitä moninaisimmilla aloilla. Laajasta sovellusalasta huolimatta anturiverkkojen sovellusten tehtävät voidaan jakaa neljään luokkaan [30]:

Monitorointi Anturiverkkoa käytetään jonkin suureen tai parametrin seuraamiseen jollain alueella tai tietyssä sijainnissa. Sovellus toteutetaan tyypillisesti jaksollisesti toistuvilla mittauksilla. Tyypillinen sovellusesimerkki on lämpötilan mittaaminen tietyltä alueelta.

Tapahtuman huomaaminen Anturiverkkoa voidaan käyttää lähes automaattisena järjestelmänä jonkin tapahtuman huomaamiseen. Sovellus voi olla toteutettu esimerkiksi jaksollisesti toistuvilla mittauksilla, joista voidaan tiettyjen raja-arvojen perusteella tunnistaa tapahtuman ilmeneminen. Tyypillinen sovellusesimerkki on

jonkin laitteen liikkeen tunnistaminen kiihtyvyysanturin mittaustietojen avulla.

Kohteen tunnistaminen Anturiverkkoa voidaan käyttää kohteen tai jonkin tapahtuman tunnistamiseen. Tyypillisesti tunnistamiseen tarvitaan useita mittaustietoja ja ulkoista tietojen prosessointia. Tyypillinen sovellusesimerkki on tulipalon tunnistaminen häkä- ja happiantureiden avulla.

Kohteen paikantaminen Anturiverkkoa voidaan käyttää kohteiden paikantamiseen. GPS (engl. Global Positioning System) toimii vain ulkotiloissa, joten anturiverkko on toimiva ratkaisu sisätilapaikannukseen. Tyypillinen sovellusesimerkki on henkilökunnan ja potilaiden paikantaminen sairaaloissa.

1.4 Anturilaitteen ohjelmisto ja laitteisto

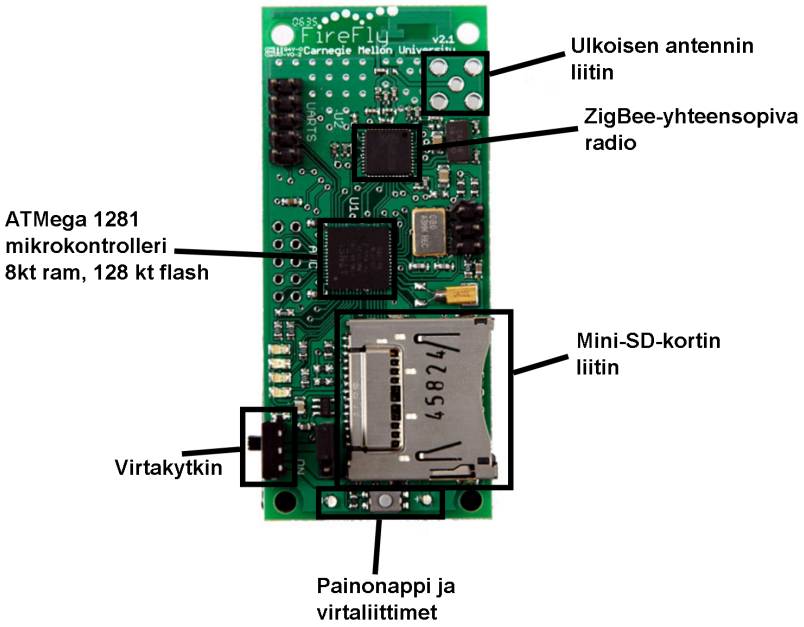
Anturilaite on itsessään reaaliaikavaatimuksia omaava sulautettu reaaliaikajärjestelmä. Langattomalle anturiverkolle asettuu reaaliaikaisuusvaatimuksia sekä anturilaitteiden sisäisen toiminnan suhteen että tiedon etenemisviiveeseen anturilta käyttäjälle.

Käyttöjärjestelmää käyttävän anturilaitteen tyypillinen protokollapino on esitetty kuvassa 1.4. Ohjelmistopino koostuu seitsemästä eri ohjelmistokerroksesta käyttöjärjestelmä mukaan lukien. Kerrosjako ei ole ehdoton. Anturilaitteen protokollapinon pohjalla on resurssirajoitteinen laitteistokerros. Se abstrahoi laitteiston hallinnan tarjoamalla rajapinnan laitteiston ohjaamisen muille kerroksille. MAC-kerros (engl. Media Access Control) tai paremminkin MAC-protokolla huolehtii tiedonvälityksestä. Se kontrolloi kanavan käyttöä laitteistokerroksen avulla.

Reitityskerros huolehtii verkon organisoitumisesta ja datan välittämisestä lähteeltä yhdyskäytävälle. Se muodostaa, pitää yllä ja valitsee reitin kullekin lähetettävälle paketille. Kuljetuskerros huolehtii vuon hallinnasta ja virheiden tunnistamisesta. Usein kuljetuskerros on kuitenkin hyvin suppeana osana lähinnä reitityskerrosta, sillä esimerkiksi päästä päähän ulottuvaa vuon hallintaa on harvemmin anturiverkoissa toteutettu resurssirajoitteista johtuen.

Väliohjelmisto abstrahoi sovelluskerrosta varten alempien kerroksen toiminnallisuuden ja tarjoaa ohjelmointirajapinnan (API, engl. application programming interface) sovelluksille. Se toimii eräänlaisena järjestelmäohjelmistona, joka hallinnoi anturilaitetta ja koko anturiverkkoa [28]. Väliohjelmisto voi esimerkiksi päättää jonkin paljon prosessoriaikaa ja samalla energiaa kuluttavan laskennan suorittamisesta toisella anturilaitteella. Väliohjelmisto nähdään usein osana sovelluskerrosta. Protokollapinon päällimmäisenä toimii sovelluskerros, joka suorittaa anturilaitteen varsinaisen toiminnallisuuden sovellusten muodossa.

Kuvassa 1.3 on nanoRK firefly-alusta [2], joka kuvaa tyypillistä anturilaitteen laitteistoa. Alusta koostuu minimissään mikrokontrollerista ja radiosta, mutta saattaa sisältää monia muitakin laitteita. Anturilaitteen laitteiston sydämenä on perin-



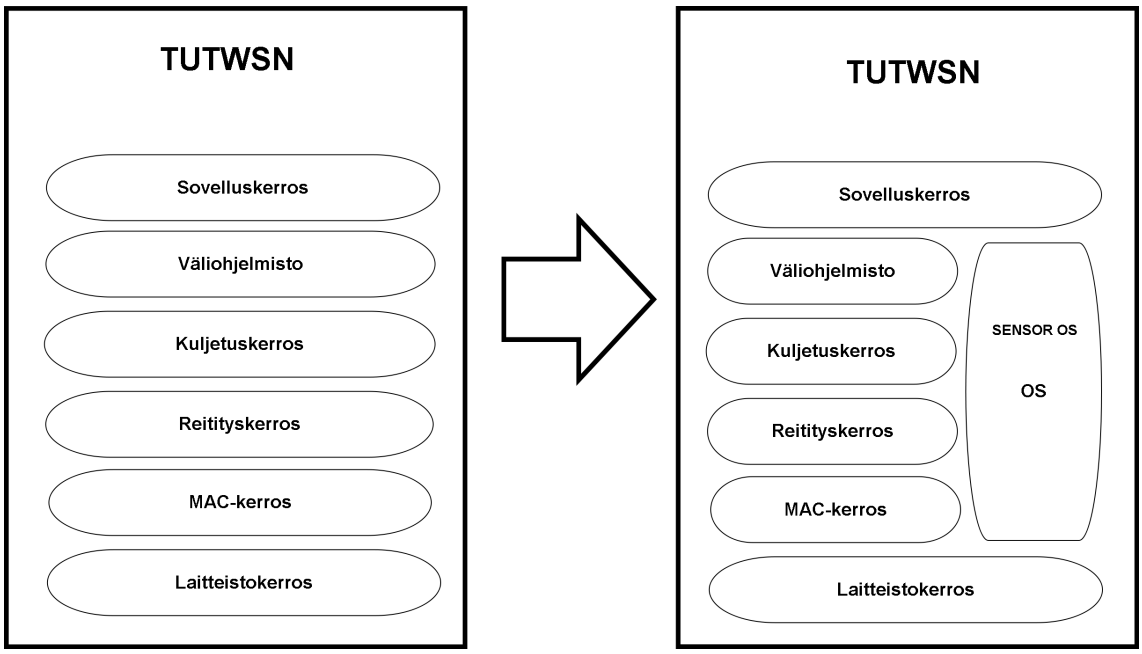
Kuva 1.3: NanoRK Firefly-anturilaitealusta. Anturilaite muodostuu pääasiassa mikrokontrollerista ja radiosta. Firefly-alustaan voi lisäksi liitäntäkortilla liittää esimerkiksi lämpötila-, kiihtyvyys, ääni- ja patterijänniteanturin [2].

teisesti edullinen ja todella resurssirajoitteinen mikrokontrolleri [23]. Mikrokontrolleri tarjoaa riittävän monipuolisen toiminnallisuuden anturilaitteen toteuttamiseen virrankulutuksen ollessa silti pieni. Toisaalta resurssirajoitteinen mikrokontrolleri on yksi anturilaitteen suurimmista rajoitetekijöistä etenkin muisti- ja laskentaresurssien rajallisuudesta johtuen. Mikrokontrolleri ohjaa muuta laitteistoa, joka koostuu esimerkiksi erilaisista antureista, tehojärjestelmän muodostavista energialähteistä ja regulaattoreista ja langattoman tiedonsiirron mahdollistavasta digitaalisesta radiosta.

1.5 Diplomityön sisältö

Teknologian kehitys on parantanut anturiverkkojärjestelmiä, mutta toisaalta monimutkaisuus on lisääntynyt. Muun muassa monimutkaisuuden hallintaan on kehitetty erityisesti anturiverkoille sopivia käyttöjärjestelmiä. Anturiverkoille on tiettyjä erityisvaatimuksia, jotka estävät tavallisille sulautetuille järjestelmille tai reaaliaikajärjestelmille tarkoitettujen käyttöjärjestelmien käytön.

Tässä työssä tutkitaan käyttöjärjestelmän käyttöönottoa ja sen hyötyjä langattomissa anturiverkoissa. Kuvassa 1.4 on kuvattu toteutettu muutos. Työssä muutetaan anturinoden sulautettu ohjelmisto käyttöjärjestelmän päällä suoritettavaksi. Prototyyppialustana toimii Tampereen teknillisen yliopiston tietokonetekniikan laitoksella kehitetty matalan viiveen TUTWSN [34] ja käyttöjärjestelmänä tutkimus-



Kuva 1.4: Tässä työssä muutetaan TUTWSN-anturiverkon anturilaitteen sulautettu ohjelmisto SensorOS-käyttöjärjestelmää käyttäväksi

käytössä kehitetty SensorOs [26].

Työn tavoitteina ovat olleet ohjelmiston rakenteen yksinkertaistaminen ja järjestelmän suorituskyvyn parantaminen. Käyttöjärjestelmän avulla periaatetasolla yksinkertaisesta järjestelmästä on tarkoitus tehdä kooditasolla selkeämpi. Lisäksi käyttöjärjestelmän avulla on mahdollista parantaa järjestelmän suorituskykyä varsinaisen käyttöjärjestelmän aiheuttamista resurssivaatimuksista huolimatta. Lisäksi SensorOs-käyttöjärjestelmää ei toistaiseksi ole todellisissa anturiverkoissa käytetty, joten todellisista testeistä on itse käyttöjärjestelmän kehitykselle hyötyä. Tietoa käyttöjärjestelmän eduista ja käyttöönoton vaatimuksista langattomissa anturiverkoissa löytyy lisäksi hyvin vähän.

Diplomityössä on seitsemän lukua. Johdannon jälkeisessä toisessa luvussa on käsitelty anturiverkkojen käyttöjärjestelmiä ja niiden käyttöönottoa ja siihen liittyviä vaatimuksia. Kolmannessa luvussa tutustutaan prototyyppisensoriverkkona toimivaan TUTWSN-teknologiaan eritoten sulautetun ohjelmiston kannalta. Neljännessä luvussa on kuvattu käyttöjärjestelmän käyttöönoton vaatineita muutoksia. Viidennessä luvussa suoritetaan vertailuja käyttöjärjestelmällisen ja ilman käyttöjärjestelmää toimivan järjestelmän välillä. Kuudes ja samalla viimeinen luku on yhteenvetoa ja loppupäätelmiä varten.

2. KÄYTTÖJÄRJESTELMÄT LANGATTOMIIN ANTURIVERKKOIHIN

Perinteisesti käyttöjärjestelmä helpottaa sovellusohjelmointia, tehostaa resurssien käyttöä ja helpottaa laitteiston käyttöä [18]. Anturilaitteet omaavat harvemmin mitään monimutkaista käyttöliittymää, joten käyttöjärjestelmää käytetään lähinnä sovellusohjelmoinnin helpottamiseksi ja resurssien käytön tehostamiseksi. Langattomien anturiverkkojen erityisvaatimukset resurssien ja reaaliaikaisuuden suhteen tuovat vaatimuksia myös käyttöjärjestelmälle. Tästä syystä langattomille anturiverkoille on suunniteltu vain tätä tarkoitusta varten suunniteltuja käyttöjärjestelmiä [30]. Useat anturiverkot on kuitenkin toteutettu ilman erillistä käyttöjärjestelmää, mutta sovellusten määrän ja monimutkaisuuden kasvu on lisännyt tarvetta käyttöjärjestelmän hyödyntämiselle [30].

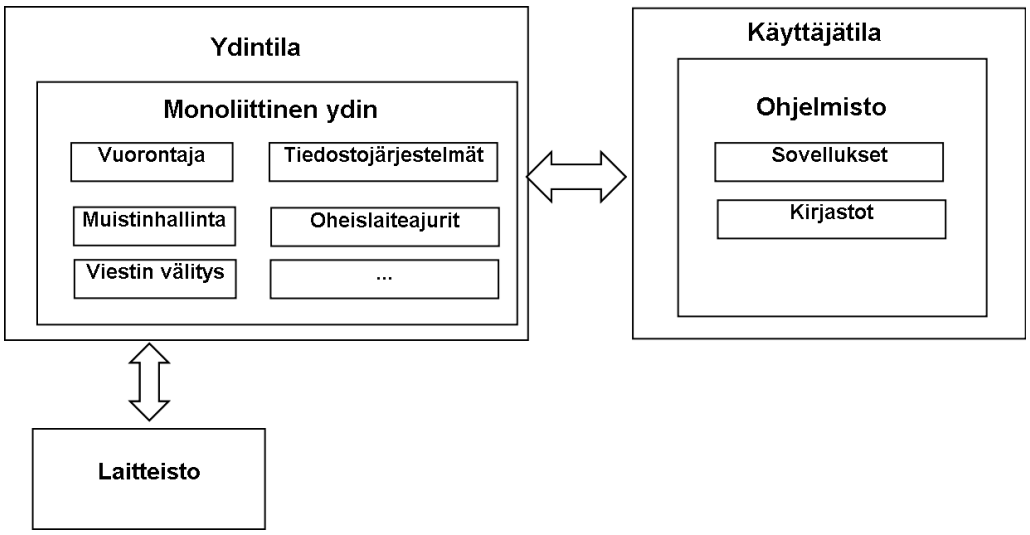
Anturiverkkojen käyttöjärjestelmän ytimenä toimii yleensä reaaliaikainen käyttöjärjestelmäydin [26]. Siksi tässä luvussa perehdytään aluksi reaaliaikaisten käyttöjärjestelmien perusteisiin. Tämän jälkeen on perehdytty anturiverkkojen käyttöjärjestelmien eri toteutusvaihtoehtoihin. Lisäksi luvussa kerrotaan käyttöjärjestelmän eduista ja sen valintaan liittyvistä kysymyksistä. Lopuksi on esitelty muutama tyypillinen käyttöjärjestelmä, sekä tässä työssä käytetty SensorOS. Esimerkkikäyttöjärjestelmät on valittu eniten käytettyjen sensoriverkkokäyttöjärjestelmien joukosta edustamaan eri näkökulmia käyttöjärjestelmän toteuttamiseen.

2.1 Reaaliaikainen käyttöjärjestelmäydin

Reaaliaikakäyttöjärjestelmä on käyttöjärjestelmä, jolla on tiettyjä reaaliaikavaatimuksia toiminnan suhteen. Ymmärtääkseen käyttöjärjestelmän tarjoamia etuja tai sen käyttöönottoa tulee ymmärtää jotain käyttöjärjestelmään liittyvää perustermistöä ja toteutustekniikoita. Seuraavassa on esitetty reaaliaikajärjestelmien toimintaperiaatteita ja toteutustapoja.

Käyttöjärjestelmä jakaantuu *ytimeen* (engl. kernel) ja *käyttäjän prosesseihin* [37]. Usein puhutaan myös *ydintilasta* (engl. kernel space) ja *käyttäjätilasta* (engl. user space). Tilojen ero tulee muistinhallinnasta: ytimen ohjelmisto pääsee käsiksi kaikkeen järjestelmän muistiin, kun taas käyttäjätilan ohjelmat vain omaansa. Eri tilojen välillä tapahtuva kommunikaatio tapahtuu *järjestelmäkutsuilla* (engl. system call).

Ydin on käyttöjärjestelmän keskeisin komponentti ja se pitää yllä järjestelmän

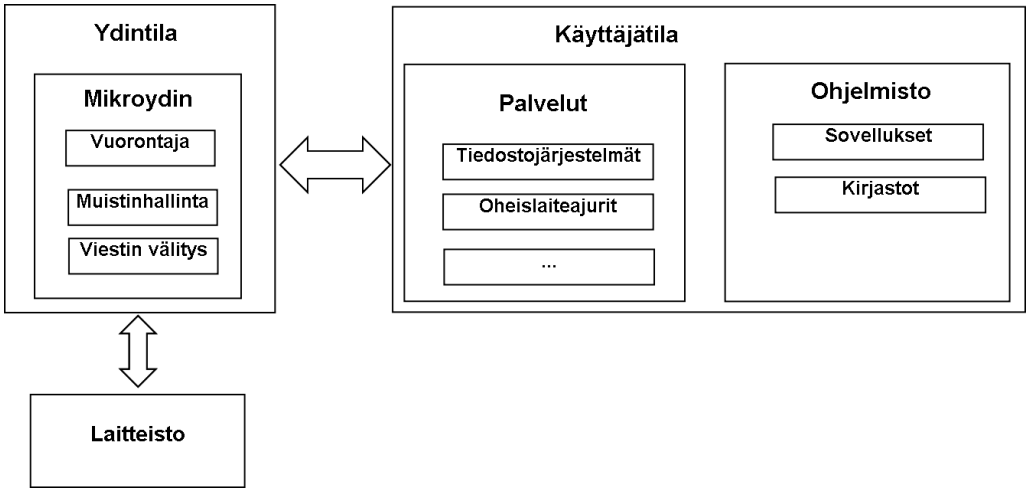


Kuva 2.1: Monoliittisen ytimen periaate. Ydintila koostuu välttämättömien toimintojen lisäksi esimerkiksi oheislaiteajureista.

perustoimintoja. Se mitä näihin toimintoihin lasketaan kuuluvan riippuu ytimen toteutustavasta. Kuvassa 2.1 esitetty *monoliittinen* ydin sisältää minimitoimintojen lisäksi kaikki käyttöjärjestelmän perustoiminnot. Sulautetuissa järjestelmissä tämä näkyy esimerkiksi oheislaitteiden ohjauksen sisältymisenä ytimeen [32]. Tämän toteutustavan etuna on nopeus, mutta heikkoutena heikko luotettavuus ja muokattavuus. Esimerkiksi oheislaitemoduulissa tapahtunut virhe voi kaataa koko käyttöjärjestelmän.

Monoliittisen ytimen vastakohta on kuvassa 2.2 esitetty *mikroydin*, jossa oheistoiminnot ovat omina moduuleinaan osana käyttäjätilaa [37] [32]. Itse ydin sisältää vain välttämättömimmät peruskomponentit. Moduuleita voi lisätä tai poistaa ja tästä syystä mikroydin tarjoaa monoliittistä ydintä paremman muokattavuuden. Mikroytimellisestä käyttöjärjestelmästä saadaan räätälöityä vain ja ainoastaan tarvitut ominaisuudet sisältävä ja resurssien käyttö tehostuu. Moduulien sijainti käyttäjätilassa tarjoaa myös paremman luotettavuuden. Moduulissa tapahtunut virhe kaataa vain kyseisen moduuliin koko käyttöjärjestelmän sijaan.

