Veli-Mikko Puupponen

Energiatehokkaan mobiilisovellusohjelmoinnin välineitä

Tietotekniikan pro gradu -tutkielma

7. helmikuuta 2017

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Veli-Mikko Puupponen

Yhteystiedot: vesepuup@student.jyu.fi

Ohjaajat: Vesa Lappalainen ja Ari Viinikainen

Työn nimi: Energiatehokkaan mobiilisovellusohjelmoinnin välineitä

Title in English: Tools for energy efficient mobile programming

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 115+60

Tiivistelmä: Älypuhelimista on tullut suosittuja ja ne ovat kehittyneet nopeasti, tarjoten jatkuvasti tehokkaampia suorittimia sekä nopeampia langattomia verkkotekniikoita. Laitteiston kehittyminen on luonut markkinat entistä monipuolisemmille ja kehittyneemmille mobiilisovelluksille, mutta mobiililaitteiden käytettävyys riippuu kokonaan niiden akun mahdollistamasta käyttöajasta. Niinpä sovelluskehittäjien on pystyttävä tarjoaman energiatehokkaita sovelluksia, joissa on silti monimutkaisia toimintoja. Energiatehokkaiden sovellusten kehittäminen vaatii kuitenkin energiatehokkaan ohjelmoinnin käytänteitä ja menetelmiä valittujen ratkaisujen energiankulutuksen arviointiin.

Tässä työssä suoritetaan kattava kirjallisuuskatsaus energiankulutuksen arviointimenetelmistä ja energiatehokkaan mobiiliohjelmoinnin käytänteistä. Tunnistettuja energiankulutuksen arviointimenetelmiä myös vertaillaan ja käydään läpi niihin liittyviä mobiililaitealustan ja ohjelmistopinon aiheuttamia haasteita ja rajoituksia. Työn empiirisessä osassa joitain tunnistetuista ohjelmointikäytänteistä sovelletaan reittimuotoisten paikkatietojen keräämiseen ja lähettämiseen kehitettävään komponenttiin. Lisäksi tässä yhteydessä kehitetään edullinen energiankulutuksen mittalaite, jota käytetään komponentille tehtävissä testeissä.

Avainsanat: Mobiililaite, mobiilisovellus, sovellusohjelmointi, energiankulutus, energiatehokkuus, älypuhelin, energiankulutuksen määrittäminen, energiankulutuksen mittaaminen

Abstract: Smartphones have gained an extensive user base and are constantly evolving to

provide more powerful processor and better connectivity. These improvements have created an increasing demand for more advanced mobile software applications. At the same time, user experience of mobile devices relies on their battery lifetime. As a result, application developers need to be able to provide advanced functionality while keeping the software highly energy efficient. However, developing such applications requires both guidelines for energy efficient programming and methods for assessing the energy consumption of the application code.

In this work, a comprehensive literature review of energy assessment methodologies and actionable practices for mobile application programming will be provided. The properties of the covered assessment methods will also be compared and the limitations imposed by the mobile device hardware and software stack on the methods will be discussed. In the empirical part some of the introduced programming practices are evaluated in the context of a background component for collecting and uploading location trails. In addition, an affordable energy measurement instrument will be designed and built for assessment of the energy consumption of the component.

Keywords: Mobile device, mobile application, application development, energy consumption, energy efficiency, smartphone, assessment of energy consumption, measuring energy consumption

Termiluettelo

A/D-muunnin	Komponentti, joka muuntaa jatkuvan analogisen signaalin sarjaksi digitaalisia lukuarvoja.
AMOLED-näyttö	(Active-Matrix Organic Light-Emitting Diode) on näyttötekniikka, jossa kunkin valoa tuottavan OLED-pikselin yhteydessä on sitä ohjaavia ohutkalvotransistoreja.
ARM	(Advanced RISC Machine) on RISC-suoritinarkkitehtuuriperhe, jota on käytetty laajasti mobiililaitteiden SoC-piireissä.
Android	Linux-ytimeen perustuva, mobiililaitteille suunnattu avoimen lähdekoodin ohjelmistopino.
Arduino	Avoimen lähdekoodin mikro-ohjainalusta elektroniikan kehittämiseen. Perustuu Atmelin AVR-mikro-ohjaimiin.
C-tilat	Suorittimen energiansäästötiloja, jotka sammuttavat sen komponentteja energiankulutuksen vähentämiseksi. C0 on tavallinen käyttötila ja C6 syvin säästötila.
CMRR	(Common-Mode Rejection Ratio) kuvaa järjestelmän kykyä torjua yhteismuotoista häiriötä differentiaalisessa signaalissa.
CPI	(Cycles Per Instruction) kuvaa käskyn suorittamisen kestoa kellojaksoina.
DFVS	(Dynamic Frequency Voltage Scaling) on mahdollistava tekniikka ja energiankulutuksen hallintamenetelmä, jossa suorittimen kellotaajuutta ja jännitettä pyritään laskemaan suorituskykytavoitteiden puitteissa energian säästämiseksi.
GPRS	(General Packet Radio Service) on GSM-verkossa toimiva pakettidatan siirtomenetelmä.
Hall-anturi	Sensori, joka muuttaa vallitsevan magneettikentän voimakkuuden analogiseksi signaaliksi.
HSPA	(High-Speed Packet Access) ja HSPA+ ovat nimiä 3G-verkoissa käytettäville arkkitehtuureille, joilla UMTS-tekniikan pakettidatan siirtonopeutta on kehitetty.

JSON	(Java Script Object Notation) on tiivis, ihmisen luettavissa oleva formaatti attribuutti-arvo-parien kuvaamiseen.
LCD-näyttö	(Liquid Crystal Display) eli nestekidenäyttö on pikseleiden valonläpäisevyyden säätelyyn perustuva näyttötekniikka.
LTE	(Long Term Evaluation) on aiempien ratkaisujen pohjalta kehittynyt standardi, joka kuvaa mobiilipakettidatan nopeita siirtotekniikoita. Yhdistetään usein DC-HSPA:n kanssa termin 4G alle.
OLED-näyttö	(Organic Light-Emitting Diode) on näyttötekniikka, jossa näytön pikselit ovat valoa tuottavia LED-elementtejä.
P-tilat	Suorittimen toiminnallisia tiloja, joissa sen käyttöjännitettä ja kellotaajuutta lasketaan progressiivisesti tilan numeron kasvaessa energian säästämiseksi.
PSRR	(Power Supply Rejection Ratio) kuvaa järjestelmän kykyä torjua virtalähteen tuottamia häiriöitä.
Suorituskykylaskuri	Käyttöjärjestelmien tai laitteistokomponenttien sisältämiä laskureita, jotka kuvaavat eri tapahtumien suoritusmääriä tai resurssien käyttöastetta.
Sähköenergia	Sähköistä potentiaalienergiaa, jota esimerkiksi akku sisältää. Tässä työssä termiä käytetään myös kuvaamaan sitä potentiaalienergiaa, jonka laite on kuluttanut ja muuttanut toisiin muotoihin. Sähköenergia on kuluneen ajan ja sähkötehon tulo.
Sähköteho	Kuvaa laitteen kuluttaman energian määrää aikayksikössä. Teho on virran ja jännitteen tulo ja sen yksikkö on watti.
Sähkövirta	Kuvaa virtapiirissä kulkevan virran voimakkuutta, yksikkö ampeeri.
WLAN	(Wireless Local Access Network) on termi, joka kattaa erilaiset langattoman lähiverkon toteutustekniikat.
WWAN	(Wireless Wide Area Network) on termi, joka kattaa kaikki langattoman laajaverkon toteutustekniikat.

Kuviot

Taulukot

Sisältö

1 Johdanto

Mobiililaitteiden merkitys on kasvanut nopeasti. Älypuhelinten maailmanlaajuinen myyntimäärä on ohittanut sekä tavalliset matkapuhelimet että tietokoneet ja tablettien myyntimäärä vastaa suuruudeltaan 75 %:a tietokoneiden myynnistä ("Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013" 2014; "Worldwide Shipments of Slate Tablets Continue to Decline While Detachable Tablets Climb to New High, According to IDC" 2016; "PC Market Finishes 2015 As Expected, Hopefully Setting the Stage for a More Stable Future, According to IDC" 2016). Samalla mobiililaitteiden laitteisto on kehittynyt merkittävästi, tarjoten muun muassa moniydinprosessoreita ja nopeita langattomia yhteystekniikoita. Kehittyneen laitteistonsa ansiosta mobiililaitteet tukevat jatkuvasti monipuolisempia sovelluksia. Kannettavina laitteina niiden käyttökelpoisuus riippuu kuitenkin akun mahdollistamasta käyttöajasta, eikä akkujen kapasiteetti ole merkittävästi kehittynyt. Niinpä laitteen toiminta-ajan ollessa käyttäjille tärkeä käytettävyyskysymys, on sovelluskehittäjän osattava ottaa työssään huomioon myös energiankulutus (Heikkinen ym. 2012).

1.1 Työn taustaa

Energiankulutuksen huomioiminen ja energiantehokkaiden sovellusten kehittäminen vaatii tietoa hyvistä ohjelmointiratkaisuista, mutta nämä eivät välttämättä yksinään riitä. Ratkaisujen energiankulutukseen vaikuttaa myös niiden tapauskohtainen ympäristö ja tekniikan kehitys on nopeaa, joten tämä ympäristö on jatkuvassa muutoksessa. Näin kehittäjän on usein tarpeen määrittää sovelluksen tai sen osien todellinen energiankulutus tehokkaiden toteutusratkaisujen valitsemiseksi (Ding Li ym. 2013).

Käytännön ohjeet energiatehokkaaseen ohjelmointiin nykyisille mobiililaitealustoille ovat vielä suhteellisen harvinaisia, eikä niiden tehokkuutta ole välttämättä tarkistettu. Tavallisesti kattavimpia tietolähteitä ovat aihealueeseen erikoistuneet internet-keskustelupalstat (Li ja Halfond 2014). Yksi merkittävistä poikkeuksista ovat kuitenkin tietyt laitteistokomponentit, kuten radiorajapinnat, joiden energiankulutusta on tutkittu jo kattavasti ja joiden käytöstä

sovelluksissa on annettu suosituksia (Balasubramanian, Balasubramanian ja Venkataramani 2009; Rice ja Hay 2010; Ou ym. 2013).

Sovellusten energiankulutuksen määrittämiseen on sen sijaan jo kehitetty ratkaisuja yleisimmille mobiililaitealustoille. Osa niistä on esimerkiksi käyttöjärjestelmäkehittäjien tai järjestelmäpiirivalmistajien tarjoamia (esim. “Trepn Profiler” 2016). Lisäksi kirjallisuudessa on julkaistu lukuisia menetelmiä, joista osa perustuu energiankulutusta mittaavaan ulkoiseen mittalaitteeseen (Rice ja Hay 2010) ja osa esimerkiksi järjestelmän tilatietoja käyttävään mallintamiseen (Murmuria ym. 2012). Määritysmenetelmien tarkkuutta, rajoituksia, helppokäyttöisyyttä tai niiden toteutusten saatavuutta ei kuitenkaan ole kattavasti käsitelty sovelluskehittäjän näkökulmasta.

1.2 Tutkimuskysymykset ja tutkimusmenetelmä

Tässä työssä tarkasteltava tutkimusongelma voidaan esittää seuraavina kysymyksinä: *Millaiset mobiilisovellusten ohjelmointikäytänteet ovat energiatehokkaita* ja *Miten sovelluskehittäjä voi määrittää sovellusten energiankulutusta*. Nämä kysymykset valittiin, koska ne liittyvät kiinteästi toisiinsa ja voivat tukea mobiilisovelluskehitystä riippumatta alan muusta kehityksestä. Valittuihin kysymyksiin vastaamiseksi teoriaosuudessa suoritetaan kirjallisuuskatsaus energiatehokkaan mobiiliohjelmoinnin käytänteistä ja sovellusten energiankulutuksen määrittämisen menetelmistä. Lisäksi empiirisessä osassa kehitetään suunnittelutieteellistä tutkimusmenetelmää soveltaen paikkatiedon keräämiseen, tallentamiseen ja lähettämiseen liittyvä komponentti, jossa testataan osaa tunnistetuista käytänteistä. Tässä yhteydessä tutkitaan myös energiankulutuksen mittaamiseen tarvittavien muutosten tekemistä mobiililaitteisiin ja kehitetään edullinen energiankulutuksen mittalaite.

Teoriaosuudessa käsiteltävä tieto on koottu kunkin aihepiirin ajankohtaisista julkaisuista. Näitä kerättiin julkaisutietokannoista, joista tärkeimpiä olivat ACM Digital Library ja IEEE Xplore, mutta myös Elsevier ScienceDirect ja EBSCOHost. Tyypillinen haku alkoi laajana sanahakuna, kuten "mobile application energy"tai "energy measurement smartphone", jota rajattiin esimerkiksi julkaisuajankohdan mukaan. Tärkeimmiksi lähteiksi osoittautuivat tutkimus- ja katsausartikkelit, joita käytiin yhteensä läpi yli kolmesataa. Näistä runsaat kak-

sisataa artikkelia valittiin tarkempaan käsittelyyn ja kunkin sisällöstä koottiin tiivistelmä. Samalla myös niiden lähteet ja viittaukset haettiin tietokannoista uusien julkaisujen tunnistamiseksi. Aihealueen ilmiöistä on käytetty ja käytetään edelleen useita rinnakkaisia termejä, joten tämä lähestymistapa osoittautui merkittäväksi tärkeimpien julkaisujen löytämisessä.

1.3 Työn rakenne

Työn sisältö on jaoteltu seuraavasti: Toisessa luvussa tarkastellaan tietotekniikan sähköenergiankulutusta laajempana ilmiönä ja mobiililaitteiden sijoittumista tähän kokonaisuuteen. Kolmannessa luvussa esitellään tässä työssä käytettävä mobiililaitteen määritelmä. Neljännessä luvussa käsitellään sähköenergiankulutuksen määrittämismenetelmiä ja mobiililaiteen määrittämiselle asettamia rajoitteita. Viidennessä luvussa käydään läpi energiatehokkaan mobiiliohjelmoinnin suositeltuja käytänteitä. Kuudennessa luvussa kuvataan työn empiirisessä, konstruktiivisessa osassa sovellettava tutkimusmenetelmä. Seitsemännessä luvussa esitellään empiirisesti tarkasteltava komponentti ja sen vaatimukset, kehityksessä sovellettava energiankulutuksen mittausjärjestely, testaus- ja kehitysympäristö, sovelletut energiatehokkaat ohjelmointikäytänteet ja saadut tulokset. Kahdeksas luku on keskustelu, jossa kootaan yhteenveto saaduista tuloksista ja esitellään työtä tehtäessä tunnistettuja kysymyksiä ja aiheita mahdolliselle jatkotutkimukselle.