Varsinaisten sovellusten suoritus koostuu tehtävistä, joiden suoritusta kontrolloi ytimen komponentti nimeltään *vuorontaja* (engl. scheduler). Vuorontaja päättää mikä tehtävä on kulloinkin suorituksessa. Kuvassa 2.3 on esitetty tehtävien tilamalli [18] [8]. Tehtävien tila vaihtuu tehtävien itsensä tai vuorontajan toimesta. Suorituksessa-tilassa olevat tehtävät ovat sillä hetkellä suorituksessa. Valmiina suoritukseen- tilassa ovat tehtävät, jotka ovat valmiita suoritukseen. Ainoa resurssi mitä ne odottavat on prosessoriaika. Odottaa-tilassa ovat tehtävät, jotka odottavat jotain tapahtumaa, kuten ajastimen laukeamista tai IO-operaatiota. Vuorontaja valitsee seuraavaksi suoritukseen tulevan tehtävän valmiina suoritukseen-tilassa



Kuva 2.2: Mikroytimen periaate. Ydintila koostuu vain välttämättömistä toiminnoista. Esimerkiksi oheislaiteajurit suoritetaan käyttäjätilassa.

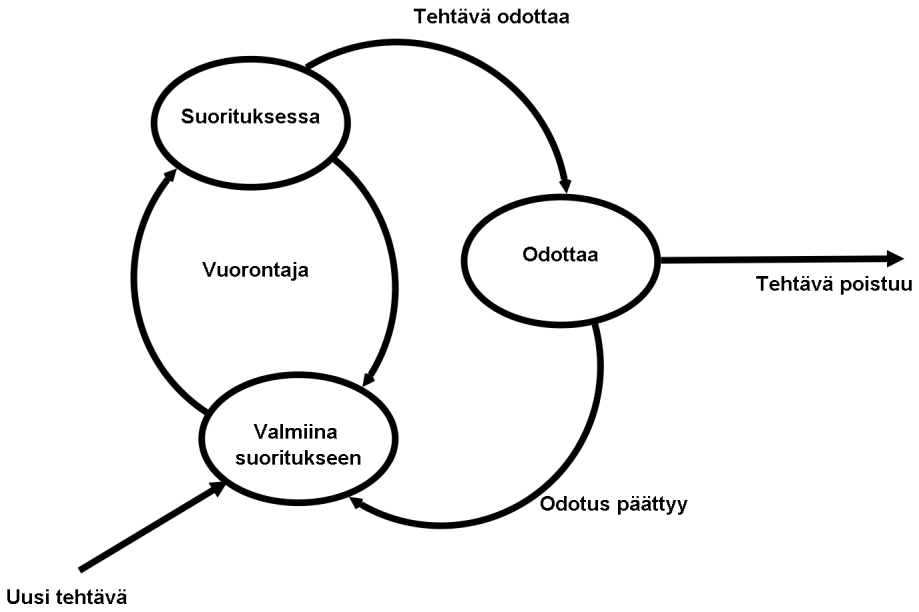
olevista tehtävistä.

Vuorontaja voi olla *keskeyttävä* (engl. pre-emptive) tai *vuoropohjainen* (engl. cooperative). Vuoropohjaisessa vuorontamisessa tehtävä itse antaa suorituksen vuoron seuraavalle tehtävälle, kun taas keskeyttävässä vuorontamisessa vuorontaja voi keskeyttää ajossa olevan tehtävän suorituksen ja vaihtaa jokin muu tehtävä suoritukseen. Suorituksen vaihto kesken tehtävän suorituksen aiheuttaa kustannuksia, sillä keskeytetyn tehtävän tila tulee tallettaa.

Vuorontaja voi olla lisäksi *tapahtumaohjattu* (engl. event-driven) tai *suoritusaikaa jakava* (engl. time-sharing). Tapahtumaohjatussa tehtävien suoritusta kontrolloi tehtävien prioriteetit ja tapahtumat. Suoritusaikaa jakavassa toteutuksessa vuorontaja vaihtaa tehtävää kellokeskeytysten mukaan.

Tehtävät ovat tyypillisesti *prosesseja* tai *säikeitä*, mutta myös muita suoritusyksiköitä on olemassa. Prosessi on ohjelman osa, joka sisältää ohjelmakoodin lisäksi tietoja ohjelmaosan tilasta [35]. Tilaan sisältyy muun muassa prosessorin tila sekä prosessipino, johon tallennetaan funktioparametrit, paluuosoitteet sekä väliaikaiset muuttujat. Prosessin tilan sisältävää tietotyyppiä kutsutaan prosessielementiksi (PCB, engl. process control block). Koska prosesseilla on oma muistiavaruus ja tieto prosessorin tilasta, ne ovat muista prosesseista riippumattomia. Oman muistiavaruuden ansiosta prosesseilla on aina suuri kontekstin vaihdosta aiheutuva kustannus.

Säie on vastaava ohjelmakokonaisuus kuin prosessi, mutta säikeet käyttävät prosesseista poiketen samaa muistiavaruutta [35]. Säikeet voivat lisäksi jakaa myös muita resursseja keskenään. Kontekstin vaihdossa tarvitsee vaihtaa vain prosessorin tila ja esimerkiksi suorituspino, mutta muistioperaatioiden määrä on huomattavasti prosessia pienempi yhteisestä muistialueesta johtue. Säikeet ovat usein prosessin osa-



Kuva 2.3: Tehtävien tilamalli. Tehtävä vaihtelee tilojen välillä joko vuorontajan niin vaatiessa tai omatoimisesti.

sia ja yksi prosessi voi sisältää useita säikeitä.

Kevyempiä suoritusyksiköitä ovat *vuorottaisrutiinit* (engl. coroutines) [9] ja *protosäikeet* [13]. Molemmilla yksiköillä on käytössä sama pino ja kummassakaan ei kontekstin vaihdon yhteydessä esimerkiksi paikallisten muuttujien tila säily. Molempia käytetään lähinnä kompleksisuuden hallintaan.

Käyttöjärjestelmä koostuu vuorontajan lisäksi useista erilaisista palveluista. Reaaliaikainen käyttöjärjestelmä tarjoaa tyypillisesti seuraavia palveluita [8]:

Synkronointi Synkronointi toteutetaan tavallisesti niin sanottujen semaforien avulla. Semaforit ovat muuttujia, joilla voidaan kontrolloida jonkin jaetun resurssin käyttöä. Semafori sisältää tiedon resurssin vapaana olosta sekä resurssin odottajien määrästä. Semaforin erikoistapaus on mutex-muuttuja (engl. mutual exclusion, poissulkeminen). Mutexin arvon ollessa yksi resurssi on varattu ja arvolla nolla resurssi on käytettävissä. Mutex ei siis sisällä tietoa odottajien määrästä.

Viestin välitys Vaikka synkronointi itsessään on eräänlaista tiedon välitystä tehtävien välillä, tarvitaan myös viestipalveluja, joissa viesti sisältää jonkin tietosisällön. Tätä varten käyttöjärjestelmissä on niin sanotut postilaatikot. Käyttöjärjestelmä hoitaa viestijonojen ylläpitämisen ja viestien välittämisen oikealle tehtävälle. Usein puhutaan niin sanotuista IPC-viesteistä (engl. Inter Process Communication) eli prosessien välisistä viesteistä.

Tapahtumat Tapahtumamekanismi on binäärityyppinen tapa tehtävien väliseen

viestin välitykseen. Tapahtuma sisältää tyypillisesti vain tiedon siitä, onko jokin tapahtuma sattunut. Tapahtumalippujen avulla tehtävät voivat kontrolloida suoritustaan jäämällä odottamaan tiettyä tapahtumaa.

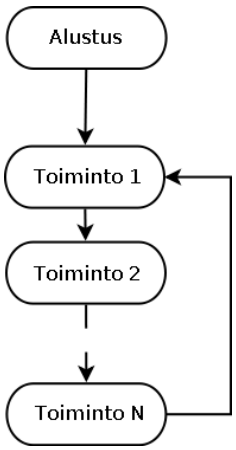
Laskimet Laskimet pitävät yllä tietoa siitä, kuinka monta kertaa tietty tapahtuma on ilmentynyt.

Ajastimet Ajastimien avulla voidaan ajoittaa tehtävien suoritusta. Ajastimien avulla tehtävän suoritus voidaan esimerkiksi keskeyttää halutun pituiseksi ajaksi. Ajastin on periaatteessa laskin, jonka tapahtumalähteenä on kello.

Muistinhallinta Käyttöjärjestelmä sisältää tyypillisesti toteutuksen dynaamiselle muistinhallinnalla. Järjestelmä sisältää tavallisesti kahdentyyppistä muistia: data- ja ohjelmamuistia. Ohjelmamuisti sisältää ohjelman käskyt ja datamuisti säilöö käsiteltävän ja käsitellyn tiedon. Käyttöjärjestelmän muistinhallinta tarkoittaa käytännössä datamuistin hallintaa. Käyttöjärjestelmällä on tietty määrä datamuistia käytössä, jonka käyttöä se kontrolloi. Muistia ei varata useimmiten tavu kerrallaan, vaan suuremmissa yksiköissä. Täysin dynaamista muistinhallintaa ei suositella käytettävän reaaliaikaisissa käyttöjärjestelmissä, koska sen käyttäytyminen voi olla arvaamatonta [6]. Muistinvarausyksikön koon määrittäminen on tärkeä osa järjestelmän suunnittelua, sillä huonosti valittu koko voi tehdä muistinhallinnasta tehotonta [7].

Resurssien hallinta Käyttöjärjestelmän vastuulla on tyypillisesti laitteistoresurssien hallinta. Käyttöjärjestelmä huolehtii tavallisesti laiteajureista ja kontrolloi laitteiden käyttöä. Lisäksi käyttöjärjestelmän vastuulla on suoritinajan jakaminen.

Keskeytysten hallinta Keskeytysten hallintaan käyttöjärjestelmä tarjoaa yleensä rajapinnan, johon kuuluu etenkin *keskeytyksen palveluohjelman* (ISR, engl. Interrupt Subroutine) ja keskeytysprioriteettien määrittäminen sekä keskeytysten salliminen/kieltäminen. Joissain käyttöjärjestelmissä on mahdollista luoda viivästettyjä palveluohjelmia (DSR, deferred subroutine), joissa suoritetaan keskeytyksen aiheuttamia enemmän aikaa vieviä toimenpiteitä. Viivästetyn palveluohjelman suoritus ei välttämättä keskeytä muun ohjelman suoritusta. Lisäksi joissain käyttöjärjestelmissä on oma pinonsa keskeytyksille, sillä muuten suorituspinossa jouduttaisiin ottamaan huomioon pahin mahdollinen tilanne eli käytännössä jättämään pinoon varaa mahdollisen keskeytyksen palveluohjelman verran.



Kuva 2.4: Tyypillinen silmukkarakenteisen järjestelmän suoritussilmukka. Alustusten jälkeen suoritetaan toimintoja ikuisessa silmukassa.

Käyttöjärjestelmän eri osia ja palveluita käytetään käyttöjärjestelmän tarjoamaa *ohjelmointirajapintaa* (API, engl. Application programming interface) käyttäen. Eräs erittäin käytetty rajapintastandardi on POSIX (engl. Portable Operating System Interface [for Unix]) [27]. POSIX määrittelee rajapintafunktiot käyttöjärjestelmän käyttöön sisältäen rajapintafunktiot mm. prosessien, semaforien, ajastimien ja muistinhallinnan käyttöön. Käyttämällä samaa ohjelmointirajapintaa on sovellusten siirtäminen toiseen järjestelmään helpompaa.

2.2 Reaaliaikaisen käyttöjärjestelmän käytön edut

Kaikissa reaaliaikaisissa sulautetuissa järjestelmissä ei käytetä käyttöjärjestelmää. Tällöin käytössä on usein kuvan 2.4 esittämä niin sanottu silmukkarakenne, jossa alustustoimenpiteiden jälkeen suoritetaan ikuisessa silmukassa järjestelmän toimintoja. Rakennetta kutsutaan myös nimellä pollaava ydin [32]. Mikäli silmukasta poistutaan keskeytysten sattuessa puhutaan niin sanotusta keskeytysohjatusta ytimestä.

Silmukkarakenteen hyvänä puolena on sen yksinkertaisuus ja keveys. Lisäksi rakenne on usein optimoitu järjestelmää varten joten suorituskyky on myöskin hyvä. Vaikka silmukkarakenne itsessään onkin hyvin yksinkertainen, voi järjestelmän toteutus käydä monimutkaiseksi järjestelmän koon kasvaessa. Järjestelmän kehitystyö jatkuu usein varsinaisen käyttöönoton jälkeen virhekorjauksien ja ominaisuuksien lisäämisen muodossa, joten lopullinen järjestelmä ei aina ole se, mitä on aluksi suunniteltu. Hyvin ja selkeäksi suunnitellusta järjestelmästä voi ajan saatossa tulla vaikeasti ymmärrettävä ja sekava.

Reaaliaikaisten käyttöjärjestelmien käyttö pienissä mikrokontrollereissa tuo seuraavia etuja silmukkarakenteeseen nähden[6], [14]:

Ohjelmistokehityksen tehostaminen Sulautettu järjestelmä on suunniteltaes-

sa usein hyvin organisoitu ja selkeä. Kuitenkin järjestelmän kehittyessä ominaisuuksien lisääntyessä ja virheitä korjattaessa järjestelmästä tulee usein monimutkainen ja sekava [16]. Käyttöjärjestelmä tarjoaa hajota ja hallitsetyyppisen ratkaisun monimutkaisuuden hallintaan. Käyttöjärjestelmää käytettäessä ohjelma pystytään jakamaan pieniksi ja helposti hallittavissa oleviksi säikeiksi ja prosessiksi, jotka taasen voidaan jakaa eri ohjelmistokehittäjien kesken. Modulaarisuuden lisääntyessä myös uudelleenkäyttömahdollisuudet lisääntyvät. Ohjelmistokehityksen tehostaminen laskee kehityskustannuksia.

Parempi ja turvallisempi synkronointi Sulautetuissa järjestelmissä käytetään perinteisesti globaaleja muuttujia järjestelmän moduulien välisen synkronointiin. Etenkin käytettäessä keskeytyksiä globaalien muuttujien käyttö saattaa aiheuttaa virhe- ja vaaratilanteita. Käyttöjärjestelmää käytettäessä näistä ongelmista päästään eroon ja synkronointi on turvallista.

Resurssien hallinta Reaaliaikaiset käyttöjärjestelmät tarjoavat usein palvelut resurssien hallintaan ja tällöin resurssien hallinta on lähes täysin käyttöjärjestelmän vastuulla. Käyttöjärjestelmä sisältää usein palvelut muistin, tehtävien, keskeytysten ja ajoitusten hallintaan, kommunikointiin sekä synkronointiin.

Parempi ajoitusten hallinta Käyttöjärjestelmä tarjoaa ajoitusten hallintaan palvelut, jotka mahdollistavat viiveiden, ajoitettujen tehtävien ja ajastimien käytön ilman laitteiston yksityiskohtien ymmärtämistä. Ilman käyttöjärjestelmää tarkkojen ajoitusten hallinta voi olla vaikeaa, sillä ohjelmoijan tulisi laitteiston yksityiskohtien ymmärtämisen lisäksi tietää miten esimerkiksi matalan tason ajastimia voi käyttää korkeammalla koodiin tasolla.

Silmukkarakennetta käytetään usein, mikäli sen käyttö on mahdollista ja järjestelmä on riittävän yksinkertainen[8]. Suurella taajuudella keskeytysohjattuja toimenpiteitä suorittavassa järjestelmässä silmukkarakenne voi myös olla paras vaihtoehto [7]. Reaaliaikaisen käyttöjärjestelmän käytön tarve riippuukin järjestelmästä, eikä ole olemassa tarkkoja kriteereitä siitä, milloin sitä tulisi käyttää [8]. Reaaliaikainen käyttöjärjestelmä saattaa tarpeettomasti monimutkaistaa järjestelmää.

2.3 Langattomien anturiverkkojen käyttöjärjestelmien piirteitä

Ensimmäiset langattomat anturiverkot on toteutettu ilman käyttöjärjestelmää [30]. Anturiverkkojen ominaisuuksien laitteistoresursseja nopeampi kasvu on luonut tarvetta käyttöjärjestelmän käytölle. Puhuttaessa anturiverkkojen käyttöjär-

jestelmistä tarkoitetaan useimmiten käyttöjärjestelmiä, jotka on suunniteltu erityisesti anturiverkkoja varten. Täydellinen anturiverkkojen ohjelmistoalusta koostuu reaaliaikaisen käyttöjärjestelmäytimen lisäksi protokollapinosta ja mahdollisesti esimerkiksi tiedostojärjestelmästä tai väliohjelmistokerroksesta [26].

Langattomien anturiverkkojen käyttöjärjestelmille asettuu seuraavia vaatimuksia palvelujen ja toiminnallisuuden suhteen [30].

Virhesietoisuus Anturiverkkoja käytetään monipuolisissa ympäristöissä ja hyvinkin pitkiä aikoja ilman toimintakatkoksia. Anturiverkko voi olla käytössä sellaisessakin ympäristössä, kuten kaivossa, maan sisällä tai kaivoksessa, jossa noden alkutilaan ulkoisesti asettaminen on hyvin vaikeaa. Katkeamaton käyttöikä voi olla ilman huoltotoimenpiteitä jopa useita vuosia. Käyttöjärjestelmän tulisi olla siten erittäin virhesietoinen. Sisäisen virheen sattuessa tulisi toiminta jatkua normaalisti ja toisaalta ulkoiset muutokset eivät saa aiheuttaa toiminnan katkeamista, mikäli se suinkin on mahdollista.

Muistinhallinta ja pieni muistitarve Anturiverkot ovat erittäin rajoittuneita etenkin datamuistiresurssien suhteen. Tehokkaalla dynaamisella muistinhallinnalla pystytään käyttämään rajoittuneet resurssit tehokkaasti hyödyksi. Toisaalta käyttöjärjestelmän tulisi tarvita mahdollisimman vähän muistia, jotta käyttöjärjestelmän tuoma hyöty ei täysin hukkuisi ja jotta käyttöjärjestelmää pystyisi yleensä käyttämään resurssirajoitteisella laitteistoalustalla.

Energiankulutuksen hallinta Paristokäyttöisyys ja pitkä elinikä asettavat tiukkoja vaatimuksia energiankulutuksen suhteen. Käyttöjärjestelmän tulisi osata hallita oheislaitteita energiatehokkaasti sulkemalla tarpeettomat pois käytöstä ja skaalaamalla dynaamisesti käyttöjännitettä tarpeen mukaan.

Reaaliaikaisuus Anturilaitteen protokollapino on hyvin aikakriittinen. Vaikka pinon suoritus toimisikin heikommilla aikakriteereillä saattaa tämä aiheuttaa esimerkiksi energiankulutuksen kasvua. Lisäksi anturiverkot ovat yhteydessä tiukasti reaalimaailmaan ja tämä aiheuttaa usein vaatimuksia reaaliaikaisuuden suhteen. Esimerkiksi anturiverkkoa voidaan käyttää datalähteenä jonkin laitteen ohjaamiseen ja silloin laitteen tarkka ohjaaminen voi edellyttää relevanttia ohjausdataa, joka edellyttää tiettyä reaaliaikaista toimintaa.

Oheislaitteiden hallinta Anturinode voi sisältää useita eri antureita ja muita oheislaitteita. Näitä voi käyttää useakin tehtävä samanaikaisesti, joten käyttöjärjestelmää tarvitaan laitteistoresurssien jakamiseen.

Tehtävien rinnakkaisuus Anturiverkot ovat rinnakkaisia perusluonteeltaan. Anturilaitteiden resurssien ja datavuon hallinta edellyttää rinnakkaista toimintaa. Käyttöjärjestelmän tulee sisältää toiminnot rinnakkaisuuden mahdollistamiseen.

Laitteiston abstrahointi Käyttöjärjestelmän tulee abstrahoida alla oleva laitteisto, sillä anturilaitteet ovat jopa saman verkon sisällä heterogeenisiä. Abstrahoinnilla mahdollistetaan saman ohjelmiston käyttö erilaisilla laitteistoalustoilla. Näin sovelluskehitys on helpompaa ja koodi monikäyttöisempää.

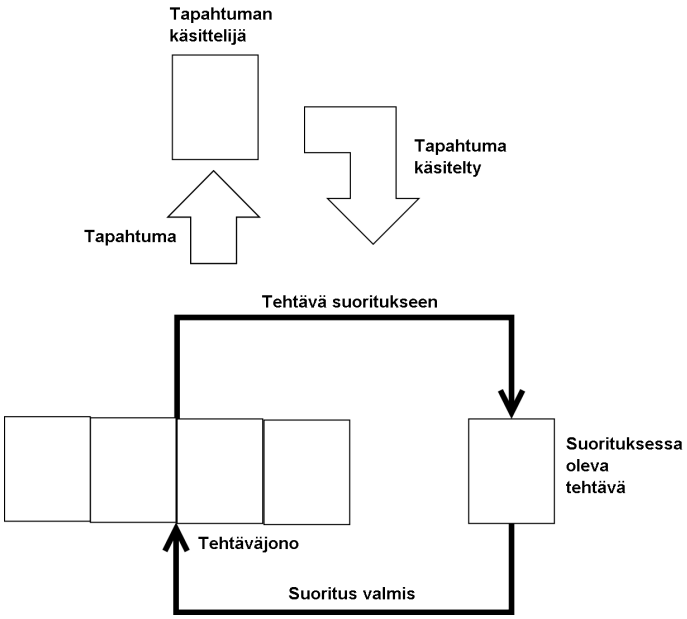
Modulaarisuus Anturiverkkojen sovellusten monimuotoisuuden takia käyttöjärjestelmältä vaaditut ajurit ja palvelut vaihtelevat. Tarpeettomien komponenttien poisto tulee tämän takia olla mahdollista.