2 Energiankulutus osana tietotekniikkaa

Tietokonelaitteet kuluttavat sähköenergiaa suorittaessaan toimintoja. Ennen 2000-lukua energiankulutukseen liittyviä kysymyksiä tutkittiin vasta vähän korkealla tasolla, laitteistokehityksen ja piirisuunnittelun ulkopuolella. Akkukäyttöisille, kannettaville laitteille ja sulautettuihin järjestelmiin suunnatun koodin energiatehokkuus oli kuitenkin jo nouseva tutkimusala (esim. Tiwari, Malik ja Wolfe 1994; Cignetti, Komarov ja Ellis 2000). 2000-luvulla tietokoneet yleistyivät aiempaa nopeammin ja syntyi uusia internet-palvelutyyppejä. Lisäksi vuosikymmenen loppupuoliskolla julkaistiin entistä monipuolisempia ja tehokkaampia älypuhelimia. Yhdessä nämä muutokset johtivat maailmanlaajuisen tietoliikennemäärän 564-kertaistumiseen vuosien 2000 ja 2014 välillä (“The History and Future of Internet Traffic” 2015). Samalla aikavälillä liikenteen määrä mobiilidataverkoissa kasvoi satoja miljoonia kertoja suuremmaksi (“Major Mobile Milestones - The Last 15 Years, and the Next Five” 2016).

Liikenteen jatkuva lisääntyminen on vaatinut lisää resursseja ja samalla merkittävästi lisännyt energiankulutusta palvelin- ja verkkoinfrastruktuurissa sekä päätelaitteissa. Viime vuosina on arvioitu, että tieto- ja viestintäteknologian osuus maailmanlaajuisesta sähköenergiankulutuksesta olisi jo 5 %:n ja 10 %:n välillä (Andrae ja Edler 2015; Heddeghem ym. 2014). Tällä hetkellä palvelinten, verkkotekniikan ja päätelaitteiden osuudet kulutuksesta ovat lähellä toisiaan, mutta mobiililaitteiden lisääntymisen arvioidaan hitaasti pienentävän päätelaitteiden suhteellista osuutta (Andrae ja Edler 2015). Laitteiden hyötysuhde myös kehittyy jatkuvasti, mutta se ei ole näyttänyt vähentävän kulutusta laitetyypeittäin, vaan mahdollistavan uusia, mobiilimpia ratkaisuja. Nämä ovat johtaneet kasvaviin markkinoihin, joilla uudet tekniikat ovat tulleet vanhojen rinnalle. Niinpä tieto- ja viestintäteknologian sähkönkulutuksen odotetaan edelleen lisääntyvän ja sen suhteellisen osuuden kaikesta kulutuksesta kasvavan entisestään (Heddeghem ym. 2014).

Energiatehokkuuteen tähtäävä tutkimus on jatkuvasti laajenemassa. Nyt selvä motivaattori on sähköstä aiheutuva kustannus, joka voi muun muassa suurille teleoperaattoreille olla varsin merkittävä (Kong ja Liu 2014). Selvä osoitus ilmiön mittakaavasta on esimerkiksi Japanilaisen NTT-yhtymän yhden prosentin osuus koko maan sähkönkulutuksesta (Arai ym. 2014). Samalla kustannusten rinnalla entistä tärkeämmiksi ovat muodostumassa myös kysy-

mykset ilmastovaikutuksista, joiden kannalta pelkkä sähkönkulutus ei ole mittarina riittävä. Sähköenergian tuotantotavat vaihtelevat kulutuspaikan ja ajankohdan mukaan merkittävästi, joten ilmastoon vaikuttavien hiilidioksidipäästöjen vähentäminen vaatii myös spatiaalisten ja temporaalisten kysymysten huomioimista (Kong ja Liu 2014). Käytännössä tämä on tarkoittanut energian varastointia uusiutuvien energianlähteiden toiminta-aikana, energiankäytön suunnittelua uusiutuvan energian ostamiseksi tai tuotantolaitosten lisäämiseksi ja tutkimusta kuorman jakamisesta datakeskusten kesken sekä tehtävien suorituksen viivästämisestä (Kong ja Liu 2014; Pierson 2010; “Renewable energy - Data Centers - Google” 2016; Al-Dulaimy ym. 2016).

Yksi selvistä suorituskykyä ja energiatehokkuutta kehittäneistä tekijöistä kaikissa tietokonelaitteissa on suoritinarkkitehtuureiden ja niitä tukevien piirinvalmistustekniikoiden kehitys. Näin esimerkiksi Intelin prosessoreiden suorituskyky on noin kymmenessä vuodessa lähes kymmenkertaistunut energiankulutuksen pysyessä muuttumattomana (Esmaeilzadeh ym. 2011). Lisäksi tehokkaasta kellotaajuuden ja käyttöjännitteen säädöstä (DFVS ja P-tilat) sekä matalan energian lepotiloista (C-tilat) on tullut tavallinen osa moderneja suorittimia (Esmaeilzadeh ym. 2011; Kambadur ja Kim 2014). Nämä tekniikat ovat välttämättömiä akkukäyttöisissä mobiililaitteissa, joissa laskentatehoa tarvitaan vain lyhyissä jaksoissa ja muuna aikana suoritetaan vaatimattomia taustaprosesseja (Falaki ym. 2010). Lepotilat voivat säästää kymmeniä prosentteja energiaa myös muilla alustoilla (Kambadur ja Kim 2014), mutta usein palvelimissa, joissa mahdollisen vaikutuksen suuruusluokka olisi merkittävä, niitä ei voida soveltaa. Tämän taustalla vaikuttavat vikasietoisuus- ja palvelutasovaatimukset, jotka estävät korkean tavoitekuormituksen. Näin palvelinten tavallinen kuormitustaso on 10–50 %, eikä kokonaan käyttämättömiä varalaitteistoja ole kustannustehokasta asentaa (Barroso ja Holzle 2007; Kistowski ym. 2015).

Palvelinten laskentaan kuluttama sähköenergia ei kuitenkaan kata niiden kokonaisvaikutusta ympäristöön. Vapautuvan hiilidioksidin kautta tarkasteltuna valmistamisen osuus voi olla viidenneksen ja ympäröivän konesali-infrastruktuuri sähköenergia neljänneksen niiden koko elinkaaren vaikutuksesta (Chang ym. 2012). Näitä tekijöitä on pyritty rajoittamaan siirtymällä korttipalvelimiin, jotka vähentävät koteloihin käytettävää materiaalia (Chang ym. 2012; “Google unlocks once-secret server” 2009). Lisäksi menetelmiä on kehitetty muun muas-

sa laitteiden sähkönsyötön ja jäähdytyksen hyötysuhteen parantamiseen (Chang ym. 2012; Gough, Steiner ja Saunders 2015; Shakshuki ym. 2013).

Edellisistä lähestymistavoista kaikki perustuvat tietokoneiden laitteistoon tai ympäröivään infrastruktuuriin. Energiankulutuksen vähentämiseen tähtäävät, ohjelmistotasoiset ratkaisut on laajasti tunnistettu tärkeiksi ja niitä tutkittu jonkin verran, mutta palvelinkeskuksissa sovelletuista hyvistä käytänteistä on todettu olevan vielä vähän dokumentoitua tietoa (Shakshuki ym. 2013). Ohjelmistotasoisten ratkaisujen mahdollista merkitystä korostaa erityisesti se, että laitteiston peruskulutuksen yläpuolella energiankulutus riippuu ohjelmasta, ja tämän vaihtelun osuus on yleensä yli puolet maksimikulutuksesta (Barroso ja Holzle 2007). Ympäröivä infrastruktuuri tuottaa laitteille niiden tarvitseman sähkön ja poistaa niiden tuottaman lämmön, joten ohjelmiston vaikutus vielä vahvistuu näihin toimintoihin käytettävän energian seurauksena (Capra, Francalanci ja Slaughter 2012).

Osa jo aktiivisesti käytettävistä, ohjelmien suorituskykyyn liittyvistä hyvistä käytänteistä ja optimointitavoista vaikuttaa kuitenkin edullisesti myös energiankulutukseen. Tällainen tekijä on esimerkiksi kääntäjäoptimointi, jonka ainakin C/C++:n osalta on todettu yleensä parantavan käännöksen energiatehokkuutta (Kambadur ja Kim 2014; Rahman, Guo ja Yi 2011). Energiankulutuksen ja suorituskyvyn suhde ei kuitenkaan ole vakaa, vaan riippuu ohjelmasta, ja osa mahdollisista optimoinneista tuottaa toisen tai molempien tavoitteiden kannalta selvästi parasta mahdollista heikomman lopputuloksen (Rahman, Guo ja Yi 2011). Muutamia ensisijaisesti energiatehokkuuteen keskittyviä kääntäjäoptimointeja on kehitetty, mutta ne eivät ole levinneet laajaan käyttöön (esim. Hsu ja Kremer 2003). Näin tulevaisuuden tavoitteiksi onkin esitetty menetelmien kehittämisen lisäksi niiden laajempaa käyttöönottoa ja tutkimuksen suuntaamista optimointia tukeviin automaattisiin työkaluihin ja kehitysympäristöihin (Kambadur ja Kim 2014). Monet tämänhetkisistä, energiatehokasta ohjelmointia käsittelevistä suosituksista perustuvat edelleen lähinnä suorituskykyä tehostamaan kehitettyihin menetelmiin (esim. “Energy-Efficient Software Guidelines - Intel Developer Zone” 2011).

Modernit, usein tulkatut moniparadigmakielet ovat muodostuneet tärkeäksi osaksi myös palvelinohjelmistojen kehitystä ilmaisuvoimansa, siirrettävyytensä ja kehitysnopeutensa ansiosta (Esmaeilzadeh ym. 2011). Ohjelmointikielten energiankulutuksesta ei ole välttämättä teh-

ty kattavia vertailuja, mutta tulkatut ja virtuaalikoneilla suoritettavat kielet ovat osoittautuneet tavallisesti energiatehokkuudeltaan käännettyjä heikommiksi (mm. Noureddine ym. 2012; Chatzigeorgiou ja Stephanides 2002). Lisäksi niihin liittyvät käytänteet, kuten runsas kirjastojen käyttö ja syvät hierarkkiset rakenteet, voivat entisestään heikentää energiatehokkuutta (Bhattacharya ym. 2012; Capra, Francalanci ja Slaughter 2012). Viime aikoina joidenkin perinteisesti tulkkitoteutuksiin perustuneiden kielten tilanne on kuitenkin muuttunut, lähinnä suorituskykyvaatimusten motivoimana, kun niistä on julkaistu myös kääntäjätoteutuksia. Tähän joukkoon kuuluvat jo muun muassa suositut PHP ja ECMAScript ("HHVM" 2016; "Chrome V8 | Google Developers" 2016).

Mobiililaitteet ovat palvelimiin verrattuna lähellä toista ääripäätä sähköenergiaan liittyvissä kysymyksissä. Niiden käyttämä teho on matala myös piikkikuormituksessa ja jäähdytyksen sekä akun kapasiteetin rajoitteiden takia tätä tasoa ei voida ylläpitää pitkäaikaisesti. Niinpä kokonaisvaikutus maailmanlaajuiseen sähköenergiankulutukseen on arvioitu suuresta laitteiden lukumäärästä huolimatta suhteellisen matalaksi (Andrae ja Edler 2015). Käytettävyyden kannalta energiatehokkuus on kuitenkin merkittävä kysymys. Lisäksi mobiililaitteiden virtalähteinä käytetyillä litium-akuilla on rajallinen käyttöikä, jota rajoittaa ensisijaisesti niiden lataus-purku-jaksojen lukumäärä (Eom ym. 2007). Energiankulutuksen vähentäminen laskee suoraan tarvittavaa jaksomäärää ja voi näin pidentää akkujen käyttöikää. Hiilidioksidiekvivalentin avulla tarkasteltuna mobiililaitteiden kokonaisvaikutuksesta valtaosa aiheutuu valmistuksesta ja raaka-aineiden tuotannosta, kun taas käytönaikainen sähköenergiankulutus jää alle viidennekseen (Moberg ym. 2014; Teehan ja Kandlikar 2013). Tämä tekee energiankulutuksen vähentämisen mahdollistamasta, käyttöikää pidentävästä kerrannaisvaikutuksesta ympäristön kannalta varsin merkittävän. Viime aikoina kypsillä markkinoilla tapahtunut mobiililaitteiden vaihtovälin pidentyminen myös tukee tämän tekijän mahdollista arvoa ("2014 US Mobile Phone sales fall by 15% and handset replacement cycle lengthens to historic high" 2015; "Gartner Says Worldwide Smartphone Sales to Slow in 2016" 2016).

Palvelinten ja mobiililaitteiden ohella myös muilla päätelaitteilla ja verkkoinfrastruktuurilla on merkittävä vaikutus sähköenergian kokonaiskulutukseen (Heddeghem ym. 2014). Näistä erityisesti mobiilidataverkkoja kehitetään aktiivisesti ja uudessa, viidennen sukupolven tekniikassa energiankulutuksen vähentäminen on yksi keskeisistä tavoitteista (esim. Olsson ym.

2013). Päätelaitteista pöytätietokoneiden ja kannettavien tietokoneiden energiankulutuksen vähentämiseen tähtäävä tutkimus ei vaikuta, laitteiston kehitystä lukuun ottamatta, olevan yhtä aktiivista kuin muilla sektoreilla. Tähän vaikuttaa luultavasti tulosten näkyvyys, joka palvelinten tapauksessa voi tarkoittaa merkittäviä taloudellisia säästöjä ja mobiililaitteissa suoraan havaittavaa akun tarjoaman käyttöajan pitenemistä. Muiden päätelaitteiden kohdalla vaikutusten mittarit eivät ole yhtä näkyviä ja jonkin verran näyttöä on myös siitä, etteivät käyttäjät kiinnitä huomiota energiankulutukseen tai ole siitä kiinnostuneita. Tämän taustalla voi vaikuttaa tietämättömyys ja asennoituminen, jossa tietokoneen kuluttamaa energiaa pidetään merkityksettömänä kustannuseränä. Tässä tilanteessa edes valmiiksi tarjottuja energiansäästöominaisuuksia ei oteta käyttöön ja niitä jopa poistetaan käytöstä, jos ne kuluttavat aikaa tai vaikuttavat muuten negatiivisesti käyttökokemukseen (Chetty ym. 2009).

Vaikka mobiililaitteiden vaikutuksen osuutta tieto- ja viestintäteknologian kokonaisenergiankulutuksesta voidaan pitää suhteessa pienenä, laitemarkkinoiden koko, käytettävyys ja mahdollinen käyttöikää pidentävä kerrannaisvaikutus ovat tärkeitä kysymyksiä. Lisäksi alustariippumattomien, universaalien menetelmien kehittäminen voi olla haasteellista ja rajoittaa mobiililaitteille erityisten piirteiden käsittelyä. Näin mobiililaitealustalle keskittyvää tutkimusta voidaan pitää jo pelkästään sen suorien tulosten ansiosta tärkeänä, mutta on mahdollista, että osa tuloksista on jatkossa sovellettavissa myös muilla alustoilla.

3 Mobiililaitteet tietokonelaitteiden kentässä

Mobiililaitteesta vaikuttaa tulleen yleisesti tunnettu termi älypuhelinten ja tablettien yleistyessä. Jo ensimmäisten modernien kämmenmikrojen tullessa markkinoille mobiililaitteen olennaisena piirteenä pidettiin pienen koon lisäksi langatonta verkkoyhteyttä, joka vapauttaa käyttäjän kiinteästä verkkoliittymästä vastaanottamaan, lähettämään ja käsittelemään tietoa missä ja milloin vain (Imielinski ja Badrinath 1994).

Edellinen määritelmä on kuitenkin vielä varsin laaja ja kattaa esimerkiksi ne pienikokoiset kannettavat mikrotietokoneet, joihin on asennettu jotain langatonta LAN- tai WAN-tekniikkaa tukeva verkkokortti. Toisaalta myös mobiililaitteiksi yleisesti mielletyt älypuhelimet ja tabletit ovat arkkitehtuuriltaan juuri tällaisia mikrotietokoneita.

NIST tarkentaa edellisiä mobiililaitteen ominaisuuksia lisäämällä niihin vaatimuksen kiinteästi asennetusta massamuistista. Lisäksi mobiililaitteeseen on oltava saatavilla sovelluksia useammasta lähteestä, joihin kuuluu laitteen mukana toimitettujen sovellusten lisäksi mahdollisuus asentaa erityisesti kolmannen osapuolen tarjoamia sovelluksia. Ehkä tärkein rajaava tekijä on kuitenkin mobiililaitteen käyttöjärjestelmä, joka on ensisijaisesti kehitetty käytettäväksi mobiililaitteissa, eikä ole täysipainoinen pöytätietokoneille tai kannettaville tietokoneille tarkoitettu käyttöjärjestelmä (Souppaya ja Scarfone 2013).

Raja mobiililaitteiden ja perinteisten tietokoneiden käyttöjärjestelmissä on kuitenkin keinotekoinen. Muutamat valmistajat tarjoavatkin jo kevyitä, ominaisuuksiltaan ja rakenteeltaan kannettavaksi tietokoneeksi luokiteltavia laitteita, joihin on esiasennettu esimerkiksi mobiililaitteissa yleinen Linux-ydintä käyttävä Android-käyttöjärjestelmä (“Lenovo A10 Multimode Laptop” 2015). Vastaavasti Android-pohjaisiin mobiililaitteisiin on usein mahdollista asentaa muita Linux-jakelupaketteja (“Convert an Android Device to Linux” 2015).

Lievästä keinotekoisuudestaan huolimatta tässä työssä mobiililaitteina pidetään NIST:in määritelmän täyttäviä laitteita. Näihin kuuluvat esimerkiksi Android-, iOS-, ja Windows Phone -käyttöjärjestelmiä suorittavat tabletit ja älypuhelimet.

4 Mobiilisovellusten energiankulutuksen määrittäminen

Mobiililaitteet saavat energiansa kapasiteetiltaan rajallisista akuista ja niiden käyttöaika riippuu siitä tehosta, jolla ne kuluttavat akusta energiaa. Energiankulutus koostuu laitteiston ja sillä suoritettavan ohjelmiston yhteisvaikutuksesta, mutta mobiililaitteiden laitteistokomponentit tai ohjelmistopino eivät ole tavallisesti käyttäjän vaihdettavissa. Näin laitteen energiankulutukseen on mahdollista vaikuttaa ensisijaisesti siinä suoritettavien sovellusten kautta.

Mobiilisovellusmarkkinat ovat kasvaneet nopeasti ja tarjolla oleva sovellusmäärä on erittäin suuri, joten samat perustoiminnot tarjoavia sovelluksia on yleensä useampia. Niiden laadullisissa piirteissä, kuten energiankulutuksessa, voi olla samoja toimintoja suoritettaessa merkittäviä sovelluskohtaisia eroja (Claas Wilke ym. 2013). Mobiilisovelluskaupat eivät kuitenkaan vielä tavallisesti testaa sovelluksia niiden energiankulutuksen osalta, jolloin käyttäjän on kokeiltava niitä energiankulutusta arvioidakseen (Claas Wilke ym. 2013). Käyttäjän tarpeettoman korkeaksi kokeman energiankulutuksen on puolestaan todettu laskevan sovellukselle annettavaa arvosanaa (C. Wilke ym. 2013). Näin energiankulutuksen huomioimisella sovelluskehityksessä on koetun laadun kautta usein suora vaikutus sovelluksen levikkiin ja näin myös taloudellista merkitystä sen kehittäjälle.

Sovelluskehittäjä voi käyttää energiankulutuksen määrittämisen tarjoamaa tietoa koko sovelluksen elinkaaren ajan. Suunnitteluvaiheessa tietoa energiankulutuksesta voidaan käyttää korkean tason arkkitehtuuriin liittyviä valintoja tehtäessä. Automatisoituun koodigenerointiin yhdistettynä energiankulutuksen määrittäminen mahdollistaa valitun arkkitehtuurin energiatehokkuuden tarkastelemisen käytettävällä alustalla ennen varsinaisen toteuttamisen aloittamista. Toteutusprosessin edetessä generoitua koodia voidaan korvata asteittain todellisella toteutuksella ja seurata energiankulutuksen kehittymistä (Thompson ym. 2011). Vastaavasti käytettäviä toteutusmalleja valittaessa niiden vaikutusta on mahdollista analysoida kohdealustalla jo ennen lopullista toteutusta (Sahin ym. 2012). Energiankulutuksen määrittäminen mahdollistaa myös yksityiskohtaisempien toteutusratkaisujen vertailemisen esimerkiksi verkkoliikenteen yhteystekniikan valinnan ja käytettävän tietoliikenneprotokollan osalta (Vergara, Nadjm-Tehrani ja Prihodko 2014). Yhtä lailla voidaan vaihtoehtoisia luok-

kakirjastoja vertailtaessa ottaa huomioon niiden energiankulutus kehitettävän sovelluksen todellisissa käyttötapauksissa (Manotas, Pollock ja Clause 2014).

Sovelluksen siirtyessä julkaisua seuraavaan ylläpitovaiheeseen on käyttäjiltä kerätty karkeakin energiankulutustieto käyttökelpoista. Sitä voidaan käyttää tunnistettaessa kontekstisidonnaisia ongelmia, joita ei välttämättä vielä kehitys- tai testausvaiheessa ole ollut mahdollista tunnistaa. Tällaisia ovat esimerkiksi vain tietyillä käyttöjärjestelmä- tai mobiililaiteversiolla esiintyvät ongelmat tai verkkoyhteyden laadun vaikutukset (Oliner ym. 2013).

Kehitys- ja ylläpitoprosessin etenemisestä on myös mahdollista johtaa tietoa energiankulutuksen määrittämisen avulla. Regressiotyyppistä energiankulutuksen testausta jokaiselle sovellusversiolle suorittamalla voidaan arvioida tehtyjen muutosten onnistumista myös energiankulutuksen näkökulmasta. Testauksen suorittaminen on jopa mahdollista liittää versiohallintajärjestelmän automaattiseksi osaksi (Hindle ym. 2014).

Käyttäjälle energiankulutuksen määrittäminen antaa yksinkertaisimmillaan mahdollisuuden verrata sovelluksen kokonaisenergiankulutusta muihin vastaaviin sovelluksiin. Tämä ominaisuus on jo sisäänrakennettuna useimmissa älypuhelinten käyttöjärjestelmissä helpottamassa energiankulutuksen huomioimista sovellusvalinnoissa (esim. “Apple - Batteries - Maximizing Performance” 2015). Kerätyn energiankulutustiedon pohjalta voidaan myös antaa käyttäjälle energiansäästösuosituksia tai automaattisesti hallita mobiililaitteen laitteistokomponentteja ja sovelluksia energiankulutuksen vähentämiseksi (Oliner ym. 2013; Datta, Bonnet ja Nikaein 2012).

4.1 Energiankulutuksen määrittämismenetelmät

Mobiililaitteiden ja niillä suoritettavien sovellusten energiankulutuksen määrittämiseen on useampia menetelmiä. Korkealla tasolla ne voidaan jaotella energiankulutusta mittaaviin ja sitä mallien avulla arvioiviin menetelmiin. Näistä kulutettua sähköenergiaa mittaavat menetelmät ovat suoraviivaisimpia ja tarjoavat tietoa energiasta, jonka laite kuluttaa akusta tai virtalähteestä. Näin ne ovat olleet suosittuja sellaisenaan, mutta erityisesti vertailukohtana muita menetelmiä kehitettäessä. Energiankulutuksen mittaamiseen on kuitenkin usein käytetty mobiililaitteeseen liitettäviä ulkoisia mittalaitteita, jotka rajoittavat menetelmän käyt-

tömahdollisuuksia. Niinpä lähestymistapaa on jatkokehitetty erilaisissa epäsuorissa mittausmenetelmissä, joita voidaan soveltaa mobiililaitteen laitteistolla, ilman erillisiä mittalaitteita (mm. Xu ym. 2013; Zhang ym. 2010; Xu ym. 2013).

Mittaamalla voidaan saada tarkkuudeltaan edustavinta tietoa energiankulutuksesta, mutta muun muassa mittausjärjestelyt ja tulosten käsitteleminen ohjelmistokehitystä tukevalle abstraktiotasolle asettavat haasteita. Tämä on motivoinut kehittämään myös energiankulutusmalleihin perustuvia arviointimenetelmiä. Niiden osalta kattavimmin on tutkittu vaihtoehtoa, jossa arvio energiankulutuksesta muodostetaan mobiililaitteen laitteistokomponenttien tiloja kuvaavien muuttujien pohjalta. Samaa ajatusta on sovellettu myös mobiililaitteiden ulkopuolella, kehitysympäristössä toimivan emulaattorin simuloitujen laitteistokomponenttien tilatietojen käsittelyyn (Mittal, Kansal ja Chandra 2012; Tu ym. 2014). Lisäksi on tutkittu mahdollisuutta soveltaa malleja energiankulutuksen arviointiin ilman laitteistoa, sovellusten lähdekoodin tai valmiiden asennuspakettien pohjalta (esim. Tu ym. 2014).

Menetelmiin liittyy erilaisia reunaehtoja ja niistä on mahdollista saada energiankulutustietoa erilaisilla abstraktio- ja tarkkuustasoilla. Yksityiskohtaisemmin niitä on kuvattu seuraavissa alaluvuissa, joista viimeiseen on koottu vertaileva yhteenveto menetelmistä.

4.1.1 Tehonmittausmenetelmät

Laitteen tai komponentin käyttämä hetkellinen teho, $P(t) = U(t) * I(t)$, on mahdollista määrittää, jos sen kuluttama virta ja käyttöjännite tunnetaan (Patrick ja Fardo 2008). Määrätyllä aikavälillä kulunut sähköenergia puolestaan voidaan laskea tehon integraalina, tai käytännössä säännöllisten, diskreetillä aikavälillä Δt otettujen tehonäytteiden summana, $E_T = \int_0^T P(t)dt \simeq \sum_n P_n * \Delta t$ (Xu ym. 2013). Tehon määrittämiseen tarvittava mobiililaitteen käyttöjännite voidaan tavallisesti mitata suoraviivaisesti A/D-muuntimella. Virran mittaamisessa on puolestaan mahdollista hyödyntää sähköä kuljettavan johtimen ympärille muodostuvaa magneettikenttää (Patrick ja Fardo 2008). Tämä kenttä voidaan muuntaa hall-anturilla jännitteeksi ja näytteistää digitaaliseksi edelleen A/D-muuntimella. Mittaukseen tarvittava johdin voi olla valmiina tai se voidaan tarvittaessa suunnitella laitteen painopiirilevyyn, jolloin virtapiiriin ei tarvita matkalle ylimääräisiä komponentteja (Manousakis ja Nikolopoulos 2012).

Kirjallisuudessa esitetyissä toteutuksissa selvästi suosituin menetelmän mobiililaitteiden virran mittaamiseen on kuitenkin ollut virtapiiriin sijoitettavan mittausvastuksen käyttäminen (esim. Brouwers, Zuniga ja Langendoen 2014). Ohmin lain, $U = I * R$, mukaan vastuksen yli vallitseva jännite on suoraan verrannollinen sen resistanssiin ja sen läpi kulkevaan virtaan (Patrick ja Fardo 2008). Näin virtalähteen ja energiaa kuluttavan laitteen tai komponentin välisessä virtapiirissä olevan, tunnetun resistanssin omaavan vastuksen yli vallitsevan jännitteen näytteistäminen A/D-muuntimella mahdollistaa virran mittaamisen.

Virranmittausmenetelmiä voidaan käyttää sekä laitteen kokonaistehon että laitteistokomponenttien erillisen tehonkulutuksen mittaamisessa. Koska mobiililaitteiden ensisijaisena virtalähteenä toimii yleensä akku, riittää laitteen käyttämän kokonaistehon määrittämiseen akun jännitteen ja siitä käytettävän virran mittaaminen. Korvattaessa akku ulkoisella vakiojännitelähteellä riittää tehon määrittämiseen myös pelkän virran mittaaminen.