Vaatimukset rajaavat anturiverkoille sopivien käyttöjärjestelmien määrää. Eritoten muisti- ja energiaresurssien rajallisuus asettavat tiukkoja vaatimuksia. Tavalliset sulautettujen järjestelmien käyttöjärjestelmät eivät kuitenkaan sovellu sensoriverkoille reaaliaikaisuusvaatimuksen takia. Yleiskäyttöisten reaaliaikaisten käyttöjärjestelmien käyttöön liittyy myös ongelmia. Joidenkin tavallisten reaaliaikaisten käyttöjärjestelmien käyttö ei ole mahdollista erittäin rajallisten muistiresurssien takia [30] [10]. Kevyimpiä reaaliaikaisia käyttöjärjestelmiä olisi mahdollista käyttää sensoriverkkojen ytimenä. Ne eivät kuitenkaan välttämättä tarjoa työkaluja energiankulutuksen hallintaan tai ole toiminnaltaan tarpeeksi energiatehokkaita [10]. Langattoman anturiverkon käyttöjärjestelmän tulee lisäksi olla sovellusta varten mukautettu, mikä estää yleiskäyttöisten käyttöjärjestelmien käytön [30].

2.3.1 Anturiverkkojen käyttöjärjestelmien toteutustavat

Rinnakkaisen prosessoinnin toteuttamiseen on vallalla kaksi eri toteutustapaa: vuoropohjainen tapahtumaohjattu ja irrottava monisäikeinen [11] [12]. Lisäksi on olemassa niin sanottuja hybridiytimiä, jotka ovat edellisten yhdistelmiä [25]. Toteutustapa vaikuttaa edellisessä kappaleessa esitettyjen vaatimusten toteutumiseen.

Kuvassa 2.5 esitetty tapahtumapohjainen ydin koostuu vuoropohjaisesta vuorontajasta, joka toteuttaa tapahtumapohjaista suunnittelufilosofiaa. Tehtävät ovat käytännössä tapahtumien käsittelijöitä, joiden suoritus ei ole keskeytyvää. Suoritus on siten enemmän sekventiaalista kuin rinnakkaista. Lisäksi prosessorin tilaa ei tarvitse suorituksen vaihdon yhteydessä tallettaa, mikäli tehtävät suoritetaan aina loppuun. Tällöin tehtävien tarvitsema datamuistin määrä on erittäin pieni. Haittana toteutustavassa on reaaliaikaisuuden puute, sillä suoritettavaa tehtävää ei voi keskeyttää [12]. Ohjelmointia pidetäänkin tavallisesti vaikeampana käytettäessä tapahtumapohjaista ydintä [26]. Tällöin sovellusohjelmoijan täytyy huolehtia paljon



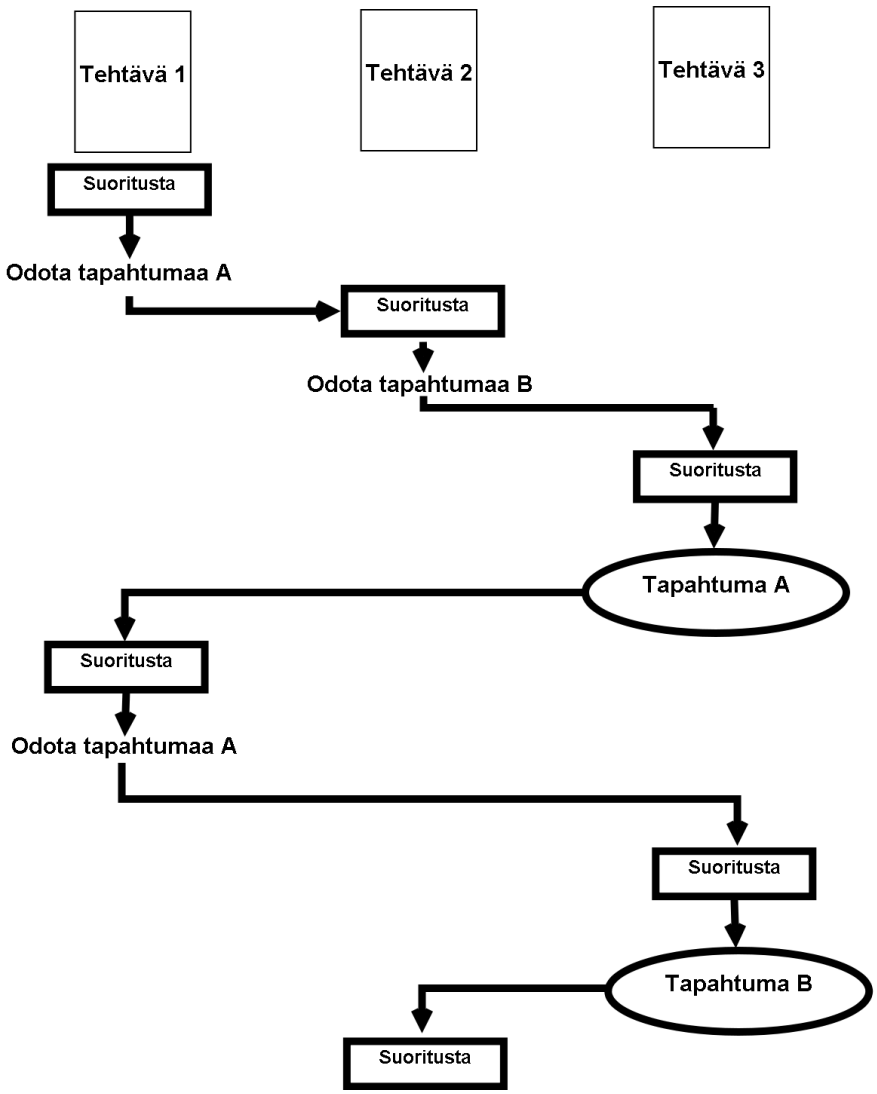
Kuva 2.5: Esimerkki tapahtumapohjaisesta ytimestä. Jonon tehtäviä suoritetaan prioriteettien mukaisessa järjestyksessä tai mikäli jokin tapahtuma suoritusta edellyttää.

aikaa vievien tehtävien suorituksen pilkkomisesta, jotta reaaliaikaisuusvaatimukset toteutuisivat. Lisäksi järjestelmän tulee toteuttaa tapahtumapohjaista suunnittelufilosofiaa.

Kuvassa 2.6 esitettyä irrottavaa monisäikeistä ydintä käytettäessä voidaan käyttää perinteisempää lähestymistapaa ohjelmointiin, jossa ohjelma koostuu useista säikeistä. Tällöin ohjelmoija voi vapaasti luoda säikeitä muusta ohjelmasta sen suuremmin välittämättä [26], sillä irrottavan vuorontamisen takia suoritus keskeytyy tarvittaessa. Irrottavalla monisäikeisellä ytimellä saavutetaan tiukat reaaliaikaisuusvaatimukset, mutta toisaalta suorituksen keskeytyksen yhteydessä tapahtuva kontekstin vaihto lisää datamuistin kulutusta [35].

Tapahtumapohjaista ydintä pidetään yleisesti energiatehokkaampana vähäisten resurssien käytön takia [20]. Irrottavan monisäikeisen ytimen energiatehokkuus lisääntyy voimakkaammin kuorman kasvaessa tapahtumapohjaiseen nähden [12]. Käyttöjärjestelmän energiatehokkuus riippuu sen tyhjäkäyntiajasta ja irrottavan monisäikeisen ytimen suoritus vaatii enemmän prosessoriaikaa.

Anturiverkkojen käyttöjärjestelmien ytimen arkkitehtuurina käytetään sekä monoliittistä, että mikroydintä. Valtaosa anturiverkkojen käyttöjärjestelmistä käyttää mikroydintä, sillä se tarjoaa paremman muokattavuuden [30]. Lisäksi mikroydin on monoliittistä ydintä robustimpi ytimen sisäisille virheille. Kuitenkin esimerkiksi TinyOS, joka on eräs suosituimmista anturiverkkojen käyttöjärjestelmistä, sisältää monoliittisen ytimen [20]. Monoliittinen ydin mahdollistaa järjestelmän komponenttien tiukemman integraation ja samalla pienemmän muistijalanjäljen [30].



Kuva 2.6: Esimerkki irrottavasta monisäikeisestä ytimestä. Tehtäviä ei suoriteta loppuun, vaan niiden suoritus keskeytyy joko suuremman prioriteetin tehtävän tarvitessa suoritusta tai niiden itse niin halutessa. Kuvassa matalimman numeron omaavalla tehtävällä on korkein prioriteetti. Tehtävää 3 suoritetaan vain, kuin korkeamman prioriteetin tehtävät antavat sille suoritusvuoron.

Taulukossa 2.1 on esitetty anturiverkkojen käyttöjärjestelmien vaatimukset ja ne parhaiten täyttävät toteutustavat. Taulukosta nähdään, että jokaisella tekniikalla on omat hyvät puolensa.

2.4 Käyttöjärjestelmiä langattomille anturiverkoille

Seuraavassa on esitelty työn kannalta keskeisimmät käyttöjärjestelmät langattomille anturiverkoille. Työssä käytetty SensorOS on esitelty erikseen kappaleessa 2.5.

TinyOS Berkeleyn yliopistossa kehitetty TinyOS on eräs tavallisimmista ja vanhimmista anturiverkkojen käyttöjärjestelmistä. Se sisältää staattisen tapahtu-

Taulukko 2.1: Anturiverkkojen käyttöjärjestelmän vaatimukset [30] parhaiten toteuttavat tekniikat. Vaihtoehdot ovat mikroydin/monoliittinen ydin ja tapahtumapohjainen ydin/irrottava monisäikeinen ydin).

Vaatimus	**Parhaiten vaatimuksen täyttävä tekniikka**
Robustisuus	Mikroydin
Pieni muistijalanjälki	Tapahtumapohjainen monoliittinen ydin
Pieni energiankulutus	Tapahtumapohjainen ydin
Reaaliaikaisuus	Irrottava monisäikeinen ydin
Tehtävien rinnakkaisuus	Irrottava monisäikeinen ydin
Modulaarisuus	Mikroydin

maohjatun ytimen, mutta itse käyttöjärjestelmä on laajennettavissa useilla eri laajennoksilla. Esimerkiksi keskeyttävä moniajo voidaan TinyOS:ssä toteuttaa laajennoksen avulla. Helppo laajennettavuus onkin yksi TinyOS:n suurimmista eduista. TinyOS:n erikoisuutena on sen ohjelmointi kieli:NesC, eräs C:n murteista [20].

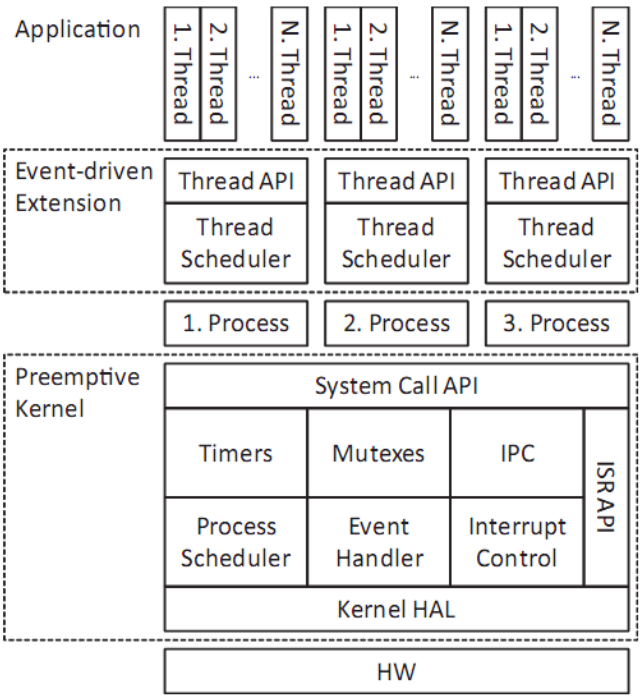
Contiki Contiki sisältää hybridiytimen, jossa irrottava moniajo on toteutettu erillisenä kirjastona tapahtumapohjaisen ytimen päälle. Käyttöjärjestelmän erikoisuus on dynaaminen lataus: ohjelmia ja palveluita voidaan ladata "lennosta"suorituksen aikana [1].

Mantis Mantis sisältää irrottavan monisäikeisen ytimen, jossa skedulointi on hoidettu aikaperusteisesti. Lisäksi käyttöjärjestelmään on integroitu verkkopino, mikä tekee sovelluskehityksestä helppoa. Toisaalta se estää protokollapinon kehityksen. Mantisin heikkous on kontekstin vaihdon aiheuttama kustannus [10].

2.5 SensorOS

SensorOS on Tampereen teknillisen yliopiston tietokonetekniikan laitoksella kehitetty käyttöjärjestelmä, joka on suunniteltu käytettäväksi eritoten TUTWSN-anturiverkon kanssa [26]. SensorOS:stä on olemassa useita eri versioita. Tässä työssä puhuttaessa SensorOS:stä tarkoitetaan lähteessä [26] esiteltyä Hybrid Kernelin sisältävää versiota. SensorOS ei itsessään ei sisällä tällä hetkellä oheislaiteajureita eikä se tue virallisesti muita mikrokontrollereita kuin TUTWSN-alustoissa käytettyjä Microchipin PIC18LF8722- ja PIC18F4560-kontrollereita.

Kuvassa 2.7 on kuvattu SensorOS:n rakenne [26]. Se sisältää niin sanotun hybridiytimen, jossa tapahtumapohjaisuus on integroitu irrottavan monisäikeisen ytimen yhteyteen. Tämä eroaa esimerkiksi ConTikin toteutuksesta, jossa irrottava moniajo



Kuva 2.7: SensorOS:n rakenne. Käyttöjärjestelmän ydin on irrottava monisäikeinen, mutta prosessien sisällä on toteutettu tapahtumapohjainen lisäys. Ytimen toimintoihin pääsee käsiksi järjestelmäkutsujen välityksellä. Kuvan lähde [26]

on toteutettu tapahtumapohjaisen ytimen päälle. Suoritus koostuu säikeistä, joiden suoritusta ohjaa prioriteettien perusteella toimiva round-robin-tyyppinen vuorontaja. Yksinkertaistettuna kyseisessä vuorontajassa suurimman prioriteetin prosessi saa aina suoritusvuoron ja tämä suoritusvuoro pysyy prosessilla niin kauan, kunnes se siitä itse luopuu. SensorOS ei tue dynaamista säikeiden hallintaa, vaan säikeet luodaan käynnistämisen yhteydessä ja ne ovat olemassa koko järjestelmän suorituksen ajan. Säikeiden suorituspinon koko on määriteltävä säiettä alustettaessa.

Säikeiden sisälle on toteutettu protosäikeitä, joihin ei sisälly kontekstin vaihdosta aiheutuvia kustannuksia [26]. Tulee kuitenkin muistaa, että toisin kuin normaaleilla säikeillä, paikallisten muuttujien arvo ei säily suorituksen vaihdon yhteydessä. Protosäikeiden hallinta on mahdollista toteuttaa dynaamisesti. Protosäikeillä on lisäksi vuorontaja, joka toimii samaan malliin kuin säikeiden vastaava. Säikeiden tulee huolehtia tämän vuorontajan kutsumisesta, mikäli se on tarpeen. Jatkossa SensorOS:n yhteydessä puhuttaessa kutsutaan järjestelmän säikeitä prosesseiksi ja protosäikeitä säikeiksi. Tämä johtuu SensorOS:n nimeämiskäytännöstä, jolla korostetaan säikeiden ja protosäikeiden aiheuttaman muistin käytön eroa. Nimeämisellä estetään myös sekaannukset protosäikeiden ja säikeiden välillä.

SensorOS voi sisältää maksimissaan 8 prosessia ja jokaisella prosessilla voi olla maksimissaan 32 säiettä. Tosin korkeimman prioriteetin prosessi eli niin sanottu

aikakriittinen prosessi voi sisältää vain yhden säikeen.

Järjestelmän suoritusta kontrolloidaan SensorOS:ssä pääasiassa tapahtumien avulla [26]. Säie tai prosessi asetetaan odottamaan tiettyä tapahtumaa, jonka tapahtuessa suoritus jatkuu odottamiskohdasta. Tapahtuman voi aiheuttaa käytännöllisesti katsoen mikä tahansa järjestelmän toiminto, sillä tapahtumia voi luoda myös itse. Valmiiksi on olemassa tapahtumat ajastimen laukeamiseen, mutexin vapauttamiseen ja keskeytyksiin. Säikeillä ja prosesseilla on omat tapahtumanjonotusfunktiot. Prosessien funktio on parametriton, mutta säikeille tarkoitettu funktio tarvitsee parametrikseen säiettä luotaessa tehdyn jatkopaikasta kertovan muuttujan.

SensorOS sisältää seuraavat palvelut:

IPC Jokaisella prosessilla ja säikeellä on postilaatikko IPC-viestien vastaanottamiseen. Viestejä lähetettäessä on kohdesäikeen/-prosessin ID tiedettävä. Käyttöjärjestelmä hoitaa viestin välityksen tämän ID:n perusteella. Viestin lähettäminen laukaisee tapahtuman, joka myös keskeyttää lähettävän prosessin suorituksen, mikäli viesti on lähetetty korkeamman prioriteetin prosessille.

Muistinhallinta Dynaamiseen muistinhallintaan on kaksi toteutusta: kiinteän kokoisella muistinvarausyksiköllä tai hakutaulukolla toteutettu. Molemmissa toteutuksissa muistinvarausyksikön koko säädetään käännösaikana. Ensin mainitussa yksiköillä on vain yksi koko, kun hakutaulukkototeutuksessa näitä on kolme. Hakutaulukkototeutuksessa muisti jaetaan osiin ja hakutaulukkoon tallennetaan tieto siitä, onko osa varattu. Hakutaulukolla toteutettu muistinhallinta on käytännöllisempi, mikäli on tarvetta varata muistia erikokoisissa lohkoissa.

Synkronointi Poissulkeminen on toteutettu mutexien avulla. Perinteisten mutexien lisäksi käyttöjärjestelmä sisältää käänteisprioriteettivapaat mutexit, joiden ansiosta korkeimman prioriteetin varaaja saa aina mutexin niin halutessaan.

Ajastimet SensorOS sisältää kaksi ajastintyyppiä: mikrosekunnin tarkkuuteen pystyvää ajastinta voi käyttää käytännössä vain yksi prosessi kun taas millisekunnin tarkkuuteen pystyvää ajastinta voivat käyttää myös muut prosessit ja säikeet. Ensimmäisestä käytetään jatkossa nimeä tarkkuusajastin ja jälkimmäisestä nimeä hälytysajastin. Molemmat ajastimet on toteutettu tapahtumien avulla, mutta tarkkuusajastimessa tämä tapahtumatoteutus on abstrahoitu täysin. Tämä tarkoittaa käytännössä sitä, että vain hälytysajastimen laukeamista voidaan odottaa kuten normaaleja tapahtumia. Tarkkuusajastimen odotus estää muiden alempiprioriteettisten prosessien suorituksen. Lisäksi hälytysajastimia voi asettaa vain yhden kutakin säiettä kohti. Käyttöjärjestelmä tarjoaa korkean tason ajastimien lisäksi rajapinnan mikrokontrollerin

Taulukko 2.2: Käyttöjärjestelmien ominaisuudet kootussa muodossa. Mukana on anturiverkkokäyttöjärjestelmien lisäksi sulautetuissa reaaliaikajärjestelmissä käytetty FreeRTOS [9].

Käyttöjärjestelmä	SensorOS	TinyOS	ConTiki	Mantis	FreeRTOS
Ydin	Mikro	Monoliittinen	Mikro	Mikro	Mikro
Vuorontaminen	Hybridi	Tapahtuma / Hybrdidi	Tapahtuma / Hybridi	Irrottava	Irrottava
Datamuistin kulutus (tavua	1053	1427	1003	2352	3308
Ohjelmamuistin kulutus (tavua)	5274	9000	4000	14000	5000

ajastinlohkojen ohjaamiseen. Tämä rajapinnan avulla on mahdollista käyttää ajastinlohkoja käyttämättä korkean tason ajastinrajapintoja.

Keskeytysrajapinta Keskeytykselle voidaan asettaa kutsufunktio ja lisäksi keskeytys voi laukaista tapahtuman. Viivästetty palveluohjelma voidaan toteuttaa esimerkiksi tämän tapahtuman avulla. DSR:n, joka voi olla tavallinen säie tai prosessi, suoritus asetetaan laukaistavaksi tapahtumasta, jonka keskeytys aiheuttaa.