Virranmittaukseen käytettävät komponentit lisäävät kuitenkin valmistuskustannuksia. Näin niitä ei laitteistokomponenttikohtaisesti ole valmiina kuin osassa tutkimus- ja kehityskäyttöön suunnatuista mobiililaitteista (McCullough ym. 2011). Monien mobiililaitteiden akkuun liittyvässä virtapiirissä on kuitenkin A/D-muunnin sen jännitteen mittaamista varten. Lisäksi osassa laitteista on myös akusta käytettävän kokonaisvirran mittaamiseen tarvittava laitteisto. Sovellusohjelmointiin käytettävien rajapintojen kautta ne kuitenkin tarjoavat usein alle kymmenen näytettä sekunnissa (Maker, Amirtharajah ja Akella 2013). Tämä nopeus ei välttämättä kaikissa käyttötapauksissa mahdollista tarkkaa arviointia, koska monien mobiililaitteiden energiankulutusspektrissä on pieni osuus piirteitä jopa yli 1 kHz:n taajuudella (Höpfner, Schirmer ja Bunse 2012; Maker, Amirtharajah ja Akella 2013; K. Kim ym. 2014).

Integroidun mittauslaitteiston puuttuessa, suurempaa näytteistysnopeutta tarvittaessa tai useiden komponenttien erillistä mittausta tavoiteltaessa on turvauduttava ulkoisiin mittalaitteisiin. Pelkkää laitteen akusta käyttämää kokonaisenergiaa mitattaessa mobiililaitteeseen tarvittavat muutokset ulkoisen mittalaitteen käyttöönottoa varten voivat olla pieniä. Parhaimmillaan käytettävä mitta-adapteri, esimerkiksi virtamittausvastus, voidaan lisätä akun ja laitteen väliin erillisenä osana (Brouwers, Zuniga ja Langendoen 2014). Jos laitteen akkua ei ole suunniteltu käyttäjän vaihdettavaksi, tarvittavat muutokset ovat monimutkaisempia ja usein pysyvämpiä (Oliner ym. 2013).

Toisin kuin kokonaisenergiankulutuksen mittaamisesta, on kaupallisten mobiililaitteiden erillisten laitteistokomponenttien energiankulutuksen mittaamisesta kirjallisuudessa vasta harvoja esimerkkejä. Yksittäisten laitteistokomponenttien mittaamisen ongelmana on tavallisesti pidetty sitä, ettei kaupallisten laitteiden kytkentäkaavioita ole julkisesti saatavilla (Carroll ja Heiser 2010). Näin mittaukseen tarvittavien muutosten tunnistaminen voi vaatia merkittävää takaisinmallinnusta ja myös itse muutokset voivat olla monimutkaisia.

Käyttökelpoiseksi lähestymistavaksi moderneissakin älypuhelimissa on kuitenkin osoittautunut laitteistokomponenttien käyttöjännitteitä tuottavien jänniteregulaattoreiden tunnistaminen ja niistä käytettävän virran mittaaminen. Osa komponenteista käyttää kuitenkin useampaa jänniteregulaattoria ja osa regulaattoreista on puolestaan jaettu useamman komponentin kesken. Näin yksittäisen komponentin energiankulutuksen mittaaminen voi vaatia useampia yhtäaikaisia mittauksia. Kaikkia komponentteja ei kuitenkaan välttämättä ole mahdollista erottaa suoralla mittauksella lainkaan, koska yksittäinen fyysinen komponentti voi sisältää useita loogisia komponentteja (Carroll ja Heiser 2013).

Matalallakin näytteistysnopeudella kerätty tieto pelkästä laitteen kokonaisenergiankulutuksesta on todettu käyttökelpoiseksi muun muassa toteutusratkaisuja vertailevaan käyttöön (Höpfner, Schirmer ja Bunse 2012). Edes komponenttitasoinen tieto energiankulutuksesta ei kuitenkaan kerro energiankulutuksen aiheuttajasta, jos järjestelmän laitteistokomponenttien tila ei ole täysin kontrolloitu (Brouwers, Zuniga ja Langendoen 2014).

Kerätyn energiankulutusinformaation analyysista voidaan johtaa monipuolisempaa ja usein kehitystyön kannalta käyttökelpoisempaa tietoa tallentamalla samalla tietoa myös laitteen ja sillä suoritettavien ohjelmien tilasta. Tätä voidaan saada laitteen ulkopuolelta esimerkiksi sen tuottamaa verkkoliikennettä tallentamalla (Rice ja Hay 2010). Sovelluksiin voidaan myös lisätä kutsuja, jotka tuottavat paikallisesti tai laitteen ulkopuolelle tallennettavia lokimerkintöjä (Ding Li ym. 2013). Lisäksi käyttäjätasoisille sovelluksille avoimista tapahtumista ja muuttujista saadaan paljon tilatietoa (Brouwers, Zuniga ja Langendoen 2014). Laitteen käyttöjärjestelmän rajoituksista riippuen sen ytimen suorituskykylaskureista ja tapahtumista on puolestaan mahdollista saada hyvinkin yksityiskohtaista tietoa yksittäisten laitteistokomponenttien ja sovellusten tilasta (Brouwers, Zuniga ja Langendoen 2014).

Tilatietojen avulla on mahdollista tunnistaa muun muassa mitatun energiankulutuksen aiheuttaneet sovellukset ja laitteistokomponentit sekä niiden suhteellinen vaikutus kulutukseen (Brouwers, Zuniga ja Langendoen 2014). Energiankulutuksen mittaamisen ja sovelluksen suorituspolkujen seurannan yhdistävällä menetelmällä on osoitettu olevan mahdollista osittaa energiankulutusta jopa yksittäisten aliohjelmien, järjestelmäkutsujen ja lähdekoodirivien tasolla (Chung, Lin ja King 2011; Ding Li ym. 2013).

Riippumatta siitä, mitataanko energiankulutusta komponenttitasoisesti vai laitteen kokonaiskulutuksena, on energiankulutusnäytteet ja tapahtumia kuvaavat tilatiedot voitava synkronoida toisiinsa. Mobiililaitteeseen sisäänrakennettua mittauslaitteistoa käytettäessä energiankulutustieto ja siihen liittyvät tapahtumat ovat valmiiksi synkronisia (Dong ja Zhong 2011). Ulkoista mittalaitetta käytettäessä synkronointi on mahdollista toteuttaa sekä mitattujen energiankulutusnäytteiden avulla että näytekanavan ulkopuolella.

Energiankulutusnäytteiden avulla suoritettavassa synkronoinnissa energiankulutukseen tuotetaan jollain laitteistokomponentilla tunnistettava muutos. Käyttökelpoiseksi komponentiksi on todettu erityisesti näyttö, koska sen energiankulutus riippuu vahvasti sen kirkkaudesta, joka on tavallisesti nopeasti muutettavissa (Rice ja Hay 2010). Yksittäinen pulssi ei kuitenkaan välttämättä erotu vaihtelevasta kulutuksesta, joten merkintään käytetään tunnistettavaa kuviota, esimerkiksi Gold-koodia (Rice ja Hay 2010).

Energianmittauskanavan ulkopuolella tapahtuva synkronointi voi perustua kellojen aktiiviseen synkronointiin tapahtumia tallentavan järjestelmän, usein mobiililaitteen, ja energiankulutusnäytteitä keräävän alustan välillä. NTP-protokollalla synkronoinnissa on mahdollista saavuttaa joidenkin millisekuntien tarkkuus (Wilke, Götz ja Richly 2013). Toinen käyttökelpoinen vaihtoehto on mobiililaitteen I/O-liitäntöjen tai niihin kytkettyjen laitteiden käyttäminen digitaalisen synkronointisignaalin tuottamiseen. Tällöin energiankulutusmittauksia keräävässä alustassa täytyy kuitenkin olla tarjolla vastaava digitaalinen tulo. Yhdeksi mahdolliseksi laitteistokomponentiksi, jonka tilatietoa voidaan käyttää sekä laitteiston että ohjelmointirajapintojen tasolla kohtalaisen helposti, on todettu värinämoottoriin liittyvä digitaalinen lähtö (Brouwers, Zuniga ja Langendoen 2014). Osassa mobiililaitteista on myös huomiovaloja, joita voidaan käyttää mittalaitteeseen kytketyn valosensorin avulla tekemättä mobiililaitteeseen mitään muutoksia.

4.1.2 Akun tilaa hyödyntävät menetelmät

Mobiililaitteen käyttämän sähkötehon mittaaminen vaatii akun jännitettä ja virtaa mittaavan laitteiston. Kaikki mobiililaitteet eivät vielä sisällä tarvittavaa laitteistoa tai julkaise tätä tietoa sovellusohjelmoinnin rajapinnan kautta (Xu ym. 2013). Käytännössä kaikki mobiililaitteet tarjoavat kuitenkin jo ohjelmointirajapinnan kautta arvion akun suhteellisesta varaustilasta (C. Wang ym. 2013) ja osa tarjoaa myös arvion akun absoluuttisesta varaustilasta (Maker, Amirtharajah ja Akella 2013). Laitteet voivat joko laskea akun varaustilan siihen ladatun ja siitä kulutetun energian mittaamisen avulla, mutta ne voivat myös käyttää jotain erilaisista akun ominaisuuksiin perustuvista malleista varaustilan arviointiin (Waag, Fleischer ja Sauer 2014). Mobiililaitteiden modernit akunhallintapiirit käyttävät usein mallintamisen ja energianlaskennan yhdistelmää hyvän arviointitarkkuuden ylläpitämiseksi (esim. “MAX17050 ModelGauge m3 Fuel Gauge - Maxim” 2015).

Absoluuttisen varaustilatiedon tarjoavissa laitteissa energiankulutuksen määrittäminen on mahdollista alustan tarjoamalla resoluutiolla ja päivitysnopeudella tarkasteltavan aikavälin varaustilan alku- ja loppuarvioiden arvojen, E_{t_1} ja E_{t_2}, erotuksena $E = E_{t_2} - E_{t_1}$. Ainoastaan suhteellisen varaustilan tarjoavilla laitteilla voidaan vastaavasti laskea suhteellinen energiankulutus. Suhteellisten arvioiden erotus on käyttökelpoinen samalla laitteella tapahtuvien, peräkkäisten mittausten vertailemiseen. Akkujen kapasiteetin vaihtelun takia tällaiset tulokset eivät kuitenkaan ole välttämättä vertailukelpoisia edes toisilla, vastaavilla laiteyksilöillä tai eri akkua käytettäessä tehtyjen mittausten tuloksiin (Zhang ym. 2010).

Suhteelliset mittaustulokset on mahdollista muuttaa absoluuttisiksi, jos akun käyttökelpoinen kapasiteetti tunnetaan. Akun kapasiteetin voidaan määrittäminen laitteella pelkkää suhteellista varaustilatietoa käyttäen, jos jonkin laitteistokomponentin todellinen tehonkulutus jossain toimintatilassa tunnetaan tarkasti (Zhang ym. 2010). Ulkoisella mittalaitteella kapasiteetti on mahdollista mitata suoraan, mutta tätäkin tulosta on tarkistettava ajoittain muun muassa akun ikääntymisen tai käyttölämpötilan muuttuessa (Waag, Fleischer ja Sauer 2014). Osassa puhelimista on myös käytössä jo akunhallintapiiri, joka osaa arvioida akun todellisen kapasiteetin (“Power Profiles for Android” 2015; “MAX17050 ModelGauge m3 Fuel Gauge - Maxim” 2015; “DS2784 1-Cell Stand-Alone Fuel Gauge IC with Li+ Protector and SHA-1 Authentication - Maxim” 2015). Tyypillisesti nämä akunhallintapiirit kuitenkin tarjoavat

samalla myös suoraan absoluuttisen varaustilatiedon.

Merkittävimpänä ongelmana suhteellisen varaustilan arvon käyttämisessä energiankulutuksen määrittämismenetelmänä on kuitenkin pidetty sen resoluutiota, joka on osassa laitteista vain yksi prosentti (C. Wang ym. 2013). Modernien älypuhelinten keskimääräisen akun kapasiteetista yksi prosentti on kohtuullisen suuri määrä energiaa. Näin useimpia sovelluksia on suoritettava normaalin käyttötapaukseen verrattuna merkittävästi pidempään, tai suhteellisen varaustilan arvossa ei voida havaita lainkaan muutosta (C. Wang ym. 2013).

Yhtenä resoluutioltaan tarkempana vaihtoehtona suhteellisen varaustilan arvolle on esitetty akun jännitteen arvon käyttämistä, koska se on saatavilla useimpien mobiililaitealustojen ohjelmointirajapinnan kautta hyvällä tarkkuudella (mm. Xu ym. 2013; Kapetanakis ja Panagiotakis 2012; Zhang ym. 2010). Mobiililaitteissa yleisten litium-akkujen jännite myös riippuu niiden varaustilasta (Waag, Fleischer ja Sauer 2014). Näin akun jännitteenmuutosta seuraamalla voidaan välillisesti havaita akusta käytettävän energian määrä (Zhang ym. 2010).

Litium-akun jännite ja varaustila eivät kuitenkaan riipu toisistaan lineaarisesti ja niiden suhteeseen vaikuttavat merkittävästi muun muassa lämpötila, akun rakenne, ikä ja kunto (Waag, Fleischer ja Sauer 2014). Ratkaisuna ongelmaan on esitetty laitteen akun jännitearvojen keräämistä mobiililaitteen ilmoittaman varaustilan funktiona akkua purettaessa. Näistä mittaustuloksista voidaan muodostaa akulle purkautumiskäyrä, jonka avulla sen suhteellinen varaustila on mahdollista määrittää sen vallitsevasta jännitteestä. Jos myös akun kapasiteetti E_{kok} tunnetaan, saadaan jännitemittausten välisenä aikana kulunut energia ratkaisemalla purkauskäyrältä aikavälin alku- ja loppujännitteiden (V_1, V_1) avulla vastaavat varaustilat $VT(V_1)$ ja $VT(V_2)$, jolloin kulunut energia on $E = E_{kok} * (VT(V_1) - VT(V_2))$ (Zhang ym. 2010).

Akkujännitteen avulla suoritettavan energiankulutuksen määrittämisen on todettu olevan tarkkuudeltaan täysin vertailukelpoinen vaihtoehto ulkoiselle mittalaitteelle järjestelmätasoisen energiankulutusmallin muodostamisessa. Korkeamman resoluutionsa ansiosta energiankulutusmalli voidaan tällä menetelmällä muodostaa jopa kertaluokan nopeammin, kuin prosentin resoluutiolla saatavan suhteellisen varaustilan arvoa suoraan käyttämällä (Zhang ym. 2010).