SensorOS vaatii 5274 tavua ohjelmamuistia [26]. Datamuistin kulutus riippuu käytettyjen prosessien ja säikeiden määrästä. Ydin vie itsessään vain 89 tavua datamuistia. Jokainen prosessi vaatii lisäksi datamuistia käytetyn pinon koon verran ja lisäksi 28 tavua ja 31 tavua jokaista säiettä kohti. Jokaisella prosessilla tulee olla vähintään yksi säie ja pinon kokona käytetään tavallisesti 128 tavua tai 196 tavua riippuen prosessista. Yhden prosessin järjestelmän käyttöjärjestelmän aiheuttama datamuistin kulutus olisi siten 276 tavua ja esimerkiksi 3:n prosessin ja 5 säikeen järjestelmän 712 tavua.

2.6 Käyttöjärjestelmän valinnasta

Taulukossa 2.2 on esitetty kootussa muodossa käsitellyt käyttöjärjestelmät. Lisäksi mukana on vertailun vuoksi tavallinen reaaliaikainen käyttöjärjestelmä FreeRTOS [9]. TinyOS sekä Contiki sisältävät laskelmissa moniajolaajennoksen. Irrottaville monisäikeisille käyttöjärjestelmille käytettiin 16 säiettä, muille kolmea säiettä ja 16 protosäiettä. Tulokset eivät täysin ole vertailukelpoisia keskenään, sillä käytetty prosessori vaikuttaa muistin kulutukseen. Lisäksi etenkin ohjelmamuistin kulutus riippuu käyttöjärjestelmän konfiguraatiosta. Tulokset ovat kuitenkin suuruusluokaltaan oikeita.

Käyttöjärjestelmän valinta riippuu aina kokonaisjärjestelmästä, jossa käyttöjärjestelmä otetaan käyttöön. Kuten taulukosta 2.1 nähdään kullakin toteutustavalla on puolensa. Anturiverkoilla on eroja, joten yhtä sopivaa käyttöjärjestelmää ei ole olemassa. Toisten järjestelmien reaaliaikaisuusvaatimukset ovat tiukempia, toisissa taas esimerkiksi laitteisto voi olla resurssirajoitteisempi.

Käyttöjärjestelmän valintaprosessi lähtee liikkeelle käyttöjärjestelmän tarpeen määrittämisestä. Vaikka käyttöjärjestelmä sinänsä yksinkertaistaakin sovellusohjelmointia se myös monimutkaistaa aivan yksinkertaisempia järjestelmiä. Käyttöjärjestelmän tarve lisääntyy prosessien ja laitteiden määrän kasvaessa [14]. Käyttöjärjestelmä soveltuu pieniinkin järjestelmiin, mutta aivan pienimpiin vain satoja tavuja datamuistia sisältäviin järjestelmiin se on turhan raskas, sillä käyttöjärjestelmän tehokas käyttö kuluttaisi kaiken datamuistin [32]. Laitteisto vaikuttaa muutenkin käyttöjärjestelmän valintaan. Käyttöjärjestelmä ei välttämättä tue kaikkia prosessoreja [8] ja lisäksi osassa saattaa olla valmiina jo tuki joillekin oheislaitteille. Lisäksi esimerkiksi keskeyttävä moniajo kuluttaa datamuistia niin, että aivan pienimmillä prosessoreilla kyseisen tyypin käyttöjärjestelmän käyttö ei ole järkevää. Käyttöjärjestelmän tulisi lisäksi toimia käytetyillä ohjelmointityökaluilla [8]. Lisäksi muutettaessa jo olemassa olevaa järjestelmää käyttöjärjestelmäpohjaiseksi voi jo olemassa olevien palveluiden ja käyttöjärjestelmän tarjoamien palveluiden ja niiden rajapintojen samanlaisuus nopeuttaa ohjelmistokehitystä.

Anturiverkkojen käyttöjärjestelmän tulee olla muokattava [30]. Käyttöjärjestelmät sisältävät usein useita valinnaisia moduuleita, joista voi muodostaa sopivan kokoonpanon. Näin käyttöjärjestelmä sisältää vain ja ainoastaan ne toiminnot, joita tarvitaan. Lisäksi tiettyjä toimintoja voi muokata. Esimerkiksi vuorontamisalgoritmin voi joissain käyttöjärjestelmissä valita käännösaikana [29]. Muokattavuus säästää resursseja ja parantaa suorituskykyä. Käyttöjärjestelmän sisältämät moduulit ovat lisäksi usein tärkeä valintaperuste, sillä valmiit moduulit nopeuttavat ja helpottavat ohjelmistokehitystä [7]. Esimerkin vuoksi taulukossa 2.3 on esitetty ConTikin moduleita. ConTiki sisältää esimerkiksi TCP/IP-verkkopinon valmiina moduulina.

Käyttöjärjestelmän tekeminen on mahdollista myös itse. Etenkin kooltaan pienissä tai suurten yritysten ja tutkimuslaitosten projekteissa saatetaan käyttää itse kehitettyä reaaliaikaista käyttöjärjestelmää [21]. Valmiiseen testattuun käyttöjärjestelmään liittyy kuitenkin tiettyjä etuja etenkin ohjelmistokehityksen suhteen [21]. Käyttöjärjestelmän kehittäminen vie kehitysresursseja ja lisäksi varsinaisen ohjelmiston kehityksessä voidaan joutua korjaamaan käyttöjärjestelmän virheitä sovelluksen virheiden sijaan. Valmis käyttöjärjestelmä voi lisäksi tarjota esimerkiksi useita kehitystyökaluja ja lisäksi tuen useille prosessoreille ja oheislaitteille. Itse toteutetulla käyttöjärjestelmällä on etuna mahdollisuus optimoida suorituskyky ha-

Taulukko 2.3: ConTiki-käyttöjärjestelmän eri moduuleita (Taulukon lähde: [1]).

Moduulii	Selitys
CTK	Graafinen käyttöliittymäkirjasto
RMH	Monen hypyn reititys
rudolph2	Yhden hypyn datan kuljetusmekanismi
uIP	Datatyyppien määrittelymoduuli
uIP TCP/IP	TCP/IP verkkopino
Raven	Ohjain LCD-näytölle
Rime	Kommunikointipino

luttua sovellusta varten.

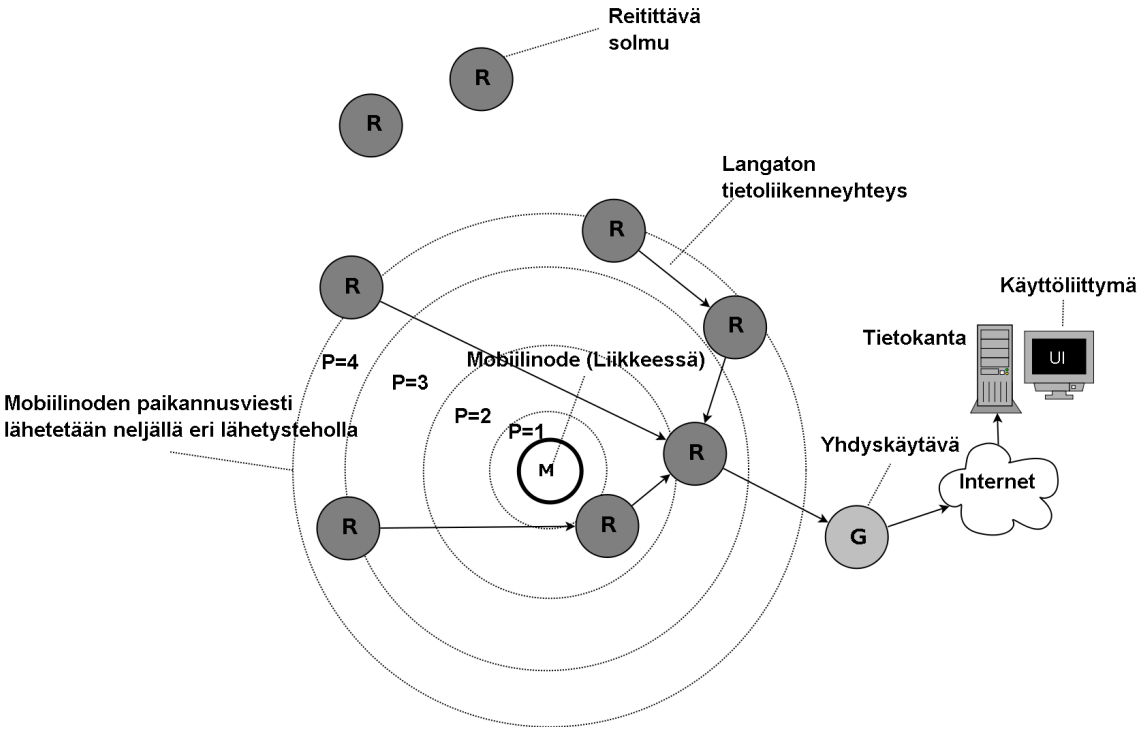
3. TUTWSN

Tässä luvussa kerrotaan Tampereen teknillisen yliopiston tietokonetekniikan laitoksella kehitetystä TUTWSN-anturiverkkoteknologiasta, jota on kehitetty vuodesta 2002 lähtien. TUTWSN on täyden toiminnallisuuden anturiverkko. Se sisältää varsinaisen anturiverkon lisäksi esimerkiksi palvelinohjelmiston ja käyttöliittymän, jolla voi ohjata anturiverkon toimintaa.

TUTWSN-anturiverkot voidaan jakaa kolmeen luokkaan: pitkän kantaman verkkoon (engl. Long Range TUTWSN), lyhyen viiveen verkkoon (LL-TUTWSN, engl Low latency TUTWSN) ja matalan energiankulutuksen verkkoon (engl. Low Energy TUTWSN). LL-TUTWSN on optimoitu lyhyttä vasteaikaa vaativiin sovelluksiin ja sisätilapaikannuskäyttöön. Sen viive on hyppyä kohden jopa alle 100 ms. Pitkän kantaman ja matalan energiankulutuksen TUTWSN-verkot ovat energiatehokkaita mittausverkkoja. Pitkän kantaman verkossa nodejen etäisyys toisistaan voi olla jopa 500 metriä, kun taas matalan energiakulutuksen verkossa tämä voi olla maksimissaan 30 metriä. Pitkän kantaman verkko eroaa muista taajuuden perusteella. Pitkän kantaman verkon radioyhteyden taajuus on 433 MHz kun se muissa verkoissa on 2,4 GHz. Tässä työssä keskitytään LL-TUTWSN-verkkoon. Jatkossa puhuttaessa TUTWSN:stä tarkoitetaan nimenomaan LL-TUTWSN-teknologiaa.

TUTWSN on autonominen anturiverkko. Tämä tarkoittaa sitä, että verkkoon liittyminen ja verkon muodostuminen tapahtuu automaattisesti ja toiminta on mahdollista ilman ulkoisia runkoverkkoja. TUTWSN käyttää monen hypyn *ad-hoc mesh*-topologiaa, mikä tarkoittaa sitä, että reitti voidaan muodostaa automaattisesti usean hypyn ylitse ilman ennalta määrättyä konfiguraatiota [22].

Kuvassa 3.1 on esitetty tyypillinen TUTWSN-anturiverkko. Se koostuu kolmen tyyppisistä anturilaitteista: mobiili-nodeista, reititin-nodeista ja yhdyskäytävä-nodeista eli nieluista. Yhdyskäytävä-nodet toimivat verkon linkkinä palvelimelle. Ne lähettävät vastaanottamansa datan TCP/IP-yhdyskäytävää pitkin palvelimen tietokantaan, josta käyttäjä voi käyttöliittymän välityksellä tarkastella verkon toimintaa. Reititin-nodet välittävät vastaanottamansa ja tuottamansa datan jopa usean hypyn välityksellä yhdelle tai useammalle yhdyskäytävä-nodelle. Reitin yhdyskäytävälle reititin-nodet muodostavat täysin automaattisesti. Mobiili-nodet lähettävät säännöllisesti paketteja reititin-nodeille. Lähetysintervalli on säädettävissä ja riippuu sovelluksesta ja verkon käyttötarkoituksesta. Reititin- ja yhdyskäytävä-nodejen

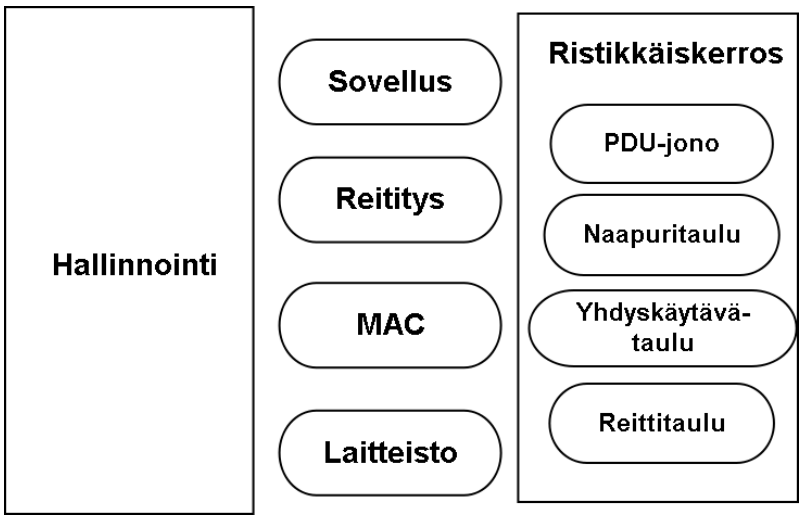


Kuva 3.1: TUTWSN:n periaate paikannuksessa. Mobiili-node lähettää paikannusviestin neljällä eri lähetysteholla ilman kohdeosoitetta. Reitittävien nodejen sijainti on tiedossa, joten paikannusviestin vastaanottajien perusteella pystytään määrittämään myös mobiilinoden sijainti.

virrankulutus on suuri, mutta taasen mobiili-nodeilla päästään jopa vuosien käyttöikään tavallisilla AA-paristoilla.

TUTWSN-verkkoa voi käyttää myös paikannuskäyttöön [22]. Kuvassa 3.1 on esitetty paikannuksen periaate. Onnistuneesti lähetettyjen paikannusviestien perusteella pystytään määrittämään mobiili-noden sijainti. Paikannus perustuu signaalin voimakkuuden arviointiin ja reitittävien nodejen tunnettuun sijaintiin. Jokaiselle lähetysteholle pystytään arvioimaan signaalin kantama, jonka perusteella pystytään arvioimaan mobiili-noden paikka suhteessa reitittäviin nodeihin. Mobiili-nodet eivät ole tietoisia ympäröivistä nodeista.

Kuvassa 3.2 on esitetty TUTWSN-anturilaitteen sulautetun ohjelmiston rakenne hyvin karkealla tasolla kuvattuna. Protokollapino koostuu neljästä varsinaisesta kerroksesta: laitteisto, MAC, reititys ja sovellus. Lisäksi ohjelmistossa on ristikkäis- (engl. cross-layer) ja hallinnointikerrokset, jotka toimivat apukerroksina protokollapinon kerroksille. Eri kerrokset on toteutettu ohjelmallisesti. Laitteistokerros voidaan jakaa kahteen osaan: HPL-kerrokseen (engl. Hardware peripheral layer) ja ajureihin. HPL on täysin alustariippuvainen. Se abstrahoi varsinaisen raudan pinnimäärittelyt ja mahdollistaa esimerkiksi saman ajurin käyttämisen eri raudalla. Ajurikerros sisältää laiteajurit raudan ohjaamiseen. Ajurikerros on lähes alustariippumaton. MAC-kerros on vastuussa datan lähettämisestä. Se päättää lähetysajat ja



Kuva 3.2: TUTWSN:n anturilaitteen sulautetun ohjelmiston rakenne.

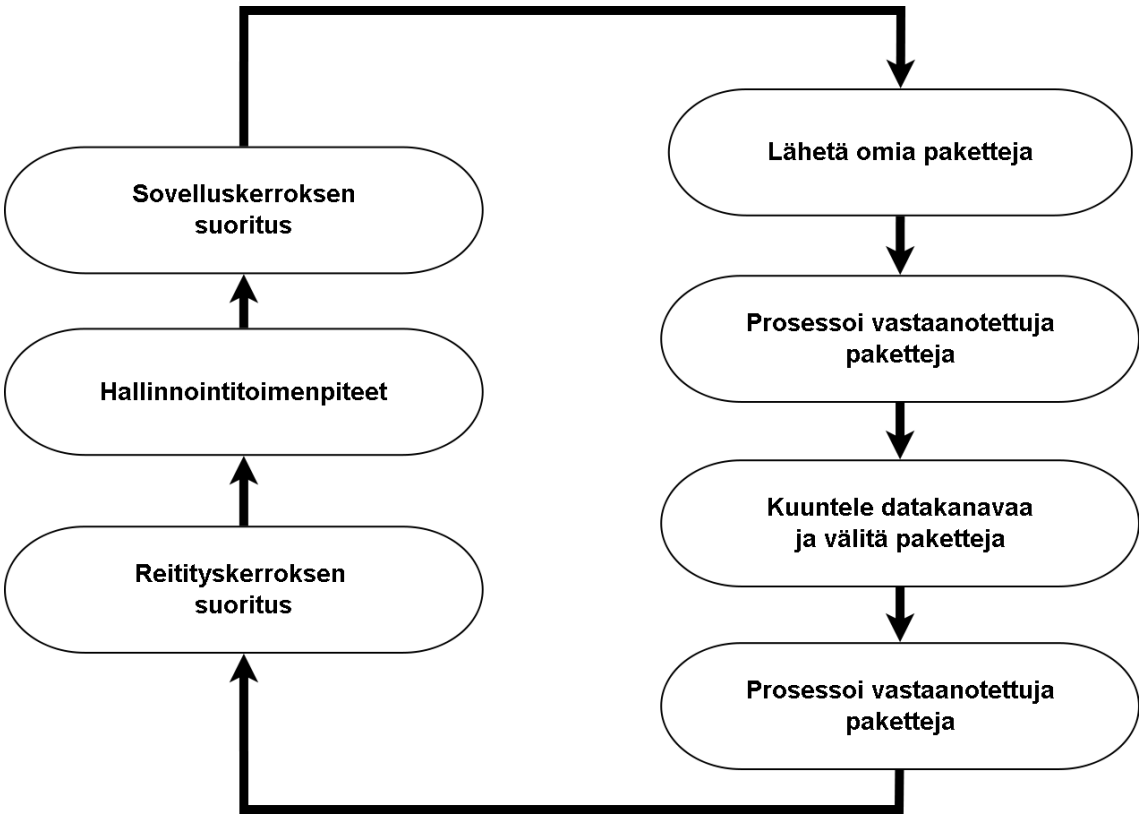
valitsee lähetettävät paketit. Reitityskerros muodostaa ja pitää yllä reittejä eri liikenneluokille. Sovelluskerros koostuu yksinkertaistettuna sovellusten suorittamiseen liittyvistä moduuleista, kuten sovellusvuorontajasta ja sovellusten ohjelmointirajapinnasta. Kaikki TUTWSN-sovellukset käyttävät samaa rajapintaa.

Hallinnointikerros suorittaa hallinnollisia toimenpiteitä. Se esimerkiksi päättää milloin tulee suorittaa verkkoskannaus tai milloin reitit tulee muodostaa uudestaan. Ristikkäiskerros sisältää useita muitten ohjelmistokerrosten yhdessä käyttämiä tietorakenteita. Se säilöö tietoa esimerkiksi anturilaitteen naapureista ja toimii samalla tiedonvälittäjänä eri kerrosten välillä PDU-jonon (engl. Protocol data unit) avulla.

TUTWSN:n ohjelmiston suoritusvuo on esitetty kuvassa 3.3. Suoritus koostuu käytännössä pakettien vastaanotosta ja lähetyksestä, hallinnollisista toimenpiteistä ja sovelluskerroksen suorituksesta. Paketteja vastaanotetaan sekunnin ajan aktiivisesti odottamalla.