4.1.3 Epäsuoraa tehonmittausta hyödyntävät menetelmät

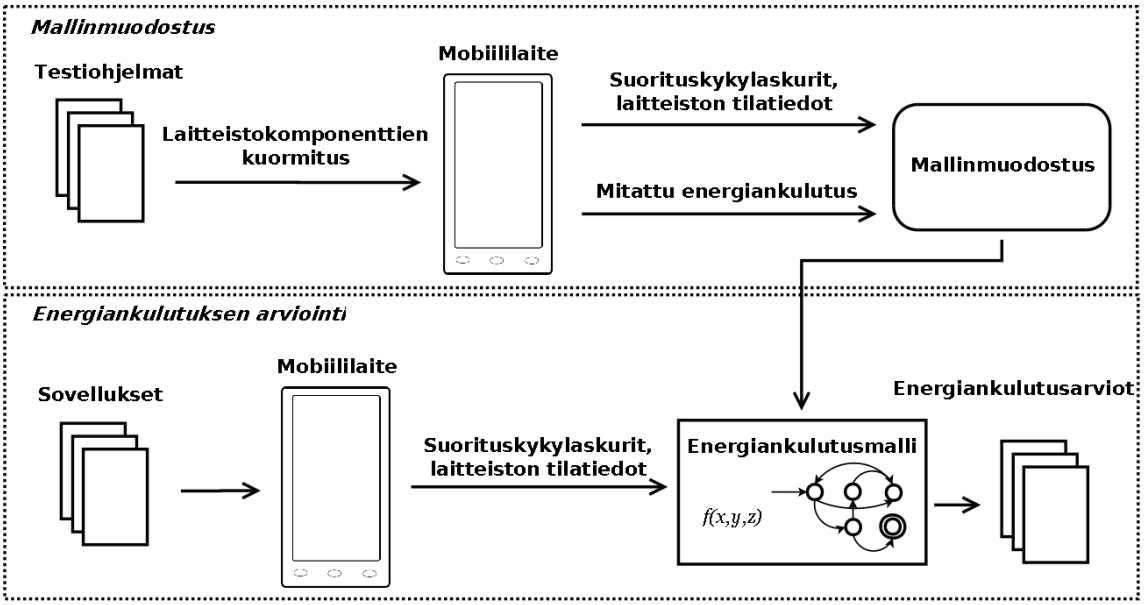
Mobiililaitteissa yleisesti käytetyillä litium-akuilla on sisäistä resistanssia, joka aiheuttaa akkua kuormitettaessa siitä saatavan jännitteen laskun (Waag, Fleischer ja Sauer 2014). Tätä resistanssia voidaan käyttää virranmittausvastuksen tavoin akusta käytettävän virran määrittämiseen, kuten tavallisissa tehonmittausmenetelmissä. Toisin kuin erillistä mittausvastusta käytettäessä, akun sisäisen resistanssin tuottamaa jännitehäviötä ei kuitenkaan voida havainnoida erikseen. Se voidaan kuitenkin mitata välillisesti akun kokonaisjännitteessä tapahtuvia, akun varaustilan vähenemiseen verrattuna nopeita jännitemuutoksia seuraamalla (Xu ym. 2013).

Litium-akun sisäinen resistanssi on tavallisesti hyvin matala ja vastaavasti sen aiheuttama muutos akkujännitteessä on pieni. Käytännössä monet mobiililaitteet tarjoavat tietoa akun jännitteestä kuitenkin jo millivolttien resoluutiolla. Tavallisessa älypuhelimessa tämän tarkkuuden on todettu riittävän laitteen virrankulutuksessa 20 mA:a suurempien muutosten havaitsemiseen 98 %:n todennäköisyydellä (Xu ym. 2013). Menetelmä ei kuitenkaan havaitse erittäin hitaita muutoksia tai täysin tasaista energiankulutusta.

Tavallisesti akun sisäinen resistanssi on kuitenkin tuntematon ja vaihtelee merkittävästi jopa samanlaisten akkuyksilöiden kesken (Dong ja Zhong 2011). Näin absoluuttisen energiankulutuksen mittaaminen menetelmällä vaatii ulkoisella mittalaitteella tehtyjä vertailumittauksia tai sisäisen mittauslaitteiston tuottamaa tietoa akun absoluuttisen varaustilan muutoksesta. Näistä rajoituksista huolimatta epäsuoran tehonmittauksen on osoitettu soveltuvan tarkkuudeltaan hyvän energiankulutusmallin muodostamiseen ilman ulkoista mittalaitetta. Prosentin tarkkuudella saatavaan varaustilatietoon verrattuna tämä menetelmä voi myös nopeuttaa mallin muodostamista jopa kaksi kertaluokkaa (Xu ym. 2013).

4.1.4 Energiankulutusmalleja hyödyntävät menetelmät

Energiankulutusta voidaan arvioida myös ilman mittaamista. Energiankulutusmalleihin pohjautuvan arvioinnin perusajatuksena on kerätä tietoa sovellusta suorittavan järjestelmän tilasta ensisijaisesti joko tapahtumien (mm. Pathak ym. 2011) tai suorituskykylaskureiden arvojen avulla (esim. Dong ja Zhong 2011). Kerätty tilatieto muunnetaan edelleen energiankulutusarvioksi käyttäen mallia, joka kuvaa eri tilojen ja niiden todellisen energiankulutuksen havaittua suhdetta. Tällainen malli voidaan muodostaa mittaamalla laitteen energiankulutusta sen suorittaessa testisovelluksia, jotka asettavat järjestelmän laitteistokomponentit tunnettuihin toimintatiloihin (kaavio 1).



Kuvio 1: Energiankulutusmalleihin pohjautuvan lähestymistavan periaate

Mittaamiseen verrattuna arviointiin liittyy aina virhettä. Mallin laadusta ja tyypistä, käytettävistä tilatiedoista ja niiden tarkkuudesta, alustan laitteistokomponenteista ja jopa tarkasteltavasta sovelluksesta riippuen saavutettava virhe voi vaihdella merkittävästi (McCullough ym. 2011). Koska valmiiseen malliin perustuva arviointi ei kuitenkaan tarvitse aktiivisesti syötteenään muuta kuin järjestelmän tilatietoja, on se ainoa menetelmä, joka on käyttökelpoinen käytännössä kaikilla mobiililaitealustoilla. Tämä on merkittävä etu muun muassa kenttämittauksissa ja käyttäjätutkimuksissa, joissa kehittyneimmätkin energiankulutuksen mittauslaitteistot tuottavat kustannuksia ja ylimääräistä työtä (Brouwers, Zuniga ja Langendoen 2014).

Energiankulutusmalleihin perustuvia menetelmiä on laajojen sovellusmahdollisuuksien ansiosta esitelty kirjallisuudessa moderneille mobiililaitteille jo varsin kattavasti. Tätä menetelmien joukkoa luokittelevina piirteinä voidaan tunnistaa muun muassa eroja mallin muodostamisessa, käytettävissä tilatiedoissa, mallin rakenteessa, energiankulutusarvion muodostuspaikassa, arvion ajallisessa resoluutiossa sekä mahdollisuudessa osittaa energiankulutus sitä tuottavien laitteisto- ja ohjelmakomponenttien kesken.

Energiankulutusmalli on mahdollista muodostaa vain kerran (esim. Shye, Scholbrock ja Memik 2009) tai päivittää sitä dynaamisesti, esimerkiksi sen tuottaman arviointivirheen ylittäessä määrätty taso (Dong ja Zhong 2011). Useimpien modernien mobiililaitteiden laitteistokomponenttien konfiguraatio ei kuitenkaan ole käyttäjän muokattavissa. Näin kerran muodostetun mallin käyttäminen jatkossa samalla laiteyksilöllä ja muilla vastaavilla laitteilla on yksi mallintamiseen liittyvistä tavoitteista, joka on saanut myös käytännön mittauksissa rajallista tukea (Shye, Scholbrock ja Memik 2009).

Saman mallin pitkäaikaiskäytön yhdelläkin laiteyksilöllä voi kuitenkin vaarantaa esimerkiksi käyttöjärjestelmäpäivitys, jos se vaikuttaa laiteajureihin tai esimerkiksi suorittimen dynaamisen kellotaajuuden ja käyttöjännitteen hallinnan toimintatapaan (Dong ja Zhong 2011). Laiteyksilöiden väliseen mallin uudelleenkäyttöön puolestaan liittyy mahdollisena ongelmana komponenttien valmistustoleranssien aiheuttama yksilövaihtelu. Tämän vaikutus todelliseen energiankulutukseen ja siten mallin tuottaman arvion tarkkuuteen voi olla modernien mobiililaitteiden monimutkaisilla komponenteilla jopa kymmenissä prosenteissa (McCullough ym. 2011). Lisäksi laitteiden järjestelmäpiirin ja muiden puolijohteiden ominaisuudet muuttuvat pitkällä aikavälillä käytöstä aiheutuvan kulumisen seurauksena (Zheng ym. 2009).

Mallia muodostettaessa käytetään testiohjelmia, jotka kuormittavat tavallisesti laitteistokomponentteja keinotekoisilla kuormilla erillään toisistaan. Näin kukin komponentti voidaan asettaa määrättyyn toimintatilaan energiankulutuksen mittaamisen ajaksi (Xu ym. 2013). Kuormitetun komponentin tuottaman energiankulutuksen mittaamiseen voidaan käyttää ulkoista mittalaitetta (esim. K. Kim ym. 2014), mutta myös mobiililaitteen tarjoamaa sisäistä mittauslaitteistoa (Maker, Amirtharajah ja Akella 2013). Mallia dynaamisesti päivitettäessä sisäinen mittauslaitteisto on tavallisesti ainoa vaihtoehto. Ulkoinen mittalaite tuottaa kuitenkin parhaan absoluuttisen tarkkuuden ja korkeimman aikaresoluution, jolloin malliin pys-

tytään sisällyttämään myös laitteistokomponenttien tuottamat lyhyet kulutusjaksot. Mobiililaitteiden sisäisten mittauslaitteistojen päivitysnopeus on vielä tavallisesti niin matala, että lyhyemmät kulutusjaksot jäävät niillä tarkasteltaessa piilotiloiksi, joita ei näin voida täydellisesti kuvata mallissa (esim. Jung ym. 2012). Tästä rajoituksesta huolimatta on todettu, että mobiililaitteen sisäistä mittauslaitteistoa käyttäen voidaan muodostaa energiankulutusmalleja, joiden tarkkuus erityisesti järjestelmän kokonaiskulutuksen, mutta myös useimpien laitteistokomponenttien erillisen kulutuksen, tasolla on vertailukelpoinen ulkoista mittalaitetta käyttäen muodostettuihin malleihin (Maker, Amirtharajah ja Akella 2013).

Mallin syötteenä käytettäviä järjestelmän tilatietoja voidaan saada suorituskykylaskureista, kuten suorittimen ytimen käyttöasteesta. Muutokset järjestelmän tilassa ovat kuitenkin seurausta järjestelmäkutsuista, joten tilamuutokset voidaan tunnistaa myös niitä seuraamalla. Kirjallisuudessa on esitetty kumpaakin lähestymistapaa käyttäviä menetelmiä. Laskureina on tarkasteltu myös laitteiston suorituskykylaskureita (Xiao ym. 2010), mutta erityisesti käyttöjärjestelmän suorituskykylaskureita (esim. Xu ym. 2013; Shye, Scholbrock ja Memik 2009; Zhang ym. 2010). Tavallisesti laitteen energiankulutuksen mallintamiseen parhaiten soveltuvien suorituskykylaskureiden kokonaisuus on muodostettu valikoimalla käsin kehittäjien asiantuntemuksen pohjalta (mm. Murmuria ym. 2012; Zhang ym. 2010; Jung ym. 2012). Yhtenä vaihtoehtona laskurikokonaisuuden käsin valitsemiselle on esitetty pienimmän selittävän kokonaisuuden muodostamista automaattisesti pääkomponenttianalyysillä. Tällä voidaan usein sekä varmistaa parhaiten selittävien laskureiden valinta että vähentää malliin tarvittavien laskureiden kokonaismäärää (Dong ja Zhong 2011).

Suorituskykylaskureiden arvoihin perustuvissa malleissa on kuitenkin nähty rajoitteita. Yksi näistä liittyy laskureiden lukemisen väistämättä vaatimaan aikaan ja energiaan, jotka riippuvat suoraan lukutiheydestä. Useimmissa toteutuksissa nämä tekijät ovatkin rajoittaneet lukunopeuden yhteen kertaan sekunnissa (mm. Shye, Scholbrock ja Memik 2009; Dong ja Zhong 2011). Samalla myös mallin tarjoaman arvion ajallinen resoluutio rajoittuu yhteen hertsiin. Korkeampiakin päivitysnopeuksia on tutkittu, mutta menetelmän toteutuksesta riippuen sen oma energiankulutus kasvaa jo alle 10 Hz:n (Pathak ym. 2011), mutta viimeistään 100 Hz:n (Dong ja Zhong 2011) nopeudella useisiin kymmeniin prosentteihin koko järjestelmän energiankulutuksesta ja vääristää näin varsin merkittävästi saatua arviota. Muina suori-

tuskykylaskureiden käytön ongelmina on pidetty muun muassa niiden usein matalaa, laskurikohtaista päivitysnopeutta ja päivityksen sekä laitteiston tilan todellisen muutoksen välistä viivettä (Pathak, Hu ja Zhang 2012; Xiao ym. 2010). Lisäksi mobiililaitteissa on energiankulutuksen kannalta merkittäviä laitteistokomponentteja, joille on ei ole esimerkiksi käyttöjärjestelmän suorituskykylaskuria tai joilla on merkittävää energiankulutusta vielä laskurin kuvaaman käytön jo loputtua (Pathak ym. 2011).

Moderneille mobiililaitteille tehdyt mittaukset ovat lisäksi osoittaneet, että niiden energiankulutuksessa on piirteitä jopa yli 1 kHz:n taajuudella, vaikkakin 99 % energiankulutusspektristä selittyykin tällä tai sitä matalammilla taajuuskomponenteilla (Maker, Amirtharajah ja Akella 2013; K. Kim ym. 2014). Niinpä monien suorituskykylaskureihin perustuvien mallien mahdollistama 1 Hz:n päivitysnopeus ei ole riittävä tunnistamaan merkittävää osaa laitteen energiankulutustapahtumista oikein (K. Kim ym. 2014).

Havaittujen rajoitteiden takia suorituskykylaskureihin perustuvia mallintavia menetelmiä on jopa kuvailtu ensimmäisen sukupolven toteutuksiksi ja esitetty, että seuraavat sukupolvet perustuvat järjestelmäkutsujen sekä muiden tapahtumien seurantaan ja laitteistoa mallintaviin tilakoneisiin (Ding ym. 2013). Rajoituksistaan huolimatta esimerkiksi alle 10 %:n keskimääräinen virhe näytteessään 1 Hz:n nopeudella arvioitaessa ja jopa alle 5 %:n virhe kokonaiskulutuksessa muutamien minuuttien käyttöjaksolle ovat mahdollisia näillä menetelmillä (esim. Zhang ym. 2010). Tarkasteltavan sovelluksen tuottaman kuorman luonteesta riippuen virhe voi kuitenkin nousta useisiin kymmeniin prosentteihin, vaikka se keskimääräiselle kuormalle olisikin matala (Pathak ym. 2011). Matalasta arviointinopeudesta huolimatta saatujen, tarkkuudeltaan suhteellisen hyvien arviointitulosten syynä on erään mallin tapauksessa pidetty virheen tasaista jakaumaa, joka johtaa pitkiä ajanjaksoja tarkasteltaessa yksittäisten virheiden kumoutumiseen (Shye, Scholbrock ja Memik 2009).

Tapahtumienseurantaa suorituskykylaskureiden arvojen korvaajana on perusteltu sillä, että järjestelmäkutsut ovat sovelluksen ainoa tapa käyttää laitteistoa ja sisältävät tiedon laitteistokomponentista, tarvittavasta käyttöasteesta sekä ovat helposti yhdistettävissä resurssin käyttäjään (Pathak ym. 2011). Niinpä ne tarjoavat ilman aktiivista lukemista kerralla enemmän tietoa kuin suorituskykylaskureista on mahdollista ylipäänsä saada. Näiden seikkojen motivoimana on kehitetty menetelmiä, jotka seuraavat järjestelmäkutsuja ensisijaisesti käyttöjär-

jestelmän ytimen tasolla (Pathak, Hu ja Zhang 2012; Pathak ym. 2011) tai sen alapuolella, alustariippuvissa laitteistoajureissa (K. Kim ym. 2014). Vaihtoehtona, joka ei vaadi lainkaan muutoksia käyttöjärjestelmään, on esitetty myös pelkkien sovellusohjelmoinnin rajapinnan kautta tapahtuvien kutsujen seuraamista (Kjærgaard ja Blunck 2012).

Useimmissa julkaistuista toteutuksista tapahtumia on kuitenkin käytännössä seurattu useammalla tasolla. Esimerkiksi aliohjelmakohtaisen energiankulutuksen tunnistavaan malliin voidaan tarvita tietoa sovellusohjelmoinnin rajapinnan, mahdollisen virtuaalikoneen ja käyttöjärjestelmän tasolla olevista tapahtumista (Pathak, Hu ja Zhang 2012). Osa laitteistokomponenteista on myös luonteeltaan sellaisia, etteivät tapahtumat kerro riittävästi niiden tilasta. Tällaisia ovat muun muassa OLED-tekniikkaan perustuvat näytöt, joiden energiankulutus on koko laitteiston tasolla merkittävä ja riippuu lähes täysin niiden kuvasisällöstä (Chen ym. 2013). Sisällön vaihtumiseen ei kuitenkaan liity yksiselitteisesti tunnistettavia tapahtumia, joten tarkkuudeltaan hyvän arvion muodostaminen vaatii käyttöjärjestelmän näyttöpuskurin aktiivista lukemista suorituskykylaskurin tavoin (K. Kim ym. 2014).

Tapahtumienseurantaan perustuvien ja suorituskykylaskureita käyttävien toteutusten välillä on tehty vasta muutamia edustavia vertailuita samoissa testiolosuhteissa. Ne kuitenkin ensisijaisesti puoltavat tapahtumienseurantaan perustuvan mallin tarjoamaa korkeampaa mallinnustarkkuutta jo pelkästään pidempiaikaista järjestelmän kokonaiskulutusta tarkasteltaessa (esim. Pathak, Hu ja Zhang 2012). Ero virhetasoissa voi testattavasta sovelluksesta riippuen olla jopa kertaluokan tapahtumienseurantaan perustuvan menetelmän eduksi (Pathak, Hu ja Zhang 2012; Pathak ym. 2011). Osassa vertailuista on päädytty kuitenkin myös vähäiseen eroon suorituskykylaskureita käyttävän menetelmän eduksi (Dong ja Zhong 2011).

Tavoiteltavan arviointinopeuden ollessa esimerkiksi 1 kHz:n tasolla näistä menetelmistä ainoastaan tapahtumienseuranta on pystynyt tarjoamaan käyttökelpoisia arviointituloksia (K. Kim ym. 2014). Eroa on perusteltu tapahtumienseurantaan itseensä liittyvän matalamman energiankulutuksen lisäksi muun muassa paremmalla mahdollisuudella laitteistokomponenttien käyttöä seuraavan aktivaation ja tilanmuutosten oikea-aikaisella tunnistamisella (K. Kim ym. 2014). Suhteellisen korkeat arviointinopeudet ovat käytännössä välttämättömiä ositettaessa energiankulutusta esimerkiksi siitä vastuullisille prosesseille, säikeille tai erityisesti aliohjelmille (Pathak, Hu ja Zhang 2012; Pathak ym. 2011). Vaikka suorituskykylaskurei-

den käyttöön mallin syötteenä liittyykin näin käyttökohteita rajoittavia puolia, on myös niihin perustuvia menetelmiä kehitetty edelleen aktiivisesti tapahtumienseurantaan perustuvien menetelmien rinnalla (esim. Xu ym. 2013; K. Kim ym. 2014).

Suurimmassa osassa julkaistuista energiankulutusta mallintavista menetelmistä, riippumatta niiden käyttämästä tilanseurantamenetelmästä, on käytetty lineaarista mallia järjestelmän energiankulutuksen kuvaamiseen (Murmuria ym. 2012; Xu ym. 2013; Dong ja Zhong 2011; Zhang ym. 2010; Pathak ym. 2011). Tällaisessa mallissa järjestelmä nähdään erillisten komponenttien joukkona, jonka kokonaisteho tarkasteltavan diskreetin aikavälin t aikana voidaan laskea riippumattomien laitteistokomponenttien i tehonkulutuksen summana, eli $P(t) = \sum_i P_i(t)$ (esim. Zhang ym. 2010). Yksittäisten laitteistokomponenttien käyttöasteen U_i ja energiankulutuksen oletetaan riippuvan lineaarisesti toisistaan, joten komponentin tehoa $P_i(t)$ mallinnetaan ensimmäisen asteen polynomifunktioilla $P_i(t) = \alpha_i U_i(t)$. Tässä energiankulutuksen ja käyttöasteen suhdetta kuvaava vakio α_i määritetään tavallisesti estimoimalla lineaarisella regressioanalyysillä, jossa selitettävänä muuttujana on havaittu tehonkulutus ja selittävänä muuttujana testiohjelmalla komponenttia kuormitettaessa asetettava käyttöastetta kuvaava muuttuja U_i (esim. Shye, Scholbrock ja Memik 2009). Jos malli käyttää syötteenään suorituskykylaskureita, käyttöastetta kuvaavat muuttuja voidaan yleensä johtaa niistä suoraan (mm. Dong ja Zhong 2011). Osa komponenteista voi kuitenkin olla ainoastaan päällä tai pois, jolloin niiden energiankulutuksen vakiokerroin α_i on suoraan mitattavissa ja käyttöastetta kuvaava muuttuja U_i saa vain arvot 1 tai 0. Funktioon voi sisältyä myös kuormituksesta riippumattomaan kulutukseen liittyvä vakio β_i, jolloin komponentin tehoa kuvaava funktio on kokonaisuudessaan $P_i(t) = \alpha_i U_i(t) + \beta_i$ (Jung ym. 2012).

Ensimmäisen asteen polynomifunktioilla on onnistuneesti mallinnettu monia laitteistokomponentteja. Vielä 2010-luvulle tultaessa esimerkiksi useimpien mobiililaitteiden suoritin oli mahdollista mallintaa sen avulla käyttämällä vain muutamaa erillistä, kellotaajuudesta riippuvaa vakiokerrointa (Shye, Scholbrock ja Memik 2009). Mobiililaitteiden suorittimet ovat kuitenkin kehittyneet merkittävästi. Nyt niitä mallintavan funktion vakiokertoimia saatetaan tarvita yli kymmenen suorittimen dynaamiseen kellotaajuuden ja käyttöjännitteen hallinnan eri tiloihin liittyen (esim. Jung ym. 2012). Moniytimisten suorittimien mallintaminen puolestaan vaatii yksimuuttujaisen, lineaarisen polynomifunktion sijaan esimerkiksi useampia

riippumattomia muuttujia sisältävän muodon (K. Kim ym. 2014).

Oletus energiakulutuksen ja käyttöasteen lineaarisesta riippuvuudesta ei kuitenkaan päde kaikille laitteistokomponenteille kaikilla alustoilla. Näin esimerkiksi joidenkin laitteiden LCD-näytön energiankulutusta sen kuvasisällön suhteen on mallinnettu muun muassa toisen asteen polynomifunktion avulla (Xu ym. 2013). Yhtä lailla rajalliseksi on osoittautunut oletus toisistaan riippumattomista laitteistokomponenteista. Niinpä joissain laitteissa osalla laitteistokomponenteista on myös toisten laitteistokomponenttien käyttöasteesta riippuvaa kuormitusta, joka tekee niiden mallintamisen lineaarisella regressioanalyysillä kannattamatonta (McCullough ym. 2011). Käyttökelpoisena vaihtoehtona toisistaan riippuvien komponenttien muodostamien kokonaisuuksien mallintamiseen on pidetty esimerkiksi neuroverkkoja, jotka sietävät hyvin syötemuuttujien välistä riippuvuutta (K. Kim ym. 2014).

Toisin kuin suorituskykylaskureita käyttävissä menetelmissä, tapahtumanseurantaan perustuvissa menetelmissä ei ole suoraan käytettävissä laskurin kaltaista arvoa, joka toimisi laitteistokomponentin käyttämää tehoa mallintavassa funktiossa syötemuuttujana. Niissä voidaan kuitenkin käyttää muuten vastaavia oletuksia järjestelmästä pitämällä yllä keinotekoisia muuttujia. Yksittäiset tapahtumat eivät kuitenkaan kuvaa kaikkien laitteistokomponenttien käytöstä kattavasti, joten muuttujiin tuottavat muutoksia usein tapahtumien lisäksi myös ajastimien laukeaminen ja yhteisvaikutus edellisten tapahtumien kanssa (K. Kim ym. 2014).

Yhtenä laitteistokomponenttien sisäistä toimintaa läheisesti muistuttavana ja siten käytännöllisenä abstraktiona laitteiston tilakirjanpitoon on pidetty äärellisiä tilakoneita (esim. Pathak, Hu ja Zhang 2012). Niiden avulla koko järjestelmä voidaan kuvata yhtenä mallina, jossa eri tiloihin liittyy yksiselitteinen vakiokulutus. Tilakoneen vallitsevan tilan vakiokulutukseen on kuitenkin lisättävä edelleen myös komponenttien käyttöasteesta riippuvaa kulutus, jota mallinnetaan edelleen lineaarista mallia käyttäen (Pathak ym. 2011). Järjestelmän kokonaiskulutuksen analysointia enemmän tilakoneita on kuitenkin kirjallisuudessa tähän asti käytetty verkkoliikenteen seurantaan perustuvaan mobiililaitteen langattomien verkkorajapintojen energiankulutuksen mallintamiseen (mm. Qian ym. 2011; Ding ym. 2013; Vergara, Nadjm-Tehrani ja Prihodko 2014).

Energiankulutusmalleja käyttäviä menetelmiä on kehitetty sekä energiankulutuksen reaaliai-

kaiseen arvioimiseen mobiililaitteella että tilatietojen tallentamiseen ja myöhempään analysointiin. Tallennettujen tilalokien myöhempi analysointi toisella alustalla on käyttökelpoinen vaihtoehto esimerkiksi käyttäjätutkimuksiin, joissa reaaliaikaisesta arviosta ei saada lisäarvoa (Shye, Scholbrock ja Memik 2009). Yhtä lailla se riittää valmiiden ja kehitettävien sovellusten energiankulutuksen tarkkaankin vertailuun ja analysointiin (Pathak, Hu ja Zhang 2012).

Riippumatta siitä, muodostetaanko reaaliaikaista energiankulutusarvioita lainkaan, on tilalokien tallentaminen nähty mahdollisuutena käyttää samaa tallennetta toistuvasti eri tavoin parametrisoiduilla energiankulutusmalleilla. Näin päästään tarkastelemaan yhden testauskerran avulla erilaisia hypoteettisia käyttöolosuhteita, joissa esimerkiksi langattomien verkkotekniikoiden signaalintasot ovat erilaisia (Ding ym. 2013).

Suuri osa kirjallisuudessa esitetyistä energiankulutuksen mallinnusmenetelmien toteutuksista on keskittynyt reaaliaikaiseen arviointiin (mm. Yoon ym. 2012; Xu ym. 2013; Dong ja Zhong 2011; Zhang ym. 2010; K. Kim ym. 2014). Eräs yksinkertaisimmista, reaaliaikaisesta arviosta riippuva sovellusryhmä ovat mobiililaitteen energiankulusta tarkkailevat analyysityökalut. Ne voivat tarjota käyttäjälle esimerkiksi erillisen energiankulutusarvion jokaiselle laitteistokomponentille tai sovellukselle (Zhang ym. 2010).