3.1 MAC

MAC-kerros on vastuussa datan lähettämisestä [34]. Mobiili-nodet eroavat reitittävistä nodeista MAC-kerroksen osalta ratkaisevasti. Mobiili-nodet eivät välitä toisten nodejen paketteja vaan toimivat alkulähteinä. Yhdyskäytävät ovat taasen pakettien päätepisteitä anturiverkkotasolla. Mobiili-nodet toimivat vain yhdellä radiokanavalla, kun taas reitittävät käyttävät kahta radiokanavaa. Mobiili-noden radiokanava ja yksi reitittävän noden radiokanavista on varattu datan lähettämiseen ja sitä kutsutaan nimellä datakanava. Reitittävän noden toista radiokanavaa käytetään verkon muodostamiseen ja sitä kutsutaan nimellä verkkokanava. Mobiili-nodet voivat lähettää paketteja täysin satunnaisesti, mikä mahdollistaa niille pitkän elin-



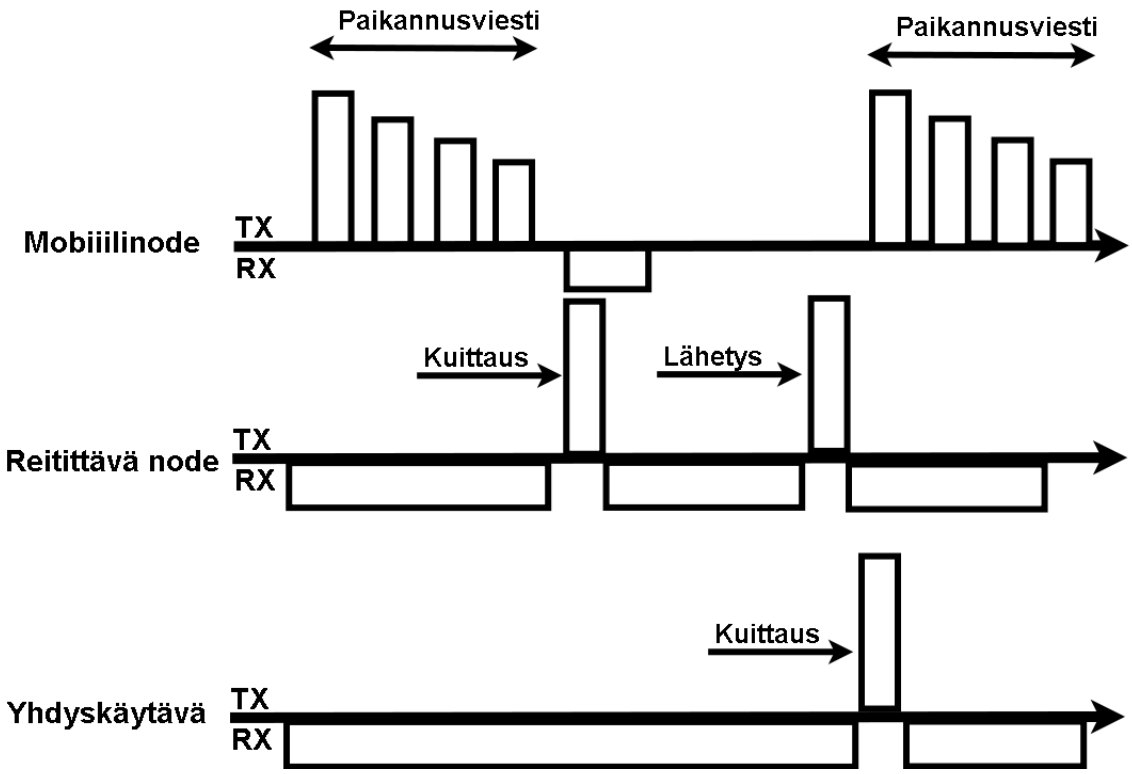
Kuva 3.3: Anturilaitteen ohjelmiston rakenne.

iän.

Esimerkki tiedon liikkumisesta TUTWSN-verkossa on esitetty kuvassa 3.4. Reititin- ja yhdyskäytävä-nodet ovat normaalitilassaan lähes jatkuvasti vastaanottotilassa mobiili-nodejen lähettäessä paikannusviestejä tietyllä aikavälillä. Lähetysintervalli voi muuttua tilanteen mukaan. Paikannusviestejä lähetetään neljä kappaletta neljällä eri lähetysteholla, jonka jälkeen siirrytään kuuntelemaan kuittausta. Vastaanotettuaan paikannusviestin reitittävä node lähettää tästä mobiili-nodelle kuittauksen. Vastaanotettu paketti lähetetään eteenpäin yhdyskäytävää kohti. Reititinnodejen ja yhdyskäytävien väliset lähetykset sisältävät myös kuittauksen vastaanotetusta paketista.

MAC-kerros kontrolloi verkkoviestien lähetystä verkkokanavalla. Verkkoviestien perusteella reitittävät nodet muodostavat reitin yhdyskäytävälle. Verkkoviesti lähetetään satunnaisesti noin kahden sekunnin välein. Lähetysaika on satunnainen, jotta törmäyksiltä vältyttäisiin.

MAC-kerros ei ole kovin aikakriittinen muuten kuin kuittausten osalta. Reititin- ja yhdyskäytävä-nodeilla on paketin vastaanotosta 5 ms aikaa lähettää kuittaus. Myöhästynyt kuittaus voi aiheuttaa uuden, tarpeettoman lähetyksen. Uudelleenlähetysten määrä riippuu lähetettävän paketin liikennöintiluokasta. Kaikki tieto



Kuva 3.4: Esimerkki datan liikkumisesta TUTWSN-verkossa. Mobiili-node lähettää paikannusviestin neljällä eri lähetysteholla sitä ympäröiville reitittäville nodeille. Reitittävä node kuittaa vastaanotetun paikannusviestin ja lähettää tarvittaessa tästä paketin eteenpäin yhdyskäytävä-nodelle muita reitittäviä nodeja hyväksi käyttäen. Reitittävien nodejen väliseen kommunikaatioon kuuluu myös kuittausten lähetys vastaanotetuista paketeista.

mobiili-nodeille kulkee kuittauspaketeissa, joten tiedonkulku viivästyy, mikäli kuittaus on myöhässä.

3.2 Reitityskerros

Reititysprotokolla muodostaa useita kustannusperusteisia reittejä yhdyskäytävälle [34]. Reitit muodostetaan automaattisesti valmiiksi ja niiden muodostaminen tapahtuu tasaisin väliajoin tai tilanteen niin vaatiessa. Node mainostaa verkkoviesteillä reittitietojaan ja näiden tietojen perusteella pystyvät muut nodet muodostamaan reittinsä. Verkkoviestien lähetys ja vastaanotto tapahtuu datakanavan sijaan verkkotason signalointikanavalla. Reitittävä node suorittaa skannauksen tällä kanavalla käynnistyksen yhteydessä ja muulloin tarvittaessa. Verkkoviestien lisäksi reitin muodostamisessa ovat oleellisia reitinmainostuspaketit, joita lähetetään datakanavalla reittitietojen muuttuessa. Tämän paketin perusteella node tietää muiden nodejen reittien muuttuneen ja tämän perusteella se pystyy suorittamaan verkkotason signalointikanavalla skannauksen, jossa se hakee ympäristössä olevat naapuri-nodet.

3.3 Järjestelmän muut komponentit

Lisäksi järjestelmässä on hallinnointikerros ja ristikkäiskerros [34]. Hallinnointikerros suorittaa hallinnollisia toimenpiteitä ja toimii eräänlaisena viestinvälitysmoduulina eri kerroksien välillä. Se esimerkiksi päättää milloin reitti tulee muodostaa uudelleen tai milloin tulee suorittaa verkkoskannaus. Muut kerrokset kommunikoivat hallinnointikerroksen kanssa yksinkertaista signalointijärjestelmää käyttäen. Lisäksi hallinnointikerros pitää huolta järjestelmän toiminnan kannalta tärkeistä ajastetuista toiminnoista, kuten naapuritaulun vanhojen jäsenten poistosta.

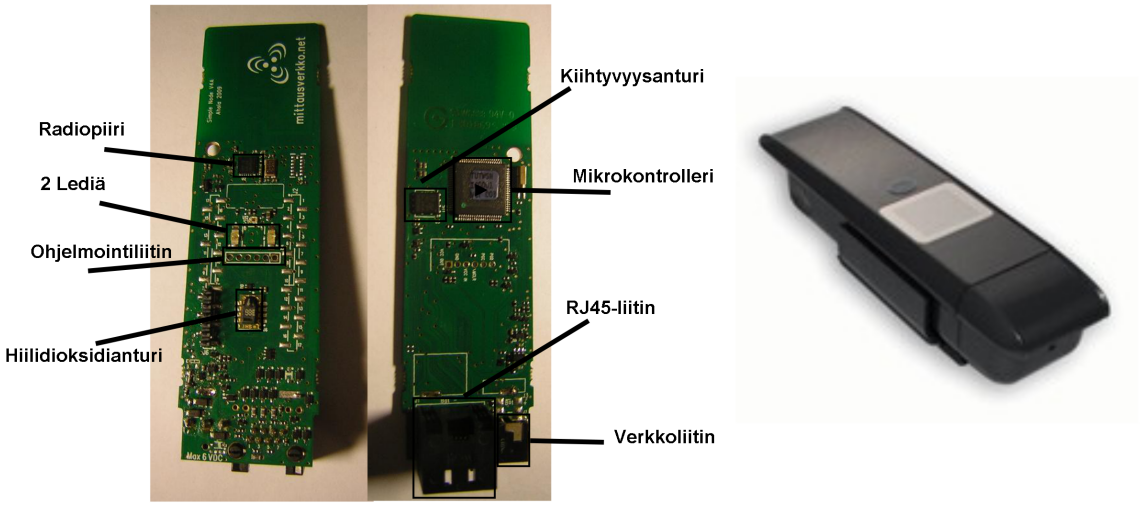
Ristikkäiskerros sisältää muun muassa tietorakenteita, joita käytetään useammalla kerroksella. Järjestelmän tiedon välitys hoidetaan pääasiallisesti staattisen prioriteettijonon ja hallinnointikerroksen signaalien kautta.

Järjestelmä sisältää lisäksi tällä hetkellä useita perinteisesti käyttöjärjestelmän vastuulle kuuluvia palveluita. Näitä ovat dynaaminen muistinhallinta ja viestin välitys. Molemmat toiminnot ovat identtisiä SensorOS:n vastaavien toimintojen kanssa. Muistinhallinta käyttää yhtä kiinteää kokoa muistinvarausyksikölle.

3.4 Laitteistoalusta

TUTWSN-anturilaitteille käytetään omia, tarkoitusta varten suunniteltuja laitteistoalustoja, joista tämän työn kannalta oleellisin, simple-malli, on esitelty kuvassa 3.5. Lisäksi on olemassa esimerkiksi pocket-alusta ja ethgw-alusta. Pocket- ja simple-alustat eroavat toisistaan pääasiassa antennin, koon, liitäntöjen ja joidenkin komponenttien osalta. Perusrakenne on molemmissa alustoissa sama. Pocket on optimoitu käytettäväksi mobiili-nodena. Se on kooltaan simple-alustaa pienempi ja sen pääasiallisena energialähteenä on AAA-paristot simplen AA-paristojen sijaan. Ethgw sisältää myös ethernet-yhteyden mahdollistavan mikrokontrollerin.

Simple-alustan sydän on Microchipin valmistama 8-bittinen PIC18LF8722-mikrokontrolleri, joka sisältää 3,9 kt data-, 128 kt ohjelma- ja 1 kt EEPROM-muistia. Mikrokontrolleri käyttää akkumulaattoriarkkitehtuuria ja sen kellotaajuus on 8 MHz antaen 2 MIPS:in suorituskyvyn. Kommunikointiin alustat sisältävät vähävirtaisen Nordic Semiconductorsin 2,4 GHz nRF24L01+-radion. Lisäksi alustat sisältävät Texas instrumentsin TMP-102 lämpötila-anturin ja Avago technologiesin APDS-9002 valoisuusanturin. Näiden lisäksi alustalla on tehojärjestelmä, joka koostuu erinäisistä regulaattoreista.



Kuva 3.5: TUTWSN-anturilaitteen simple-alusta. Alustan sisältää mikrokontrollerin ja radion lisäksi useita erilaisia antureita. Käyttöliittymänä toimivat 2 lediä ja painokytkin.

4. SENSOR OS:N KÄYTTÖÖNOTTO TUTWSN-ANTURIVERKOSSA

SensorOS otettiin käyttöön vain reitittäville anturilaitteille. Mobiili-nodejen ohjelmiston rakenne on yksinkertaisempi, joten käyttöjärjestelmän käyttö ei ole tarpeen.

SensorOS:n käyttöönotto suoritettiin kahdessa vaiheessa. Ensin toteutettiin kaikki muutokset, jotka oli mahdollista toteuttaa ilman käyttöjärjestelmän vuorontajan käyttöönottoa. Niihin muutoksiin kuuluivat mm. radion ajurin muuttaminen keskeytyspohjaiseksi, tietorakenteiden muuttaminen dynaamisemmiksi ja SensorOS:n oheispalveluiden käyttöönotto. Toisessa vaiheessa otettiin varsinainen käyttöjärjestelmä käyttöön. Tällöin tehtiin muutoksia lisäksi järjestelmän hallinnointikerrokseen ja siirrettiin protokollapinon suoritus käyttöjärjestelmän vastuulle. Syitä näin toimimiseen oli useita. Tekemällä muutokset mahdollisimman pienissä osissa varmistettiin järjestelmän toimivuus paremmin. Suurten muutosten tekeminen kerralla saattaa johtaa helpommin toimimattomaan järjestelmään. Lisäksi saatiin paremmin selville varsinaisen käyttöjärjestelmän käyttöönoton edut, koska vertailukohtana oli aiemman TUTWSN:n version lisäksi ensimmäisen vaiheen ilman SensorOS:ää toimiva keskeytysohjatun radion sisältävä versio.

4.1 Aiemman järjestelmän solmukohdat

Käyttöjärjestelmää ei tietenkään kannata käyttää ihan vain käyttöjärjestelmän käytön vuoksi. Käyttöjärjestelmä vie itsessään resursseja ja joissain yksinkertaisissa tapauksissa saattaa myös turhaan monimutkaistaa järjestelmää. TUTWSN:ssä on kuitenkin tiettyjä ongelmia, joita käyttöjärjestelmän käytöllä pyrittiin ratkaisemaan. Lisäksi käyttöjärjestelmä on varautumista tulevaisuutta varten. Selkeään ja tehokkaasti toteutettuun järjestelmään on helpompi tehdä muutoksia ja lisätä uusia ominaisuuksia. Seuraavassa on kerrottu TUTWSN:n rakenteellisista ongelmista.

4.1.1 Aktiivinen odotus

Aikaisempi TUTWSN on niin sanottu silmukkarakenteen omaava sulautettu järjestelmä. Silmukkarakenne koostuu ikuisesti suoritettavasta while-silmukasta, jonka sisällä suoritetaan järjestelmän eri toimintoja. TUTWSN:ssä nämä eri toimin-

not ovat eri ohjelmistokerroksia, joita suoritetaan vuorotellen. Radion vastaanotto hoidetaan aktiivisella odotuksella radion keskeytyssignaalia jatkuvasti pollaamalla. Koska kerrosten suoritus tapahtuu vuorotellen, ei järjestelmässä juurikaan ole rinnakkaisuutta. Ainoastaan tiettyjen oheislaitteiden, kuten kiihtyvyysanturin tai painokytkimen, toimintaa on mahdollista ohjata keskeytyksillä, mikä tuo mukanaan rinnakkaisuutta järjestelmään.

Rinnakkaisuuden lisääntyminen ei välttämättä ole etu. Se monimutkaistaa järjestelmää ja aiheuttaa vaikeasti selvitettäviä ongelmia. Tässä tapauksessa pinon suoritusaika on aina pois radion kuunteluajasta. Koska reitittävien nodejen kommunikointi ei ole millään tavalla synkronoitua, aiheutti pinon suoritus aina epäonnistuneita lähetyksiä.

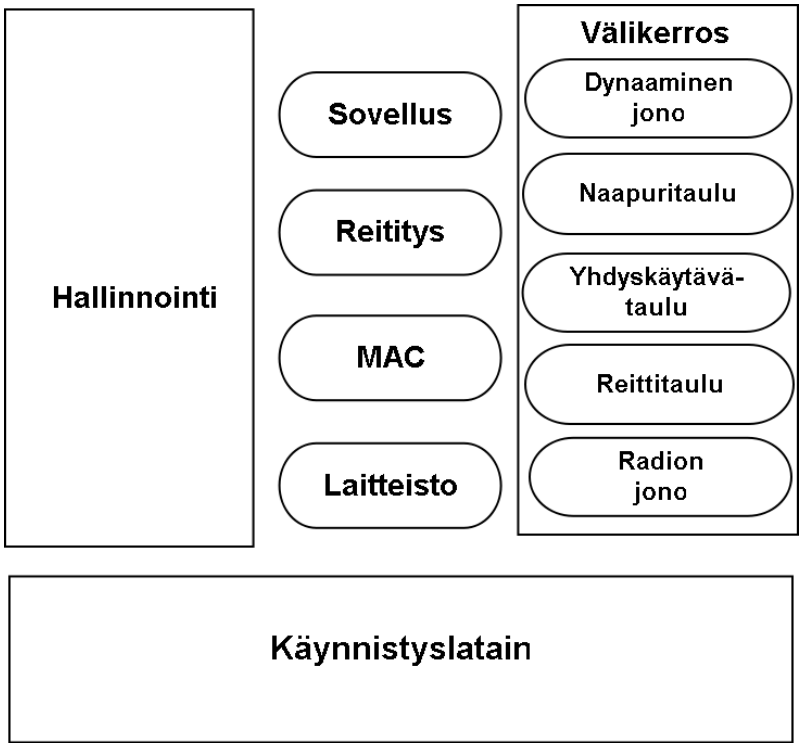
Aktiivinen odotus lisää myöskin virrankulutusta. Mikrokontrolleri on tällöin suoritustilassa, mutta ei tee mitään järkevää. Aktiivisen odottamisen poistaminen saattaisi lisätä myös nodejen elinikää.

4.1.2 Rakenteen monimutkaisuus

TUTWSN-verkon reitittävän anturilaitteen toimintaperiaate on suhteellisen yksinkertainen. Ainoa aikakriittinen toimenpide on kuittauksen lähetys vastaanotetusta paketista. Reititin-noden normaalitoiminta on käytännössä jatkuvaa radion kuuntelua paketteja käsitellen. Kuitenkin kooditasolla rakenne on aiemmassa järjestelmässä turhan monimutkainen. Monimutkaisuus johtuu osin historiasta: kooditasolla on toteutettu eri toteutusvaihtoehtoja, joita ei kuitenkaan käytetä. Lisäksi järjestelmään on lisätty ominaisuuksia vähitellen. Itse rakenne voisi olla yksinkertaisempi ja tässä asiassa käyttöjärjestelmä voisi olla avuksi. Esimerkiksi viestinvälityksen voisi siirtää staattisesta pakettijonosta IPC-viesteillä tapahtuvaksi. Lisäksi esimerkiksi hallinnointikerros on toteutettu tarpeettoman monimutkaiseksi. Tätä voisi yksinkertaistaa jo ilman käyttöjärjestelmän apua.

4.1.3 Resurssien käyttö

Aikaisemmassa järjestelmässä suurimmat tietorakenteet, kuten pakettijono ja naapuritaulu, ovat staattisesti varattuja. Ongelma ei ole järin suuri, sillä datamuistia on vapaana. Kuitenkin tulevaisuutta ja uusia ominaisuuksia ajatellen muistin käyttö olisi järkevää pitää mahdollisimman tehokkaana. Esimerkiksi naapuritaulun koko on 10 naapuria, mikä ei toteudu kovinkaan useissa käytännön verkoissa. Muuttamalla järjestelmän tietorakenteita dynaamisemmiksi saataisiin resursseja varattua muuhun käyttöön ja lisäksi sopeuttamaan verkko kulloiseenkin tilanteeseen niin hyvin kuin se resurssien rajoitteissa on mahdollista.



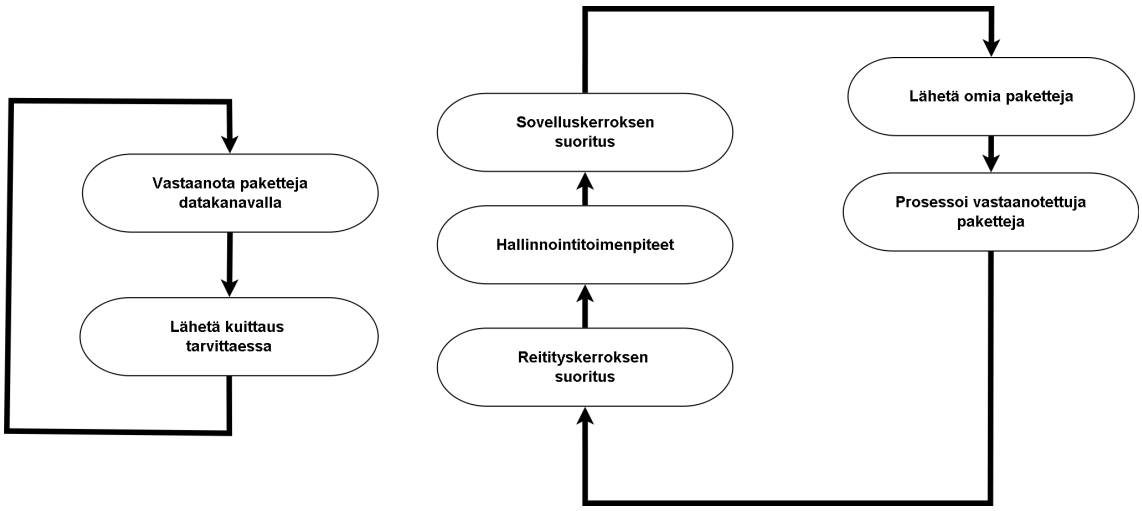
Kuva 4.1: TUTWSN:n anturilaitteen ohjelmiston rakenne ensimmäisen vaiheen muutosten jälkeen. Dynaaminen prioriteettijono ja radion jono ovat tulleet staattisen pakettijonon tilalle. Lisäksi järjestelmä sisältää käynnistyslataimen.