Pelkkää energiankulutuksen esittämistä tärkeämpänä kehitysaskeleena ja käyttökohteena pidetään mobiililaitteen rajoitetun energiaresurssin hallintaa ja jakamista sitä käyttävien prosessien kesken. Näin käyttäjä voi vaikuttaa muun muassa sovelluksen energiankulutuksen ja suorituskyvyn välillä tehtävään kompromissiin pidentääkseen akusta saatavaa käyttöaikaa (Roy ym. 2011). Yhtä lailla reaaliaikaisesta energiankulutusarviosta hyötyvät myös dynaamisesti laskentaa mobiililaitteesta pilvipalveluun siirtävät menetelmät, jotka voivat sen avulla tehdä päätöksiä laskennan siirtämisestä ja arvioida päätösten energiankulutukseen tuottamia muutoksia (Kwon ja Tilevich 2013).

Laajassa käytössä olevista mobiililaitteiden käyttöjärjestelmistä käytännössä kaikki tarjoavat jo ainakin kumulatiivisen, sovelluskohtaisen ja usein myös laitteistokomponenttikohtaisen arvion energiankulutuksesta (esim. “Battery: making it last on your Windows Phone” 2015; “Apple - Batteries - Maximizing Performance” 2015; “Power Profiles for Android”

2015; “Monitor the battery life, memory usage, CPU usage, and storage space on your device” 2015). Vaikka ne todennäköisesti muodostavat arvionsa ainakin osittain energiankulutusmalliin perustuvalla menetelmällä, niiden dokumentaatiosta käytetty menetelmä ei yleensä käy ilmi. Tässä suhteessa poikkeus on Android-käyttöjärjestelmä, joka dokumentaation mukaan käyttää mallintavaa menetelmää sovellus- ja laitteistokomponenttikohtaisen energiankulutusarvion muodostamiseen (“Power Profiles for Android” 2015).

Android-käyttöjärjestelmään sisältyvä menetelmä käyttää energiankulutusmallin syötteenä ensisijaisesti laitteistokomponenttien virransäästötilaan liittyvien lukkojen tapahtumanseurantaa (“Power Profiles for Android” 2015). Tapahtumien välillä laitteistokomponenttien tilakirjanpitoon ja aktivaation keston seurantaan käytetään ensisijaisesti ajastimia (“platform_frameworks_base/BatteryStatsImpl.java at master - android/platform_ frameworks _base GitHub” 2015). Tapahtumanseurannan lisäksi malli käyttää myös esimerkiksi suorittimeen ja muistiin liittyviä suorituskykylaskureita, joita luetaan sovelluksen vaihtuessa. Menetelmän käyttämä energiankulutusmalli on luonteeltaan lineaarinen ja sen sisältämät laitteistokomponenttikohtaiset vakiokertoimet mittaa kullekin mobiililaitemallille niiden valmistaja (“Power Profiles for Android” 2015).

Tämä energiankulutuksen arviointimenetelmä on ollut osana Android-käyttöjärjestelmää jossain muodossa jo sen vuonna 2008 julkaistuista, ensimmäisistä versiosta lähtien (“History for core/java/com/android/internal/os/BatteryStatsImpl.java - android/platform _frameworks_base GitHub” 2015). Vaikka sen tarjoamat sovellus- ja laitteistokomponenttikohtaiset kumulatiiviset energiankulutusarviot eivät ole virallisesti kuuluneet julkiseen sovellusohjelmoinnin rajapintaan, niitä on kuitenkin voinut käyttää tavallisissa käyttäjätason sovelluksissa. Vuonna 2013 julkaistun Android-version 4.4 myötä tämä mahdollisuus kuitenkin poistettiin tavallisilta käyttäjäsovelluksilta (“Issue 61975 - android - Undo removal of access to BATTERY_STATS permission for apps - Android Open Source Project - Issue Tracker - Google Project Hosting” 2015).

4.1.5 Emulaatiopohjaiset menetelmät

Energiankulutusmalleihin perustuvat menetelmät mahdollistavat energiankulutuksen arvioinnin ilman sen mittaamista, mutta sovellusten testaaminen rajoittuu edelleen mobiililaitealustalle. Malleihin perustuva arviointitapa voidaan kuitenkin yhdistää myös simuloituun laitteistoon, jolloin energiankulutuksen arviointia on mahdollista suorittaa jo kehitysalustalla, ilman erillistä mobiililaitetta. Näin saadaan lähestymistapa, jonka vahvuutena on pidetty erityisesti toiminnallisen testauksen ja energiankulutuksen arviointiprosessin yksinkertaistamista sekä mahdollisuutta koko ohjelmistopinon energiankulutuksen arviointiin (mm. Mittal, Kansal ja Chandra 2012; Tu ym. 2014).

Käytännössä kaikki mobiililaitteiden kaupallisten käyttöjärjestelmien sovelluskehitysympäristöt sisältävät jo sovellusten kehitysympäristössä tapahtuvaa toiminnallista testaamista tukevan työkalun, jonka ne esittelevät emulaattorina tai simulaattorina (esim. “Using the Emulator | Android Developers” 2016; “About Simulator” 2016; “Test with the Microsoft Emulator for Windows 10 Mobile” 2016). Käytetystä nimestä riippumatta työkalujen yhteinen tavoite on tarjota ympäristö mobiilisovellusten ja niitä tukevan ohjelmistopinon suorittamiseen. Näin niitä voidaan pitää mobiililaitealustan arkkitehtuurin toteuttavina emulaattoreina (“QEMU” 2016; Smith ja Nair 2005).

Kaupallisten sovelluskehitysympäristöjen sisältämien emulaattoreiden toteutustavat vaihtelevat. Useimmat niistä perustuvat suorituskyvyltään vaativien, interaktiivisten sovellusten tukemiseksi laitteistoavusteiseen virtualisointiin ja käyttävät useimpia kehitysalustan laitteista sellaisenaan (esim. “Using the iOS Simulator to test and debug AIR applications” 2016; “Test with the Microsoft Emulator for Windows 10 Mobile” 2016; “SailfishOS first application” 2016). Näin niillä ei voida suorittaa sovelluksen testauksen kannalta edustavinta, mobiililaitteelle tarkoitettua käännösversiota. Toteutustavaltaan merkittävän poikkeuksen muodostaa kuitenkin Android-alustan emulaattori, joka tarjoaa mobiililaitteita vastaavan ARM-suoritusympäristön (Tu ym. 2014).

Kehitysalustan tarjoamat resurssit, esimerkiksi verkkoyhteyden osalta, eroavat tavallisesti mobiililaitteiden keskimääräisten käyttöolosuhteiden resurssitasosta. Lisäksi useimmista kehitysympäristöistä puuttuu mobiililaitteille tyypillisiä oheislaitteita kuten GPS, kamera tai

kiihtyvyysanturi. Niinpä sovelluskehitysympäristöjen emulaattorit tavallisesti tarjoavatkin esimerkiksi simuloidun kameran. Muita simuloituja oheislaitteita tai mahdollisuus realistisemman testauksen vaatimaan resurssien skaalaamiseen on kuitenkin toteutettu vasta harvoihin emulaattoreihin (esim. “Test with the Microsoft Emulator for Windows 10 Mobile” 2016).

Mahdollisuus myös energiankulutuksen arviointiin toiminnallisen testauksen yhteydessä laajentaisi emulaattoreiden käyttömahdollisuuksia. Tässä käytössä sen tarjoama simuloitu ympäristö on nähty jopa mobiililaitteella tapahtuvaa testausta edustavampana, koska sen tarjoamien resurssien taso on mahdollista sovittaa vastaamaan sovellusten keskimääräisiä käyttöolosuhteita (Mittal, Kansal ja Chandra 2012). Lisäksi simuloitu ympäristö voi parhaimmillaan vähentää laitteella tapahtuvassa testauksessa vaikuttavan ympäristön tuottamaa satunnaisvaihtelua ja näin helpottaa tarkasteltavan ilmiön havainnointia (Tu ym. 2014).

Emulaatioon perustuva energiankulutuksen arvioiminen riippuu kuitenkin aina edustavista tila- ja ajoitustiedoista, joilta vaadittavan granulariteetin määrää valittu energiankulutusmalli. Energiankulutuksen arviointimalleista yksi yksinkertaisimmista ja ainoastaan komponenttien ulkoisista tiedoista riippuva vaihtoehto on äärellisten tilakoneiden ja lineaaristen regressiomallien käyttäminen (esim. Benini, Hodgson ja Siegel 1998). Tätä yksityiskohtaisempina, granulariteetiltaan tarkempina vaihtoehtoina erityisesti suorittimen osalta on käsitelty muun muassa mallia, joka kuvaa yksittäisten konekäskyjen tai -käskyluokkien kuluttaman energian (esim. Li ja John 2003; Mittal, Kansal ja Chandra 2012; Lin ym. 2015). Lisäksi tarkasteluun on käytetty myös suorittimen rekisteri- tai porttitasoisen rakenteen energiankulutusta kuvaavia malleja (Sakamoto ym. 2002). Nämä matalan tason mallit riippuvat kuitenkin vastaavalla tasolla toimivista simulaatiomenetelmistä, joiden suorituskyky ei vielä käytännössä riitä interaktiivisten sovellusten tai kokonaisten ohjelmistopinojen testaamiseen (Varma ym. 2008; Benini, Hodgson ja Siegel 1998).

Granulariteetiltaan yhteensopivan tilatiedon lisäksi energiankulutuksen mallintamisen kannalta olennaista on suorituksen ajoituksen mallintaminen. Karkeimmillaan tähän on käytetty tarkasteltavaa alustaa vastaavan suoritusnopeuden toisintamista emulaatioon käytettäviä resursseja rajoittamalla (Mittal, Kansal ja Chandra 2012). Tätä edustavampia ja tarkempia ratkaisuja ovat olleet suorittimien osalta muun muassa konekäskyille taulukoitujen, kiinteiden

viiveiden käyttö (Luiz Sartor, Brisolara Correa ja Schneider Beck 2013), käskyjen suoritusaikojen arviointi tilastollisesti (Eeckhout ym. 2003) tai arviointi mikroarkkitehtuurin toimintaa dynaamisesti simuloivien mallien avulla (Li ja John 2003; Tu ym. 2014). Oheislaitteiden ajoituskäytöksen simuloinnissa käytetyt ratkaisut ovat puolestaan vaihdelleen laitetyypeittäin, mutta muun muassa äärellisiin tilakoneisiin ja ajastimiin pohjautuvat mallit ovat olleet yleinen vaihtoehto (esim. Benini, Hodgson ja Siegel 1998; Tu ym. 2014).

Energiankulutuksen arviointiin ja siihen liittyvään ajoituksen määrittämiseen käytettävissä olevien tilatietojen taso riippuu laitteiston simulaatiototeutuksen toimintatavasta. Aiemmin mainituista, natiivia suoritusta virtualisoinnin muodossa käyttävistä emulaattoreista tietoa saadaan yleensä rajallisesti ja lähinnä korkealla tasolla. Tämän pohjalta on mahdollista suorittaa energiankulutuksen arviointia, mutta arkkitehtuurien erot rajoittavat tarkkuutta merkittävästi ja saatavien arvioiden granulariteetti on matala (Mittal, Kansal ja Chandra 2012; Gerin, Hamayun ja Pétrot 2009). Edustavampien tulosten saavuttamiseksi natiiviin suoritukseen on yhdistetty ajoitustietojen ja edustavan kontrollivuon keräämiseksi muun muassa ohjelman esianalyysi ja annotaatio staattisilla menetelmillä (Brandolese, Corbetta ja Fornaciari 2011) tai toisella, ajoitustarkalla simulaattorilla (T. K. Tan ym. 2002). Vaaditusta esianalyysistä huolimatta nämä menetelmät ovat osoittautuneet käyttökelpoisiksi myös kokonaisten ohjelmistopinojen energiankulutuksen tarkastelussa (Gerin, Hamayun ja Pétrot 2009).

Toinen, virtualisoinnin ohella merkittävä mobiililaitealustojen emulaattoreissa käytettävä toteutustekniikka on ollut käskykantasimulaatio (ISS). Sen avulla testattavan sovelluksen ja muun ohjelmistopinon kohdealustalle tehtyjä käännösversioita voidaan suorittaa sellaisenaan tai vähäisin muutoksin. Vaikka suoritin ja mahdollisesti myös oheislaitteet on tällöin toteuttava niiden toiminnan tarjoavilla simulaatiomalleilla virtualisoinnin sijaan, voi menetelmän suorituskyky ilman ajoitussimulaatiota vastata tarkasteltavaa mobiililaitealustaa (Hsu, Hung ja Tu 2010). Lisäksi käskykantasimulaation toimiminen konekäskyjen tasolla mahdollistaa mikroarkkitehtuurin simulaatiomallien integroinnin ja myös ulkoisten, yksityiskohtaisten ajoitus- ja energiankulutusmallien käytön ilman testattavan koodin instrumentointia (esim. Hübert ja Stabernack 2010; Tu ym. 2014; Varma ym. 2008; J. Lee ym. 2008).

Lukuisista vaihtoehtoisista toteutusratkaisuista ja mahdollisista eduistaan huolimatta energiankulutuksen määrittämiseen liittyvät ominaisuudet ovat mobiililaitteiden kaupallisten käyt-

töjärjestelmien emulaattoreissa kuitenkin edelleen harvinaisia. Yksi merkittävä syy on todennäköisesti keskittyminen suorituskykyyn. Tästä lähtökohdasta toteutetut, puhtaasti funktionaaliset emulaattorit eivät mallinna laitteiston toimintaa. Näin niistä on saatavilla vain vähän tietoa ajoitus- ja energiankulutusmallien käyttöön.

Kirjallisuudessa on kuitenkin jo esitetty joitain toteutuksia energiankulutuksen arviointiin myös mobiililaitteiden arkkitehtuureja vastaavilla simuloiduilla alustoilla. Näistä moni on rajoittunut pelkän suorittimen toiminnan simulaatioon ja energiankulutuksen mallintamiseen (esim. Brandolese, Corbetta ja Fornaciari 2011; Muttreja ym. 2005). Näyttö, verkkorajapinta ja erilaiset laitteistokiihdyttimet muodostavat kuitenkin yleensä merkittävän osan mobiililaitteiden energiankulutuksesta (Chen ym. 2013; Vergara, Nadjm-Tehrani ja Prihodko 2014; Varma ym. 2008). Lisäksi mobiilisovellukset riippuvat merkittävästi muun ohjelmistopinon tarjoamista palveluista (S. Lee ym. 2015). Näin sovellusten testaaminen ilman näitä komponentteja tai prosessina ilman muun ohjelmistopinon palveluita ja niiden energian arviointia ei tarjoa edustavaa kuvaa energiankulutuksesta.