4.2 Ensimmäisen vaiheen muutosten kuvaus

Ensimmäisessä vaiheessa tehtiin esimerkkijärjestelmään muutokset, mitkä oli mahdollista tehdä ilman varsinaisen käyttöjärjestelmäytimen käyttöönottoa. Tässä vaiheessa otettiin käyttöön käyttöjärjestelmän tarjoamia palveluita. Muutokset olivat laajuudeltaan jopa suurempia kuin toisessa vaiheessa tehdyt. Kuvassa 4.1 on kuvattu järjestelmän ohjelmiston rakennetta ensimmäisen vaiheen muutosten jälkeen. Kuvassa näkyy erona vanhaan järjestelmään (kuva: 3.2) staattisen jonon korvaantuminen dynaamisella prioriteettijonolla ja radion jonon sekä käynnistyslataimen lisääminen järjestelmään. Seuraavassa on kerrottu tärkeimmistä tehdyistä muutoksista.

4.2.1 Radioajuri keskeytysohjatuksi

Radioajuri on vanhassa järjestelmässä toteutettu ilman keskeytyksiä. Ajurista on olemassa myös keskeytysohjattu versio, mutta se käyttäytyy hyvin samalla tavalla kuin ilman keskeytyksiä toimiva versio: keskeytyslipun sijaan pollataan muuttujaa, jonka arvoa muutetaan keskeytyksen palveluohjelmassa. Tällöin rinnakkaisuutta ei juuri ole, joten vastaanoton aikana ei voi suorittaa protokollapinoa.



Kuva 4.2: TUTWSN-pinon suorituskaavio ensimmäisen vaiheen muutosten jälkeen. Pakettien vastaanotto on siirtynyt muun pinosuorituksen kannalta rinnakkaiseksi toimenpiteeksi. Käytännössä vastaanotto onkin oikeasti rinnakkainen toimenpide, mutta kuittausten lähetys hoidetaan radion jonoa pinon suorituksen yhteydessä jatkuvasti pollaamalla. Kuvassa kuittausten lähetys on selkeyden vuoksi pakettien vastaanoton yhteydessä oleva toimenpide.

Ajuri muutettiin rinnakkaisuutta tukevaksi lisäämällä funktiot vastaanoton päälle- ja poiskytkemiselle. Kuvassa 4.2 on nähtävissä radion uusi toteutus osana muuta järjestelmää. Järjestelmän käynnistyksen yhteydessä kytketään vastaanotto päälle ja vastaanotto on päällä käytännössä aina lähetyksiä lukuun ottamatta. Vastaanoton poiskytkentä tehdään mahdollisimman matalalla tasolla lähetysfunktiossa.

Radioajurille toteutettiin lisäksi staattinen neljän radiopaketin pituinen puskuri, jota täytetään radion keskeytyksen palveluohjelmassa. Jonon pituus tulee mobiilinoden paikannusviestien määrästä. Jono on LIFO-tyyppinen, sillä näin saadaan mahdollisimman suuri määrä kuittauksia perille. FIFO-tyyppisen jonon ensimmäisen alkion kuittaus kun on jo mahdollisesti myöhässä pakettia käsiteltäessä, mikäli jonossa on muita alkioita. Jonon käsittely hoidetaan aikakriittisiä toimenpiteitä tarkkailevassa funktiossa. Tätä funktiota kutsutaan niin usein kuin se on mahdollista, jotta kuittaukset pystyttäisiin lähettämään ajoissa. Reitittävällä nodella kutsuväli on keskimäärin n. 2 ms:n luokkaa ja yhdyskäytävällä n. 3 ms. Alkuperäisessä järjestelmässä lähetetään kuittaus 5 ms:n sisällä vastaanotosta, mutta tämä aika jouduttiin nostamaan 7 ms:n, jotta kuittaukset pystyttäisiin lähettämään ajoissa. Näin estetään turhat uudelleenlähetykset. Paketin käsittely keskeytyksen palveluohjelmassa olisi liikaa aikaa syövä toimenpide. Ideaalitilanteessa paketti käsiteltäisiin DSR:ssä, mutta DSR-jonoa käsitellään noin sadan millisekunnin välein, mikä tarkoittaisi käytännössä sitä, että kuittausten lähetykseen tulisi asettaa

yli sadan millisekunnin viive seitsemän millisekunnin sijaan.

Lähetyksen toteustapaa ei muutettu. Paketit lähtevät edelleen ilman keskeytyksiä. Keskeytysten käyttö olisi monimutkaistanut järjestelmää ja tuonut haittoihin nähden vähän etua, sillä lähetys on nopea vain noin millisekunnin kestävä toimenpide.

Rinnakkaisuuden hallintaan toteutettiin älykäs atomisuuden hallinta. Perinteisesti siirryttäessä atomiseen tilaan kytketään keskeytykset pois ja siirryttäessä pois atomisesta tilasta kytketään keskeytykset päälle. Toteutetussa atomisuusratkaisussa kytketään keskeytykset pois tilaan siirryttäessä, mutta pois siirryttäessä kytketään keskeytykset päälle vain mikäli ne olivat päällä tilaan siirryttäessä. Näin estetään peräkkäisten kutsujen aiheuttama mahdollinen virhetoiminta. Toisaalta tämä älykäs atominen tila vaatii järjestelmän suunnittelijalta tarkkaavaisuutta atomisuusoperaatioita käytettäessä, sillä yksikin virheellinen kutsu voi aiheuttaa keskeytysten pysyvän kieltämisen.

Atomisena suoritetaan radion jonoa käsittelevät operaatiot sekä SPI-väyläliikenne mikrokontrollerin ja radion välillä. Tämä johtuu siitä, että molempia atomisena suoritettavia operaatiota suoritetaan keskeytyksen palveluohjelmassa.

4.2.2 Dynaaminen prioriteettijono ja sisäinen viestin välitys

Aikaisemman järjestelmän viestin välitys ja pakettien käsittely hoidetaan staattisen pakettijonon kautta. Sovellusten luomille paketeille varataan järjestelmässä silti ensin dynaamisesti muistia ja paketin sisältö kopioidaan tämän jälkeen staattiseen pakettijonoon. Tämän rakenteen tilalle luotiin dynaaminen prioriteettijono, jonka pituus vaihtelee vapaana olevan keon muistin määrän suhteen. Jono on rakenteeltaan linkitetty lista, jonka muisti varataan dynaamisesti muistinhallintayksiköltä. Jonon alkion rakenne on tarkemmin kuvattu listauksessa 4.1. Alkio sisältää kohdekerroksen ja kohdeosoitteen lisäksi tietoja paketin lähetykseen vaikuttavista asioista kuten liikennöintiluokasta. Kohdekerrokset on jaettu järjestelmän eri kerrosten mukaan. Ainoastaan MAC-kerroksella on kolme erityyppistä kohdetta:

MAC_Own: Tähän kohteeseen lähetetään prosessointia odottavat paketit. Esimerkiksi radion jonon paketteja käsiteltäessä paketit lähetetään mahdollisen kuittauksen lähetyksen jälkeen tähän kohteeseen.

MAC_Nbor: Lähetystä odottavat paketit. Paketteja ei enää jatkossa käsitellä vaan ne lähetetään haluttuun aikaan kohdeosoitteeseen.

MAC_LocationNode: Kohteena mobiili-node. Reitittävillä nodeilla on tieto viime hetkinä kuulluista mobiili-nodeista. Pakettia yritetään lähettää mobiili-nodelle, mikäli tältä vastaanotetaan paikannusviesti.

Jokaiselle kerrokselle luotiin lisäksi postilaatikko IPC-viestejä varten. Käytännössä postilaatikko on linkitetty lista, joka sisältää kyseiselle kerrokselle lähetettyjen

käsittelemättömien viestien osoittimet. Esimerkiksi lähetettäessä viestiä reitityskerrokselle viesti kulkee ensin viestien välittäjämoduulin, joka kutsuu vastaanottajan vastaanottofunktiota, joka lisää kyseisen viestin kerroksen postilaatikkoon. Postilaatikon viestejä käsitellään kerrosta suoritettaessa.

```
1  typedef struct StackQueue_Item {
     //Seuraavan alkion osoite
3    sl_list_t *next;
     //Alkion id. Sisältää tiedon mm. kohdekerroksesta
5    uint8_t  id;
     //Aika sekunteina, jolloin alkio lisättiin jonoon
7    uint16_t tm;
     //Palveluluokka (engl. Quality of Service Class)
9    Mac_pdu_qos_t macQosClass;
     //Kohdeosoite
11   Mac_addr_t target;
     //Varsinainen tietosisältö
13   PDU pdu;
     //Maksimi viive vastaanotosta
15   uint32_t maxDelayUs;
     //Uudelleenlähetysten määrä
17   uint8_t reTxCnt;
     //Poistetaanko alkio onnistuneen lähetyksen jälkeen?
19   bool removeOnAck;
   } StackQueue_Item;
```

Listaus 4.1: Dynaamisen prioriteettijonon tietotyyppi.

4.2.3 Järjestelmän luotettavuuden parantaminen

Järjestelmän luotettavuutta parannettiin ottamalla korkean tason vahtikoira käyttöön. Korkean tason vahtikoira alustaa järjestelmän, mikäli vahtikoiran ajastinta ei ole nollattu kymmenen minuutin kuluessa. Vahtikoira nollataan järjestelmässä vastaanotettaessa paketteja ja onnistuneiden lähetysten yhteydessä.

Vahtikoiran toiminta perustuu ajastinkeskeytyksiin. Järjestelmän alustusoperaatio suoritetaan ajastimen keskeytyksenpalveluohjelmassa, johon hypätään ajastimen ylivuodon sattuessa. Nollattaessa vahtikoira asetetaan ajastin alustusarvoonsa.

Korkean tason vahtikoiran käyttöön liittyy riskitekijöitä. Mikäli järjestelmän keskeytykset ovat kiellettyjä, mikä on hyvälläkin suunnittelulla mahdollista tapahtua atomisia operaatioita käsiteltäessä, vahtikoira ei toimi. Tämän takia järjestelmässä on toiminnassa mikrokontrollerin laitteistotasolla toteutettu vahtikoira, jonka toimintaan eivät keskeytykset vaikuta.

4.2.4 Käynnistyslatain

Ensimmäisessä vaiheessa otettiin käynnistyslatain käyttöön. Tämä ei varsinaisesti ole käyttöjärjestelmän komponentti, vaikka useassa käyttöjärjestelmässä se voisi sitä olla. Käynnistyslatain on TUTWSN:n muusta järjestelmästä täysin erillinen ohjelmistokomponentti. Käynnistyslataimen avulla pystytään järjestelmän ohjelmisto päivittämään langattomasti. Päivitys tapahtuu koko ohjelmistoon käynnistyslatainta lukuun ottamatta. Eli esimerkiksi käyttöjärjestelmää käytettäessä myös käyttöjärjestelmä päivittyy.

Käynnistyslataimen toiminta perustuu käännösvaiheessa alustan ja ohjelmistokomponenttien perusteella luotaviin sarjanumeroon ja ohjelmistoversioon. Sarjanumeron perustella erotellaan eri laitteistoversiot ja esimerkiksi anturilaitetyypit (esimerkiksi: reitittävä, mobiili) toisistaan. Anturilaite hyväksyy vain saman sarjanumeron ohjelmistoja, kuin mitä sillä itsellä on.

Käynnistyslataimen toiminta eroaa hieman tavallisilla reitittävillä nodeilla ja yhdyskäytävillä. Reitittävät nodet mainostavat vain omaa ohjelmistoaan. Yhdyskäytävät eroavat tavallisista reitittävistä nodeista siinä, että ne päivittävät oman ohjelmistonsa yhdyskäytävän kautta, eivätkä mainosta täten omaa ohjelmistoaan muille. Yhdyskäytävät mainostavat kuitenkin palvelimelta löytyviä anturilaiteohjelmistoja muille reitittäville nodeille.

Langaton päivitys toimii reitittävillä nodeilla seuraavasti: Sarjanumeroa ja versiota mainostetaan muille anturilaitteille verkkokanavalla verkkoviestien lähetyksen yhteydessä niin sanotuilla ohjelmistoviesteillä. Näiden numeroiden perusteella pystyy vastaanottava anturilaite päättelemään, onko mainostettu ohjelmisto sille sopiva ja uudempaa versiota kuin nykyinen. Vastaanottava anturilaite lähettää lähettävälle anturilaitteelle kuittauksen, mikäli ohjelmisto on sille uusi ja sopiva. Tämän jälkeen se siirtyy niin sanottuun käynnistyslataintilaan odottamaan lähettävältä anturilaitteelta ohjelmistoa. Mikäli lähettävä anturilaite vastaanottaa kuittauksen se siirtyy myös käynnistyslataintilaan lähettämään mainostamaansa ohjelmistoa. Mikäli kuittauksen lähetys epäonnistuu, molemmat laitteet jatkavat toimintaansa normaalisti muutaman sekunnin kuluttua.

4.2.5 Ensimmäisessä vaiheessa esiintyneitä ongelmia

Rinnakkaisuus aiheutti useita vaikeasti paikannettavia ongelmia. Merkittävimpänä ongelmana oli sekä ohjelmistopinossa että mikrokontrollerin laitteistopinossa tapahtuneet ylivuodot. Nämä aiheuttivat satunnaisesti tapahtuneita järjestelmän nollautumisia, joiden syy oli vaikeasti määritettävissä. Kutsupuun pituutta arvioitiin aikaisemmin tutkimuskäyttöön tehdyllä skriptillä, joka toisaalta ei osannut ottaa keskeytyksiä tai funktio-osoittimia huomioon ja se sisälsi noin 10 prosentin virheen

[4]. Skripti antoi järjestelmän ohjelmistopinon maksimikooksi 220 tavua, minkä ei vielä pitäisi aiheuttaa ongelmia pinon maksimikoon ollessa 256 tavua. Kuitenkin virherajat ja keskeytykset huomioon ottaen ylivuoto oli mahdollista, joten pinon kokoa kasvatettiin 384 tavuun. Määrittämättömät nollautumisongelmat katosivat pääosin tämän myötä.

Laitteistopinon ylivuodosta päästiin muuttamalla järjestelmän rakennetta. Alun perin kuittausten lähetyksen yhteydessä tapahtui myös paketin välitys seuraavalle kohteelle, mikä aiheutti pahimmassa tapauksessa liian suuren kutsupuun pituuden. Välitys siirrettiin tapahtuvaksi MAC-kerroksen suorituksen yhteyteen, jolloin kutsupuun pituus pieneni ja ongelmat katosivat.

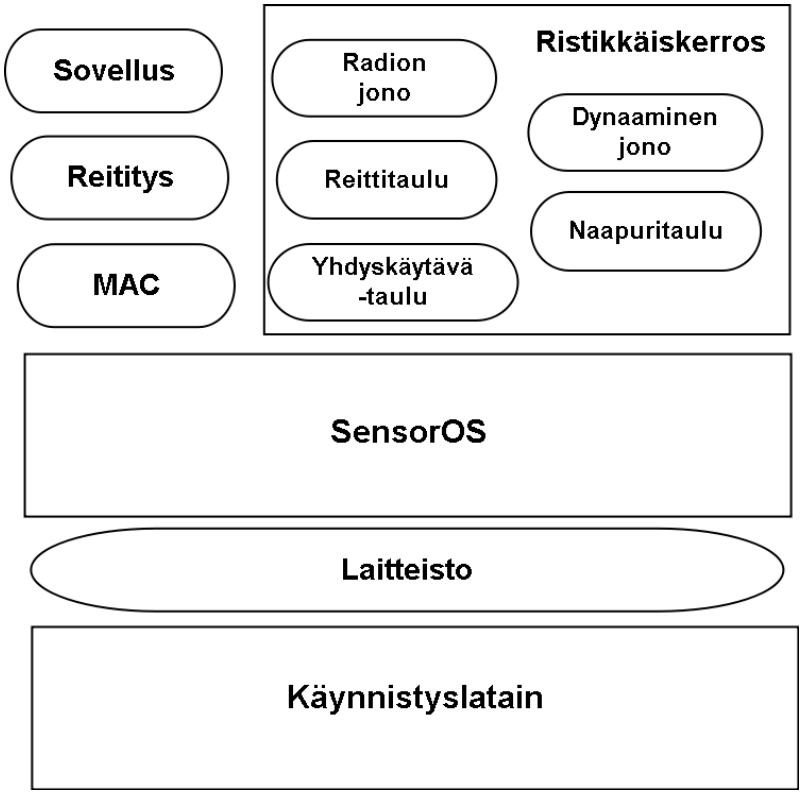
4.3 Toisen vaiheen muutosten kuvaus

Toisessa vaiheessa ohjelma paloiteltiin osiin säikeiksi ja prosesseiksi. Suorituksesta tuli oikeasti eri sovelluskerrosten tasolla rinnakkaista. Hallinnointikerros siirtyi kiinteäksi osaksi MAC- ja reitityskerroksia ja signaloinnin sijaan siirryttiin käyttämään IPC-viestejä. Merkittävä muutos oli myös radion jonon pollaamisesta luopuminen, sillä käyttöjärjestelmän avulla oli mahdollista toteuttaa riittävän nopeasti suoritettava DSR tälle. Ohjelman rakennetta muutoksien jälkeen on kuvattu kuvassa 4.3. Vaiheeseen 1 verrattuna merkittävin muutos on hallinnointikerroksen häviäminen ja SensorOS:n tuleminen osaksi järjestelmää.

Toisen vaiheen muutosten toteutus jäi aikaresurssien takia osin kesken. Esimerkiksi sovelluskerroksen täydellisen toiminnallisuuden siirtäminen käyttöjärjestelmän vastuulle jäi toteuttamatta. Järjestelmästä saatiin toteutettua kuitenkin toimiva prototyyppi, jolla suorituskykyvertailujen teko oli mahdollista. Kesken jääneet osat eivät vaikuta merkittävästi suorituskykyyn, vaan tuovat anturiverkkoon lähinnä lisätoiminnallisuutta.

4.3.1 Pinon suoritus säikeiksi

Järjestelmä oli tarkoitus jakaa prosesseiksi tarkalleen kerrosrajojen mukaan. Tällöin prosessit toteuttaisivat selkeän toiminnallisen kokonaisuuden. SensorOS:n säikeet on suunniteltu lähinnä käytettäväksi sovelluksissa, joten säikeitä ei ennakoitu tarvittavan muilla kuin sovelluskerroksella. Säikeiden monikäyttöisyyttä vähentää lisäksi niiden ohjelmointirajapinta. Kutsuttaessa säikeessä säikeille tarkoitettua tapahtuman odottamisfunktiota takia voidaan suoritus odottamisen aikana vaihtaa toiseen säikeeseen tai prosessiin. Kyseinen funktio vaatii kuitenkin lisäksi parametrikseen säikeen suoritushetken sisältävän muuttujan. Prosesseilla vastaava odottaminen on toteutettu parametrittomalla funktiolla. Kuitenkin kutsuttaessa kyseistä funktiota säikeessä keskeytyy koko prosessin suoritus eikä vain kutsuvan säikeen. Tapah-



Kuva 4.3: TUTWSN:n sulautetun ohjelmiston rakenne käyttöjärjestelmän käyttöönoton jälkeen.

tumien odotus säikeissä on järkevää siis vain ylimmällä säikeen suorituksen tasolla, sillä muuten joudutaan suoritushetkimuuttujaa kuljettamaan alemmille tasoille parametrinä. Tämä mutkistaa suunnittelua, sillä säikeitä ei voida käyttää joka käyttötapauksessa, ja koska prosessit toisaalta kuluttavat muistia kontekstin vaihdon takia, niitä ei voida luoda määräänsä enempää. Toisaalta prosesseillakaan odotus ei ole ongelmatonta, sillä odotettaessa tapahtumaa on tiedettävä, mitkä tapahtumat on prosessille asetettuna tai muuten voi joitain tapahtumia jäädä huomioitta.

Prosessijakoa mutkisti lisäksi SensorOS:n ongelmallinen ajastimien käyttö. Matalan tarkkuuden ajastimia voi luoda vain yhden kutakin prosessia/säiettä kohti. SensorOS:ssä säikeitä on suunniteltavaksi käytettävän lähinnä sovelluskerroksella, jolloin säikeen ajastin toimisi sovelluksen ajastimena. Kuitenkin lähes kaikilla pinon kerroksilla on tarvetta useammille ajastimille. Järjestelmässä on useita ajastettuja palveluita sekä esimerkiksi sovelluskerroksella jotkin sovellukset vaativat useamman ajastimen. Koska korkean tarkkuuden ajastin on pyhitetty MAC-kerroksen käyttöön sitä ei voi käyttää muualla. Säikeen luominen jokaista palvelua kohti kuluttaisi tarpeettomasti muistia ja monimutkaistaisi mahdollisesti järjestelmää. Yhdellä ajastimellakin tulee toimeen, mutta sen käyttö monimutkaistaa tarpeettomasti järjestelmää. Aina asetettaessa ajastin tulee poistaa edellinen ajastin, mikä johtaa tilanteisiin, jossa voidaan uuden ajastimen laukeamisen jälkeen joutua asettamaan

Taulukko 4.1: Toisessa vaiheessa järjestelmä pilkottiin prosesseiksi. Prosessit noudattelevat kerrosrakennetta.

PID	Prosessi	Selitys	Pinon koko (tavua)	Säikeitä
0	MAC	MAC-kerrokseen liittyvät aikriittiismmät toimenpiteet	256	-
20	Reititys	Reitityskerros sekä pakettien käsittely	256	1
40	Sovellus	Sovellusten vuorontaja sekä sovellukset	192	Sovellusten määrä
60	Tyhjäkäynti	Tyhjäkäyntiprosessi	128	-

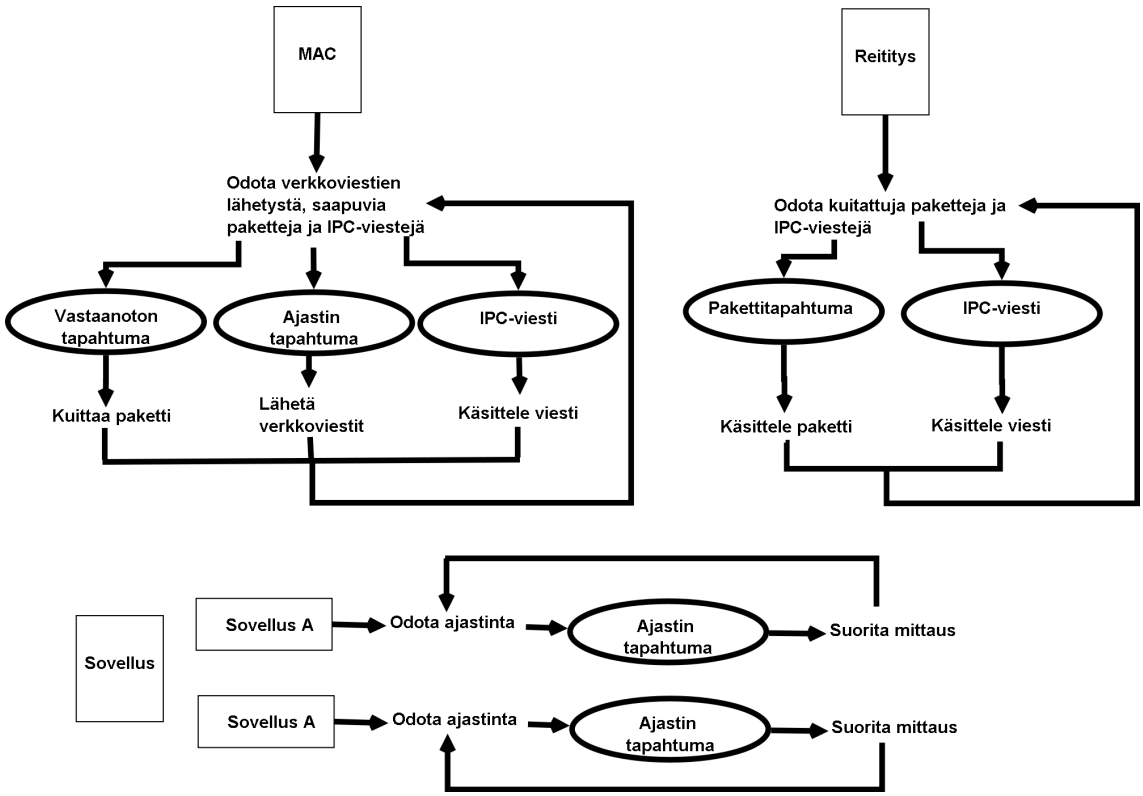
vanha ajastin uudelleen, mikäli se yleensä on mahdollista. Kolmas epätarkka ja monikäyttöinen ajastinpalvelu olisi erittäin hyödyllinen.

Ongelmista huolimatta järjestelmä saatiin paloiteltua prosesseiksi ja säikeiksi. Taulukossa 4.1 on esitetty järjestelmän prosessijako, joka ei kuitenkaan mennyt aivan kerrosrajojen mukaan. Järjestelmä sisältää kolme varsinaisen toiminnallisen kokonaisuuden sisältävää prosessia ja lisäksi tyhjäkäyntiprosessin, jota suoritetaan, kun mikään muu prosessi ei ole suoritusvuorossa. Kuvassa 4.4 on esitetty lisäksi ohjelman suorituskaavio toisen vaiheen muutosten jälkeen. Kuvasta nähdään suorituksen muuttuneen rinnakkaiseksi.

Prosessille varataan luomisen yhteydessä staattisesti sen käyttämän prosessipinon muisti. Lisäksi on käynnistettävä säievuorontaja ja osoitettava vuorontajalle prosessin säikeet, mikäli säikeitä käytetään. Prosessille asetetaan luomisen yhteydessä myös sen ID, jonka perusteella suoritusta vaihdetaan. Mitä pienempi ID, sitä korkeampi prioriteetti.

Aikakriittinen prosessi sisältää MAC-kerroksen osat pakettien jatkokäsittelyä ja lähetystä lukuun ottamatta. Näihin kuuluvat vastaanotettujen pakettien kuittausten lähetys ja verkkoviestien lähetys. Prosessista käytetään tästä syystä jatkossa nimitystä MAC-prosessi. MAC-prosessi sisältää lisäksi hallinnointikerrokselta MAC-kerrosta ohjaavat toimenpiteet, joihin kuuluu lähinnä verkkoskanneista päättäminen. Prosessi sisältää lisäksi postilaatikon IPC-viesteille. Prosessin pinon kooksi asetettiin kokeilujen perusteella 256 tavua. 192 tavun pinon havaittiin olevan prosessille liian pieni.

Pakettien jatkokäsittely ja lähetys jouduttiin siirtämään alemman prioriteetin prosessille. Tämä johtuu siitä, että aikakriittiselle prosessille ei voi luoda säikeitä aikakriteerien toteutumisen takia. Tämä estää esimerkiksi pakettien kuittausten ja pakettien lähetysten suorittamisen samassa prosessissa nykyistä lähetystapaa käyt-



Kuva 4.4: Yksinkertaistettu versio järjestelmän suorituksesta toisen vaiheen jälkeen. Kunkin prosessin suoritus on periaatteessa rinnakkaista. Käytännössä suoritus tapahtuu kuitenkin sekventiaalisesti, sillä käytössä on yksiprosessorijärjestelmä.

täen. Paketteja lähetettäessä joudutaan odottamaan lähetysaikaa ja odotus estää prosessin muiden toimintojen suorituksen odottamisen aikana, sillä toiminnolle ei voida aikakriittisessä prosessissa luoda omaa säiettä. Lähetystoimenpide voi pahimmilleen kestää yli sekunnin uudelleenlähetyskertojen määrästä riippuen ja näin pitkä tauko vastaanotossa on protokollan kannalta käytännössä sietämätön.

Aikakriittisen prosessin ohjelmakoodi on esitetty hieman yksinkertaistettuna listauksessa 4.2. Prosessin alussa asetetaan halutut tapahtumat ja tämän jälkeen odotetaan kyseisiä tapahtumia ikuisessa silmukassa. Radion vastaanottoa vastaava tapahtuma laukeaa radion keskeytyksenpalveluohjelmassa vastaanotettaessa paketteja. Kuittausfunktiossa lähetetään vastaanotetulle paketille tarvittaessa kuittaus ja siirretään se dynaamiseen prioriteettijonoon. Verkkoviestien lähetyksen ajastuksesta huolehtii hälytysajastin, joka laukeaa noin kahden sekunnin välein. Samalla nollataan myös vahtikoiran ajastin. Verkkoviestejä lähetetään, mikäli järjestelmä sen sallii. Näin on silloin, kun hyväksyttävä reitti on tiedossa.

Suuren tarkkuuden ajastinta käytetään ainoastaan MAC-kerroksella tarkkuutta vaativiin toimenpiteisiin, mikä tässä tapauksessa tarkoittaa kuittausten vastaanottoa ja paketin lähetystä.

Toiseksi korkeimman prioriteetin prosessiin siirrettiin reitityskerroksen toiminnot. Näihin kuuluu käytännössä reitin muodostus ja reitinmainostuspakettien lähetys, joiden suorituslogiikka tulee hallinnointikerrokselta. Lisäksi prosessiin siirrettiin edellä esitetyistä syistä pakettien käsittely.

```
void time_process(){
  //Asetetaan radion tapahtuma
  os_event_assign( os_radio_get_event(), 0);
  //Asetetaan IPC-viestien tapahtuma
  os_event_assign(os_ipc_get_event(), 0);
  //Satunnaistetaan verkkoviestien lähetysaika
  nbSendTime=2000+randomSlot();
  //Ajastetaan verkkoviestien lähetysaika
  os_alarm_set(nbSendTime);

  while(1){
    event=os_event_wait(EVENT_REACT_PENDING); //odotetaan tapahtumia
        ikuisessa silmukassa
    if( event == 0 ){
      assert(1==0);
    } else if (event ==os_ipc_get_event()) {
      //Tee viestin vaatimat tehtävät
      processMessages();
      //Valitaan tapahtuma IPC-viesteille uudelleen
      os_event_assign(os_ipc_get_event(), 0);
    } else if (event ==os_radio_get_event()) {
      atomic_on();
      prosessoiRadionJonoa();
      atomic_off();
    } else if( event == os_alarm_get_event() ) {
      nbSendTime=2000+randomSlot();  //Satunnaistetaan verkkoviestien l
          ähetysaika
      if(Mgmt_sendAllowed()){ //Lähetetään verkkoviestejä, mikali lä
          hetys sallittu
        Locmac_relay_SendNBs();
        atomic_on();
        Radio_setReceiveOn(Locmac_LOCATION_CHANNEL_COMPILED);
        atomic_off();
      }
      CLR_WDT(); //Nollataan vahtikoira
      os_alarm_set(nbSendTime); //asetetaan ajastin verkkoviesteille
    }
  }
}
```

Listaus 4.2: MAC-kerroksen suoritus yksinkertaistettuna.

Kolmas prosessi huolehtii sovelluskerroksen toiminnoista. Se sisältää ajastimet

sovellusten suorittamiseen ja huolehtii sovelluksiin liittyvistä toimenpiteistä. Prosessi sisältää säikeen hallinnollisten toimenpiteiden suorittamiseen ja lisäksi oman säikeen jokaiselle sovellukselle.

Lisäksi järjestelmälle toteutettiin niin sanottu tyhjäkäyntiprosessi. Tätä suoritetaan silloin, kuin mikään muu prosessi ei ole suoritusvuorossa. Tyhjäkäyntiprosessi on tällä hetkellä vain ikuisesti suoritettava silmukka, mutta siihen olisi helppo toteuttaa esimerkiksi virransäästötoiminnallisuus asettamalla prosessia suoritettaessa mikrokontrolleri unitilaan.

4.3.2 Hallinnointikerros käyttöjärjestelmän vastuulle

Hallinnointikerros muutettiin joukoksi IPC-viestejä, joita tarkastellaan prosesseissa. Vanha hallinnointikerros oli kevyt toteutukseltaan, mutta tarpeettoman monimutkainen. Järjestelmän suoritusrakenne on muutoksen jälkeen helposti nähtävissä prosessien ylimmän tason ohjelmakoodista. Suoritus koostuu muutamista lähes jatkuvasti suoritettavista toimenpiteistä ja IPC-viestien muodostamien signaalien sekä ajastimien laukaisemista kertaluontoisista toistuvista toiminnoista.

4.3.3 Muistinhallinta

Muistinhallinnassa siirryttiin käyttämään hakutaulutoteutusta. Aikaisemmassa TUTWSN:n versiossa varataan dynaamisesti muistia vain IPC-viestejä ja naapuritaulun alkioita varten. Näiden koko on lähes vakio, joten hukkamuistia aiheutuu lähinnä IPC-viestejä pienempien naapuritaulun alkioiden koon vuoksi. Käyttöjärjestelmää käytettäessä siirryttiin käyttämään signalointiin lyhyitä IPC-viestejä. Vaihteleva muistinvarausyksikön koko on tarpeen, jotta hukkamuisti pysyisi pienenä. Muistinvarausyksiköllä on kolme kokoa: 5 tavua, 29 tavua ja 42 tavua. Ensimmäinen on tarkoitettu lyhyille IPC-viesteille, toinen naapuritaulun alkioille ja kolmas normaaleille IPC-viesteille.

4.3.4 Ajurit

Radion ajuri on lähes sama kuin vaiheessa 1 toteutettu. Lähetettäessä paketteja joudutaan kuittauksen vastaanotossa varaamaan radio prosessin käyttöön korkeimman prioriteetin prosessilta. Muuten keskeytyksen sattuessa suoritus siirtyisi ensin aikakriittisen prosessiin ja vasta tämän jälkeen kuittauksen tarkasteluun. Lisäksi ongelmana olisivat rinnakkaisuuden aiheuttamat tiedonsiirto-ongelmat radion kanssa. Radion varaaminen on toteutettu viestien avulla. Aikakriittiselle prosessille lähetetään viesti, jossa ilmoitetaan tarpeesta radion käyttöön. Tällöin aikakriittinen prosessi luovuttaa radion hallinnan itseltään. Lisäksi radion keskeytykselle

toteutettiin tapahtuma, joka laukeaa vastaanottokeskeytyksen sattuessa. Koska järjestelmän tapahtumankäsittelyyn ja kontekstinvaihtoon kuluva aika on satojen mikrosekuntien luokkaa, pystytään vaiheen 1 pollausrakenteesta luopumaan. Aikakriittisessä prosessissa odotetaan passiivisesti käytännössä jatkuvasti tätä tapahtumaa ja täten paketin ensimmäinen käsittely toimii eräänlaisen DSR-funktiona.

Alemman tason lähetysfunktiot on toteutettu yksinkertaisesti ilman keskeytyksiä. Lähetyksen alussa kielletään keskeytykset ja lopussa keskeytyslippu nollataan varmuuden varalta, jotta turhia ja etenkin väärästä syystä eli lähetyksestä aiheutuneita keskeytyksiä ei tapahtuisi.

5. MITTAUKSET KÄYTTÖJÄRJESTELMÄN VAIKUTUKSISTA

Käyttöjärjestelmän etuja pyrittiin selvittämään myös mittaamalla ja vertailemalla tässä työssä luotua käyttöjärjestelmällistä anturiverkkoa vertailujärjestelmiin. Käyttöjärjestelmällistä TUTWSN-järjestelmää verrattiin sekä vaiheessa 1 luotuun keskeytysohjatun radion sisältävään versioon että aiempaan silmukkarakenteiseen TUTWSN-anturiverkkoon. Vertailuparametreinä toimivat TUTWSN-verkon kannalta oleelliset anturilaitteen toiminnan parametrit, muistinkulutus sekä järjestelmän rakenne.

5.1 Toiminnan sisäinen vertailu

Koska reititin-nodet eivät ole millään tavalla synkronoituja keskenään, on radioliikenne sitä luotettavampaa, mitä enemmän radio on päällä vastaanottotilassa. Mittaukset tehtiin hyvin pienellä testiverkolla, johon kuului sekunnin aikavälillä paikannusviestejä lähettävä mobiili-node, reititin-node ja yhdyskäytävä. Verkon koolla ei tässä tapauksessa juuri ole väliä - suurempi verkko voisi jopa aiheuttaa esimerkiksi kuuluvuusvaihteluista johtuvia eroja, jotka vääristäisivät mittaustuloksia.

Radion päälläolomittaus suoritettiin mittaamalla oskilloskoopilla mikrokontrollerin IO-nastaa, joka asettiin ohjelmallisesti kytkettäessä radion vastaanotto päälle. IO-nastan asettamiseen kuluva aika on merkityksettömän pieni mittaustulosten kannalta. Päälläoloaikaa mitattiin kymmenen sekunnin ajan. Mittaus suoritettiin kullakin järjestelmällä kolmeen kertaan tulosten oikeellisuuden varmistamiseksi. Tällainen lyhyt mittaus on mahdollista suorittaa luotettavasti, sillä TUTWSN:n toiminnallisuus ei juuri muutu, vaikka mittausaika kasvaisikin moninkertaisesti. Ainoa muutos olisivat harvoin toistuvat verkkoskannaukset. Lisäksi simuloitiin tilanne, jossa lähetys epäonnistui. Tämä tehtiin kytkemällä yhdyskäytävistä käyttöjännite pois päältä. Näin saatiin selville myös epäonnistuneiden lähetysten aiheuttama vaikutus radion päälläoloaikaan.

Mittaustulokset on esitetty taulukossa 5.1. Radion päälläoloaika on keskeytysohjattua radiota käyttävissä versioissa huomattavasti alkuperäistä järjestelmää parempi. Tämä johtuu osittain myös protokollapinosta, sillä lähetys aiheuttaa pitkiä katkoksia radion vastaanottoon. Vaikka protokollapinon aiheuttama ero, joka käytännössä on nähtävissä uudelleenlähetyksen ajasta, poistettaisiin, on keskeytys-

Taulukko 5.1: Radion päälläoloaikamittaukset. Keskeytysohjattu radio tuo selkeitä etuja vanhaan järjestelmään nähden. Epäonnistunut lähetys aiheuttaa keskeytysohjattua radiota käytettäessä vain 2 ms:n katkon radion vastaanottoon.

Järjestelmä	Radion päälläoloaika prosentteina	Epännistuneen lähetyksen vaikutus radion päälläoloaikaan (ms)
Alkuperäinen	80,0	150-200
Vaiheen 1 (ei OS)	94,5	2
Vaiheen 2 (OS)	94,5	2

Taulukko 5.2: Paketin käsittelyviive.

Järjestelmä	Viiveen keskiarvo(ms)	Viiveen maksimi (ms)	Viiveen minimi (ms)
Vanha	< 1	< 1	< 1
Vaiheen 1 (ei OS)	2,2	3,5	1,6
Vaiheen 2 (OS)	1,8	1,6	2,0

ohjattu radiototeutus silti muutaman prosenttiyksikön parempi radion päälläoloajan osalta. Tämä johtuu siitä, että vanhassa toteutuksessa protokollapinon suoritukseen menee joka sekunti n. 25 millisekuntia ja tällöin ei radio ole kuuntelutilassa.

Järjestelmiä vertailtiin lisäksi paketin käsittelyviiveen perusteella. Tämä viive on se, joka muodostuu vastaanottokeskeytyksestä paketin käsittelyyn. Viiveen laskeminen aloitettiin keskeytyksenpalveluohjelman suorituksen alusta. Viivettä varsinaisen keskeytyksen tapahtumisesta palveluohjelmaan siirtymiseen ei otettu huomioon. Mittaus suoritettiin vastaavalla tavalla kuin päälläolomittaus. Mikrokontrollerin IO-pinni asetettiin keskeytyksen palveluohjelman suorituksen alussa ja nollattiin paketin käsittelyyn siirryttäessä. Järjestelmille suoritettiin 20 mittausta, joista laskettiin keskiarvo. Keskeytyksenpalveluohjelman suoritus kesti noin 600 mikrosekuntia molemmilla radiokeskeytyksiä käyttävillä järjestelmillä. Taulukossa 5.2 on esitetty saadut mittaustulokset. Tuloksista näkee vanhan silmukkarakenteisen pollaavan järjestelmän edun: pollauksen ansiosta paketti käsitellään käytännössä heti. Toisaalta tuloksista näkee myös reaaliaikaisen käyttöjärjestelmän edut: käsittelyviive on vaiheen 1 järjestelmää pienempi ja etenkin ennalta-arvattavampi.

5.2 Resurssien käyttö

Eri järjestelmien resurssien käyttöä vertailtiin muistin käytön ja energian kulutuksen osalta. Muistin käyttöä vertailtiin data- ja ohjelmamuistien osalta. Energiaku-

Taulukko 5.3: Muistin käyttö.

Järjestelmä	Datamuistin kulutus (max. 3996 tavua)	Ero alkuperäiseen (%)	Ohjelmamuistin kulutus (max. 128 kt)	Ero alkuperäiseen (%)
Alkuperäinen	3041	0	50400	0
Vaiheen 1	3409	12	66261	31
Vaiheen 2 (OS)	3886	28	77298	53

lutusvertailu on toteutettu lähinnä teoreettisessa mielessä.

5.2.1 Muistin käyttö

Taulukossa 5.3 on vertailtu eri järjestelmien muistin käyttöä. Taulukosta nähdään käyttöjärjestelmän lisäävän ohjelma- ja etenkin datamuistin kulutusta huomattavasti. Lisäksi TUTWSN:n aikaisemmassa versiossa on dynaamiselle muistille varattu 1024 tavua, mutta tätä ei käytetä ollenkaan. Muistin käyttöä vertailtaessa tulee huomata tässä työssä toteutettujen versioiden sisältävän enemmän ominaisuuksia kuin aiempi TUTWSN. Ominaisuudet lisäävät muistin käyttöä ja vaikeuttavat vertailua. Kuitenkin käyttöjärjestelmällisen version datamuistin käyttöaste on n. 99 %, mikä on huomattavasti enemmän kuin vaiheen 1 version 85 %. Tämä kertoo lisääntyneiden ominaisuuksien lisäksi käyttöjärjestelmän aiheuttamasta suuremmasta datamuistin käytöstä. Eroa saattaisi olla mahdollista pienentää optimoimalla, mutta käyttöjärjestelmän aiheuttama muistikustannus neljää prosessia käytettäessä olisi kuitenkin helposti lähes kilotavun luokkaa riippuen prosessien vaatiman pinon koosta. Käyttöjärjestelmän dynaamisen muistinhallinnan mahdollistava tehokkaampi datamuistinkäyttö ei esimerkkijärjestelmän muistin kulutuksessa näy. Eri järjestelmällä ja suuremmalla määrällä datamuistia voisi käyttöjärjestelmän hyödyt tulla paremmin esille. Tällöin käyttöjärjestelmän käyttö olisi perustellumpaa.

Ohjelmamuistinkäytössä erot eivät ole aivan yhtä suuria. Ohjelmamuistia on lisäksi huomattavasti datamuistia enemmän järjestelmässä vapaana. Pelkkä SensorOS vaatii 5274 tavua ohjelmamuistia. Lisäksi käyttöjärjestelmän käyttö vaati hieman koodimuutoksia, mikä lisäsi ohjelmamuisti kulutusta. Optimoimalla erot todennäköisesti pienenesivät. Ohjelmamuistin määrässä näkyy selkeästi alkuperäisen TUTWSN:n pienempi ominaisuuksien määränä pienempänä ohjelmamuistin käyttönä.

5.2.2 Energiankulutus

Lyhyen viiveen TUTWSN ei ole erittäin energiatehokas anturiverkko reitittävien anturilaitteiden osalta. Normaalitoiminnassa radio on kokoajan päällä, jolloin anturilaitteen elinikä on vain muutamia viikkoja 2000 mAh:n paristoilla, mikä on hyvin vähän verrattuna esimerkiksi matalan energiakulutuksen TUTWSN:n anturilaitteiden useiden vuosien elinikään. Tästä syystä reitittävien anturilaitteiden ohjelmistoa ei ole optimoitu virrankulutuksen kannalta, sillä esimerkiksi mikrokontrollerin unitilaan siirtymisen aiheuttama säästö olisi eliniän kannalta lähes merkityksetön.

Joka tapauksessa vertailujärjestelmien ohjelmistoja voidaan verrata energiankulutuksen kannalta siinä määrin, miten helposti energiansäästötoimenpiteet olisi toteutettavissa. Käytännöllisesti katsoen järjestelmien energiatehokkuus riippuu siitä, miten paljon mikrokontrolleri on tyhjäkäyntitilassa ja miten helppo tähän tilaan on siirtyä. Lisäksi radion vastaanotto- ja lähetysaikojen minimoiminen on oleellinen osa energiankulutuksen pienentämistä.

Alkuperäisellä anturilaitteen ohjelmistolla tyhjäkäyntitilaan siirrytään aina odotettaessa passiivisesti jotain tapahtumaa. Unitilaan siirtyminen on osana ajastinmoduulin normaalia toimintaa. Unitilassa ei kuitenkaan voida olla äärettömän kauan, sillä ajastimen toiminta sekä aikakriittiset toimenpiteet vaativat jatkuvaa tarkkailua, mitä ei voida suorittaa unitilassa. Käyttöjärjestelmää käytettäessä unitilaan siirtyminen on toteutettu helposti tyhjäkäyntiprosessin avulla.

Käyttöjärjestelmän käyttö pienensi paketin käsittelyviivettä verrattuna vaiheen 1 järjestelmään ja lisäsi radion päälläoloaikaa verrattuna alkuperäiseen järjestelmään. Molemmat toimenpiteet vähentävät turhien lähetysten määrää, mikä parantaa järjestelmän suorituskykyä ja samalla energiatehokkuutta.

5.3 Rakenne

Ohjelmiston rakenteen monimutkaisuutta arvioitiin mutkikkuusanalyysillä ja subjektiivisesti ohjelmakoodia ja ohjelmiston rakennetta tarkastelemalla. Subjektiiviset rakenneanalyysit olisi parempien tulosten saamisten kannalta hyvä tehdä suurella joukolla ihmisiä, mutta sitä ei tämän työn puitteissa pystyttä toteuttamaan.

Ohjelmiston rakenne selkiytyi käyttöjärjestelmän käytön myötä modulaarisuuden lisääntyessä. Eri kerrokset ovat käyttöjärjestelmää käytettäessä selkeästi omina prosesseinaan. Tämä vähentää eri kerroksien riippuvuutta toisistaan ja mahdollistaa helpommin hajautetun ohjelmistokehityksen. Lisäksi vaiheen 1 pollausrakenteiden poistuminen yksinkertaisti järjestelmän ohjelmointia ja rakennetta - reaaliaikaisuusvaatimuksien toteutuminen siirtyi suunnittelijalta käyttöjärjestelmän vastuulle.

Sovelluskerros yksinkertaistui sovellusten säikeiksi kapseloitumisen myötä. Säikeet mahdollistavat lisäksi dynaamisen sovellusten hallinnan. Toisaalta käyttöjär-

jestelmän käyttö muuttaa sovellusten ohjelmointirajapintaa, mikä tekee sovelluksista sopimattomia muiden TUTWSN:n versioiden sovellusten kanssa. Tämä heikentää ohjelmistokehityksen tehokkuutta.

Hallinnointikerroksen integrointi prosessien yhteyteen selkeytti järjestelmän rakennetta, sillä hallinnointitoimenpiteet ovat käyttöjärjestelmää käytettäessä osana toimenpiteisiin liittyvää kerrosta. Hallinnointikerroksen kutsurakenne myös yksinkertaistui muutoksen myötä. Toisaalta muutos olisi mahdollista tehdä ilman käyttöjärjestelmän käyttöönottoakin, joten varsinaiseksi käyttöjärjestelmän eduksi hallinnointikerroksen yksinkertaistamista ei voida täysin laskea.

Käyttämällä suurta määrää prosesseja käyttöjärjestelmän modulaarisuutta olisi mahdollista lisätä ja rakennetta selkeyttää. Prosessien käytössä ongelmana on kuitenkin rajalliset muistiresurssit. Muistiresursseiltaan vähemmän rajoittuneella mikrokontrollerilla olisi todennäköisesti mahdollista saada käyttöjärjestelmän aiheuttamat edut rakenteeseen paremmin selville.

5.3.1 Mutkikkuusanalyysi

Eri järjestelmille suoritettiin mutkikkuusanalyysi käyttäen Thomas McCaben kehittämää mutkikkuusmittaa (engl. cyclomatic complexity) [31], joka on käytetyin tapa ohjelmistojen mutkikkuuden mittaamiseen [15]. Lisäksi mitattiin järjestelmien suorituspolkujen määrä. Mittauksiin käytettiin SciToolsin Understand-ohjelmaa [3].

Mutkikkuusmitta kertoo funktioiden suorituspolkujen määrästä. Mikäli suoritusteitä on vain yksi, on mutkikkuuden arvo yksi. Eri mutkikkuuden tasojen määrittäminen on monimutkaisempaa. Mutkikkuusmitan kehittäjä McCabe, pitää selkeän ohjelman maksimimutkikkuuden arvona kymmentä [31]. Mutkikkuudelle on määritetty kuitenkin seuraavat hyväksi havaitut tasot [15] :

1-10	Yksinkertainen funktio, virheen riski pieni
11-20	Monimutkaisempi funktio, keskinkertainen riski
21-50	Monimutkainen funktio, korkea riski
51-	Epävakaa funktio, todella korkea riski

Mittaustulokset on esitetty taulukossa 5.4, josta nähdään järjestelmien erojen olevan hyvin pieniä. Maksimimutkikkuuden erot johtuvat lähinnä muista, kuin tässä työssä tehdyistä muutoksista. Kuitenkin suorituspolkujen määrän erot vaiheen 1 järjestelmän ja muiden välillä on huomattava. Tämä johtunee vaiheen 1 järjestelmän

Taulukko 5.4: Järjestelmien mutkikkuusvertailu. Pienempi luku tarkoittaa pienempää mutkikkuutta. Mutkikkuutta on vertailtu McCaben mutkikkuusmittaa käyttäen [31].

Järjestelmä	Max. mutkikkuus	Keskimääräinen mutkikkuus	Suorituspolkujen määrä
Aiempi	44	2,7	22821
Vaiheen 1 (ei OS)	30	2,65	39894
Vaiheen 2 (OS)	28	2,71	23852

pollausrakenteiden aiheuttamista muutoksista järjestelmään. Kuitenkaan analyysin perusteella ei voi tehdä suuria johtopäätöksiä järjestelmän rakenteiden eroista.

6. JOHTOPÄÄTÖKSET

Tässä diplomityössä muutettiin TUTWSN-anturiverkon anturilaitteen sulautettu ohjelmisto SensorOS-käyttöjärjestelmän päällä toimivaksi. Työ toteutettiin kahdessa vaiheessa. Ensimmäisessä vaiheessa otettiin käyttöjärjestelmän tarjoamat oheispalvelut käyttöön ja muutettiin radion ajuri keskeytysohjatuksi. Toisessa vaiheessa otettiin käyttöön SensorOS:n hybridimallinen käyttöjärjestelmäydin. Vaiheistetulla käyttöönotolla pyrittiin saamaan käyttöjärjestelmän edut paremmin esille.

Taulukossa 6.1 on vertailtu aiempaa versiota TUTWSN:stä, vaiheessa 1 toteutettua keskeytysohjatun radion sisältävää järjestelmää ja vaiheessa 2 toteutettua SensorOS:n päällä toimivaa järjestelmää. Järjestelmät eivät ole täysin vertailukelpoisia, sillä TUTWSN-anturiverkkoon on lisätty ominaisuuksia järjestelmän muutostyön aikana. Kuitenkin taulukosta voi hyvin perustein nähdä tiettyjä kuhunkin järjestelmäarkkitehtuuriin liittyviä etuja ja heikkouksia. Taulukossa on kerrottu lisäksi, voiko kyseisen ominaisuuden laskea tässä työssä saavutettujen tulosten perusteella käyttöjärjestelmän käytön eduksi tai haitaksi langattomissa anturiverkoissa.

Käyttöjärjestelmä sisältää vertailujärjestelmiä suuremman muistijalanjäljen pääosin prosessien aiheuttaman datamuistikustannuksen takia. Tapahtumapohjaisen ytimen käyttö pienentäisi datamuistin kulutusta, mutta silti päästäisiin tuskin käyttöjärjestelmättömän version muistinkäyttölukemiin. Vaiheessa 1 jouduttiin ohjelmistopinon kokoa kasvattamaan vanhaan TUTWSN:ään nähden, mikä lisäsi muistin käyttöä. TUTWSN:n aiempi versio on sekä data- että ohjelmamuistin kulutukseltaan pienin, mutta toisaalta järjestelmä on hyvin staattinen. Käyttöjärjestelmä mahdollistaa tehokkaan resurssien hallinnan, mutta toisaalta rinnakkaisuus monimutkaistaa ohjelmointia. Käyttöjärjestelmä itsessään tuo yhden virhekomponentin ohjelmistoon lisää, mikä ei paranna järjestelmän luotettavuutta. Toisaalta helpompi ohjelmistokehitys auttaa tuottamaan paremmin toimivia ohjelmistoja.

SensorOS:n sisältämä hybridiydin tuo useita etuja. Se mahdollistaa pienet ja hyvin arvioitavissa olevat keskeytysviiveet ja lisäksi tekee ohjelman suorituksesta aidosti rinnakkaista. Lisäksi ohjelman modulaarisuus lisääntyy, mikä helpottaa ohjelmistokehitystä. Irrottava vuorontaminen mahdollistaa helpon toteutuksen tyhjäkäyntiprosessille, jonka avulla järjestelmän energiankulutus on helppo optimoida järjestelmän niin vaatiessa. Käyttöjärjestelmän etu energiankulutuksen pienentämisessä vertailujärjestelmiin nähden tulee lähinnä toteutustavan helppoudesta ohjel-

Taulukko 6.1: Eri järjestelmiä on vertailtu kappaleessa 2.2 esitettyjen reaaliaikaytimen tarjoamien etujen ja kappaleessa 2.3 esitettyjen langattomien anturiverkkojen käyttöjärjestelmien vaatimuksien perusteella. Numeron yksi saanut järjestelmä on kyseisessä ominaisuudessa paras, kun taas numeron 3 järjestelmä heikoin. Taulukossa kerrotaan myös, onko kyseinen ominaisuus käyttöjärjestelmän aiheuttama etu langattomissa anturiverkoissa

Vaatimus	Vanha	Vaihe 1 (ei OS)	Vaihe 2 (OS)	Käyttöjärjestelmän etu/haitta
Robustisuus	2	1	3	-
Pieni muistijalanjälki	1	2	3	Haitta
Pieni energiankulutus	2	3	1	Etu
Reaaliaikaisuus	3	2	1	Etu
Tehtävien rinnakkaisuus	3	2	1	Etu
Modulaarisuus	3	2	1	Etu
Ohjelmistokehityksen helppous	3	2	1	Etu
Synkronointi	3	2	1	Etu
Resurssien hallinta	3	2	1	Etu

moijan kannalta. Toisaalta käyttöjärjestelmän käyttö itsessään vie prosessoriaikaa ja lisää täten energiankulutusta.

Alun perin työhön lähdettiin lähinnä järjestelmän yksinkertaistamisen vuoksi. Käyttöjärjestelmän havaittiinkin helpottavan ohjelmistokehitystä, joten tavoitteisiin päästiin osin. Lisäksi havaittiin käyttöjärjestelmän käytön tuovan etuja TUTWSN-verkon suorituskykyyn, mikä oli myös tavoitteena.

SensorOS todettiin toimivaksi käyttöjärjestelmäksi anturiverkkoihin ja joitain sen toiminnallisuuteen liittyviä puutteita tuli myös esille. Työ toimi apuna sen kehittämiseen. Lisäksi työ toimii avustavana dokumenttina SensorOS:n käyttöönottoon. SensorOS voi toimia käyttöjärjestelmänä myös anturiverkkojen ulkopuolella resurssirajoitteisissa sulautetuissa järjestelmissä, joten sovellusalue on hyvin laaja.

Jatkossa työtä voisi laajentaa käsittämään muita anturiverkkoja ja käyttöjärjestelmiä. Käyttöjärjestelmän edut vaihtelevat toteutustavan mukaan ja muiden käyttöjärjestelmien käyttö olisi siten perusteltua. Esimerkkijärjestelmänä voisi käyttää lisäksi jotain muuta järjestelmää kuin TUTWSN:ää. Eri anturiverkkojen vaatimukset ja ominaisuudet eroavat toisistaan ja samaten eri esimerkkijärjestelmien käyttö voisi tuoda uusia etuja ja näkökulmia käyttöjärjestelmän käyttöön.

KIRJALLISUUTTA

[1] Contiki on-line documentation. `http://www.sics.se/~adam/contiki/docs/` [ONLINE],viitattu 1.2.2011.

[2] Nanork firefly-alusta. `http://www.nanork.org/wiki/FireFly` [ONLINE], viitattu 1.2.2011.

[3] Scitoolsin understand-ohjelman kotisivut. ohjelmalla voi mitata mm. ohjelmiston mutkikkuutta. `www.scitools.com` [ONLINE], viitattu 1.2.2011.

[4] Tutkija DI Jukka Suhosen kanssa käydyt keskustelut elokuussa 2010.

[5] Ieee standard for information technology- telecommunications and information exchange. *IEEE Std 802.15.1-2002*, 2002.

[6] Tran Nguyen Bao Anh and Su-Lim Tan. Real-time operating systems for small microcontrollers. *IEEE Micro*, 29(5):30–45, 2009.

[7] Stuart R. Ball. Embedded microprocessor systems : Real world design / S.R. ball., 2002.

[8] Michael Barr and Anthony Massa. *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media, Inc., 2006.

[9] R. Barry. *Using the FreeRTOS real time kernel: a practical guide*. Real Time Engineers Ltd., 2009.

[10] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 10:563–579, August 2005.

[11] Wei Dong, Chun Chen, Xue Liu, and Jiajun Bu. Providing os support for wireless sensor networks: Challenges and approaches. *Communications Surveys Tutorials, IEEE*, 12(4):519 –530, 2010.

[12] Cormac Duffy, Utz Roedig, John Herbert, and Cormac Sreenan. An experimental comparison of event driven and multi-threaded sensor node operating systems. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 267 –271, 2007.

[13] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In Andrew T. Campbell, Philippe Bonnet, and John S. Heidemann, editors, *SenSys*, pages 29–42. ACM, 2006.

[14] Jack G. Ganssle. The challenges of real-time programming. `http://embedded-systems.com/98/9807fe.htm` [ONLINE],viitattu 1.2.2011.

[15] Jack G. Ganssle. Taming software complexity. `http://www.eetimes.com/design/embedded/4007519/Taming-software-complexity` [ONLINE],viitattu 1.2.2011.

[16] Jack G. Ganssle. *The art of programming embedded systems.* Academic Press Inc., 1992.

[17] Marko Hännikäinen. *Design of quality of service support for wireless local area networks.* PhD thesis, Tampere University of Technology, 2002.

[18] Hannu-Matti Järvinen Ilkka Haikala. *Käyttöjärjestelmät.* Talentum, 2004.

[19] Institute of Electrical and Electronics Engineers. IEEE standard for local and metropolitan area networks — part 16: Air interface for fixed and mobile broadband wireless access systems. IEEE Std 802.16e-2005, February 2006.

[20] Alec Woo Seth Hollar David Culler Kristofer Pister. Jason Hill, Robert Szewczyk. System architecture directions for network sensors. In *SIGPLANNot., vol. 35, no. 11*, 2000.

[21] Raj Kamal. *Embedded Systems : Architecture, Programming and Design.* McGraw-Hill, 2003.

[22] Ville Kaseva. Rf-based indoor localization for wireless sensor networks. Master's thesis, Tampere University of technology, 2007.

[23] Mikko Kohvakka. *Medium Access Control and Hardware Protoype Designs for Low-Energy Wireless Sensor Networks.* PhD thesis, Tampere University of Technology, 2009.

[24] Mauri Kuorilehto. *System Level Design Issues in Low-Power Wireless Sensor Networks.* PhD thesis, 2008.

[25] T. Laukkarinen, V.A. Kaseva, J. Suhonen, T.D. Hämäläinen, and M. Hännikäinen. Hybridkernel: Preemptive kernel with event-driven extension for resource constrained wireless sensor networks. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 161 –166, 2009.

[26] Teemu Laukkarinen. Operating system kernel for wireless sensor networks. Master's thesis, Tampere University of Technology, 2010.

[27] Donald A. Lewine. *POSIX programmer's guide: writing portable UNIX programs with the POSIX.1 standard.* O'Reilly & Associates, Inc., pub-ORA:adr, 1991. March 1994 printing with corrections, updates, and December 1991 Appendix G.

[28] W. Masri and Z. Mammeri. Middleware for wireless sensor networks: A comparative analysis. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 349 –356, 2007.

[29] Anthony J. Massa. *Embedded Software Development with eCos.* Upper Saddle River, EUA : Prentice-Hall, 2003.

[30] Jukka Suhonen Panu Hämäläinen Marko Hännikainen Timo D. Hämäläinen Mauri Kuorilehto, Mikko Kohvakka. *Ultra-Low Energy Wireless Sensor Networks in Practice stands by itself as an essential guide to a promising-and potentially disruptive technology.* Wiley, feb 2007.

[31] T. J. McCabe. A measure of complexity. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[32] Tommi Mikkonen. Tty:n sulautettu ohjelmointi-kurssin luentomoniste. `http://www.cs.tut.fi/~sulo/luennot/` [ONLINE],viitattu 1.2.2011.

[33] K. M. Znati T Raghavendra, C.S. Sivalingam. *Wireless Sensor Networks.* Springer, 2006.

[34] Antti Riikonen. Low-latency wireless sensor network. Master's thesis, Tampere University of Technology, 2009.

[35] A. Silberschatz, J. Peterson, and P. Galvin. *Operating System Concepts.* Addison Wesley, 1991.

[36] R. Buehrer T. Rappaport, A. Annamalai and W. Tranter. Wireless communications: past events and a future perspective. *CommunicationsMagazine, IEEE, vol. 40, no. 5, pp. 148-161,*, 2002.

[37] Renesas technology corp. General rtos concepts. `http://documentation.renesas.com/eng/products/tool/apn/res05b0008_r8cap.pdf` [ONLINE],viitattu 1.2.2011.