Suorittimen ohella esimerkiksi väylän, muistin ja mahdollisia kiihdyttimiä simuloivia toteutuksia on julkaistu vasta muutamia (mm. Varma ym. 2008; Lin ym. 2015). Sovelluskehittäjän kannalta ehdottomasti hyödyllisempiä, kaikki olennaisimmat mobiililaitteen arkkitehtuurin komponentit simuloivia ja niiden energian mallintavia toteutuksia on esitetty yhtä lailla vain muutamia. Samalla näissä julkaisuissa on kuitenkin myös keskitytty jonkin kaupallisten mobiililaitteiden ohjelmistopinon suorittamiseen (mm. Tu ym. 2014; Mittal, Kansal ja Chandra 2012).

Mobiililaitteen energiankulutusta tarkastelevista emulaatiototeutuksista kahdessa kattavimmassa on kummassakin käytetty toteutuksen pohjana valmista ohjelmistokehitysympäristön emulaattoria. Nämä ovat tarjonneet valmiina mahdollisuuden mobiililaitteen ohjelmistopinon suorittamiseen. Molemmissa toteutuksissa on myös käytetty äärellisiä tilakoneita ja lineaarisia regressiomalleja oheislaitteiden energiankulutuksen mallintamiseen, mutta muilta osin niissä käytetyissä ratkaisuissa on eroja (Mittal, Kansal ja Chandra 2012; Tu ym. 2014).

Ensimmäisen toteutuksen pohjana on käytetty Windows Phone 7 -emulaattoria, joka perustuu virtualisoimalla toteutettuun natiivisuoritukseen. Näin sen tarjoama suorituskyky riip-

puu täysin kehitysympäristöstä, eikä sen ajoitus vastaa sellaisenaan realistista mobiililaitetta. Energiankulutuksen arvioinnin mahdollistamiseksi tähän alustaan on lisätty emulaattorin resurssien skaalaamiseen perustuva ajoitusmalli, jonka valintaa on perusteltu interaktiivisten sovellusten suorituskyvyn ja samalla käytöksen säilyttämisellä. Suorittimen osalta skaalaaminen on toteutettu emulaattorin kellojaksoja rajoittamalla ja verkkoyhteyteen on vastaavasti lisätty langattomia verkkotekniikoita simuloiva kanavamalli. Muiden energiankulutuksen kannalta oleellisten oheislaitteiden ajoitusta ei kuitenkaan ole mallinnettu (Mittal, Kansal ja Chandra 2012).

Energiankulutuksen mallintaminen on suorittimen osalta toteutettu lineaarisella mallilla, joka huomioi ainoastaan käytettyjen kellojaksojen osuuden mahdollisesta maksimimäärästä. WLAN- ja 3G-yhteydet on käsitelty niiden toimintaa mallintavilla äärellisillä tilakoneilla. LCD-näytön energiaa arvioidaan kirkkaudesta riippuvalla lineaarisella mallilla ja AMOLED-pohjainen näyttö on käsitelty värisisältöön perustuvan mallin avulla (Mittal, Kansal ja Chandra 2012). Vaikka emulaattorin tarjoama arkkitehtuuri eroaa merkittävästi mobiililaitteiden vastaavasta, on toteutuksen keskimääräinen virhetaso alle 10 %. Vertailuun käytetyt sovellukset on kuitenkin valittu kuormittamaan ensisijaisesti mallinnettuja komponentteja, joten niiden tuloksia ei voida pitää täysin yleistettävinä esimerkiksi mallin ulkopuolelle jätettyä GPU:ta käyttävien sovellusten tapauksessa (Mittal, Kansal ja Chandra 2012).

Toteutuksista toisessa pohjana on Android-emulaattori, joka perustuu QEMU-ympäristöön. Näin se tarjoaa binäärisen translaation avulla toteutetun simuloidun ARM-suorittimen ja paravirtualisoituja oheislaitteita (“QEMU” 2016). Tuki ajoituksen ja energiankulutuksen arviointiin on toteutettu lisäämällä QEMU:n suoritinsimulaattoriosaan käskyjenseuranta, jolloin käskyt voidaan välittää suorituskyky- ja energiankulutusmalleille. Suorittimen ajoituksen käsittelemiseen on lisätty useita eritasoisia malleja, joista yksinkertaisin käyttää ainoastaan taulukoituja, käskykohtaisia CPI-arvoja suoritusaikojen laskemiseen. Malleista yksityiskohtaisimmassa huomioidaan puolestaan useimmat mikroarkkitehtuurin piirteet, kuten liukuhihnan viiveet ja haarautumisen ennakointi (“Performance Analysis Tool for Heterogeneous Multicore Virtual Platform” 2011). Samalla nämä ajoitusmallit toimivat edelleen tilatietojen lähteenä suorittimen energiankulutusta mallintavalle äärelliselle tilakoneelle, joka huomioi myös dynaamisen jännitteen ja kellotaajuuden säädön (Tu ym. 2014).

Emulaattorin suoritinsimulaation käskyjenseuranta tarjoaa samalla tietoa MMIO:sta, joka yhdessä oheislaitteiden simulaatiomallien tilatietojen kanssa mahdollistaa niiden energiankulutuksen käsittelemisen. Näin on voitu äärellisten tilakoneiden avulla mallintaa myös LCD-näytön, verkkorajapinnan, äänilaitteen, GPS:n ja flash-muistin energiankulutusta. Verrattaessa menetelmän kokonaiskulutuksesta tuottamaa arviota laitteistoon oli virhe noin 8 %. Tämä taso saavutettiin käyttämällä suorittimen energiankulutuksen arviointiin yksinkertaisinta, vain sen käyttöasteesta riippuvaa mallia kalibroimattomana. Jos se kalibroitiin vertailualustan suorittimen kulutustiedoilla, oli virhetaso vain noin 2 % (Tu ym. 2014).

Android-alustalle on julkaistu myös toteutuksia, jotka mahdollistavat ajoituksen ja näin sovellusten suorituskyvyn arvioinnin, mutta eivät tarjoa sen pohjalta energiakulutusarviota (mm. Chidambaram Nachiappan ym. 2014; Sunwoo ym. 2013). Näissä yhtenä mielenkiintoisena toteutusvaihtoehtona on esitetty suorittimen funktionaalisen ja ajoitustarkan simulaation vuorottelevaa yhdistämistä. Lisäksi tätä ajatusta on jo sovellettu myös energiankulutuksen arviointiin yleisesti ARM-alustalla, keskittyen kuitenkin ohjelmistopinojen sijasta yksittäisten ohjelmaprosessien analysointiin (Muttreja ym. 2005).

4.1.6 Ohjelma-analyyttiset menetelmät

Tehonmittaukseen ja energiankulutusmalleihin perustuvat menetelmät määrittävät sovellusten energiankulutuksen välillisesti, niitä suorittavan laitteiston energiankulutusta mittaamalla tai arvioimalla. Tälle toimintatavalle suurelta osin ortogonaalinen vaihtoehto on laitteistotason ohittaminen ja sovellusten suora analyysi niiden energiankulutuksen arvioimiseksi. Näin toimivassa ohjelma-analyyttisessa lähestymistavassa käytetään energiankulutusmalleja, joihin sisältyy kuvaus ohjelman tarkasteltavalta abstraktiotasolta, esimerkiksi konekielestä, energiankulutukseksi (esim. Brandolese, Corbetta ja Fornaciari 2011; Hao ym. 2012). Koska sovelluksen suorituksen seuranta on usein mahdollista lisätä sen osaksi, voi sama arviointiperiaate olla käyttökelpoinen toimittaessa mobiililaitealustalla (Hao ym. 2012), kehitysympäristössä (Brandolese, Corbetta ja Fornaciari 2011) tai simulaatioympäristössä. Perustuminen sovelluksen toiminnan eristettyyn seuraamiseen mahdollistaa ympäristöstä, kuten ohjelmistopinon taustaprosesseista ja prosessinvaihdosta, aiheutuvan satunnaisvaihtelun poistamisen kokonaan energiankulutusarviosta. Tämä lisää ohjelma-analyyttisen menetelmän tarkkuutta

suuruusluokaltaan pieniä eroja tarkasteltaessa (Hao ym. 2012, 2013).

Ohjelma-analyyttiseen lähestymistapaan perustuvat arviointimenetelmät ovat kuitenkin simulaatiomenetelmien tavoin edelleen harvinaisia. Yleisemmin sulautettujen järjestelmien osalta on esitetty joitain ratkaisuja RISC-alustoille tarkoitettujen C-ohjelmien arviointiin (esim. Brandolese, Corbetta ja Fornaciari 2011). Modernien mobiililaitteiden kontekstissa tutkimus on puolestaan rajoittunut Android-alustalle, jolle on julkaistu muutamia toteutuksia. Näistä tärkeimmät perustuvat suoritusympäristön tai sovelluksen instrumentointiin, mobiililaitteella tapahtuvaan suoritustallenteen keräämiseen ja myöhempään analyysiin energiankulutuksen määrittämiseksi (Behrouz ym. 2015; Hao ym. 2012, 2013).

Toteutuksista ensimmäinen julkaistiin vuonna 2012. Alkuperäisessä muodossaan se tarjoaa Android-sovellusten energiankulutuksen arviointia sovellus- ja metoditasolla, käsitellen ainoastaan suorittimen osakulutusta. Analyysi perustuu sovellusten instrumentointiin ja suorituksesta kerättävään tallenteeseen, joka muunnetaan energiankulutusarvioksi tavukooditasoisella mallilla. Koska suoritusympäristöä ei ole instrumentoitu, kirjastometodien ja järjestelmäkutsujen energiankulutus on jouduttu käsittelemään erillään muusta mallista. Tähän on käytetty lineaarisia, empiirisesti muodostettuja metodikohtaisia malleja, jotka kuvaavat energian esimerkiksi argumentin koon funktiona (Hao ym. 2012).

Kirjastometodien ja oheislaitteiden käsittelyn puutteiden takia mallin virheen todettiin olevan sovellustasolla keskimäärin 5 % ja metoditasolla 9 % tarkasteltaessa tavallisiin sovelluksiin nähden epäedustavia, ainoastaan suoritinta kuormittavia testejä. Myöhemmin menetelmää kuitenkin jatkokehitettiin tuottamaan lähdekoodirivitasoisia arvioita ja sen mallia paranneltiin (Hao ym. 2012, 2013). Tehdyt muutokset mahdollistivat edeltäneen version tarkkuustason säilyttämisen edustavammissa testitapauksissa, mutta toivat samalla esiin ohjelma-analyyttiseen lähestymistapaan liittyviä perustavanlaatuisia rajoitteita. Näihin kuuluivat oheislaitteet, jotka käsiteltiin instrumentoimalla niiden tilasiirtymät mobiililaitealustalla ja tekemällä energiankulutusmallit tiloista riippuviksi (Hao ym. 2013). Laitteistosta riippuvan osuuden lisääminen malliin olisi kuitenkin todennäköisesti mahdollista välttää mallintamalla oheislaitteiden sisäistä toimintaa simulaatiomallien tavoin (esim. Tu ym. 2014).

Toinen tunnistettu rajoitus olivat ulkoisesta tiedosta riippuvat metodit, joille lisättiin uusi

mallikerros yhdistämään pyydettävän resurssin yksilöivä tunniste, resurssin koko sekä sen mitattu prosessointiaika metodikohtaisten mallien käyttöön. Yksilöiviä tunnisteita sisältävät mallit oli kuitenkin muodostettava aina suoritusaikana sovellus- tai käyttökertakohtaisesti ja ne asettavat vaatimuksia resurssien saatavuudelle testausympäristössä (Hao ym. 2013). Oikeiden sovellusten kaikki resurssit eivät yleensä ole testausympäristön hallinnoimia, joten riippuvuutta suorituksen aikana kerättävistä tiedoista ei todennäköisesti ole mahdollista poistaa kokonaan.

Myöhemmin ohjelma-analyyttistä lähestymistapaa on sovellettu myös samaan ryhmään kuuluvien sovellusten energiankulutuksen vertailuun. Tässä käytössä arviolta vaadittava granulariteetti ja absoluuttinen tarkkuus eivät kuitenkaan ole muiden käyttötarkoitusten tasolla. Niinpä arviointiin valittiin pelkästään API-kutsujen keskimääräisen energiankulutuksen huomioiva energiankulutusmalli, jolla sovellusten metoditasoinen energiankulutus määritettiin staattisesti. Lopulliseen kokonaisarvioon sisällytettävät metodit valittiin dynaamisen toiminnan huomioimiseksi suorittamalla sovelluksia satunnaisia tapahtumia tuottavassa testiympäristössä, keräten samalla tallenne metodikutsuista. Näin saadun menetelmän todettiin pystyvän järjestämään joukko sanakirjasovelluksia mittaustuloksiin verrattuna edustavasti (Behrouz ym. 2015). Koska menetelmän testauksessa käytetyillä sanakirjasovelluksilla on kuitenkin todennäköisesti vähän riippuvuuksia esimerkiksi ulkoisista resursseista, voi API-kutsujen energiankulutuksen hajonta olla niissä keskimääräistä sovellusta vähäisempää. Näin on mahdollista, että pelkkiä kiinteitä kustannuksia API-kutsuille käyttävän mallin tarkkuus ei ole riittävä edes suhteelliseen arviointiin kaikissa sovellusryhmissä.

Täysin staattisesti tehtävää sovellusten energiankulutuksen arviointia on sivuttu edellisessä vertailumenetelmässä, mutta lopullinen arvio on koottu edelleen dynaamisten polkutietojen pohjalta (Behrouz ym. 2015). Todennäköisesti puhtaasti staattisesti toimivaa ohjelma-analyyttista menetelmää energiankulutuksen arviointiin ei ole vielä julkaistu mobiililaitteiden kontekstissa. Edellä esitettyjen menetelmien energiankulutusmallien periaatteet voisivat silti olla käyttökelpoisia myös staattisessa kontekstissa, koska ne on kehitetty tallenteiden analysointiin (Hao ym. 2012, 2013; Behrouz ym. 2015). Dynaamisesti, edustavia testitapauksia suorittamalla saatava tieto sovelluksen kontrollivuosta on kuitenkin osoittautunut hankalaksi korvata jo pelkästään sovellusten tapahtumapohjaisuuden ja kirjastoriippuvuuden takia.

Yksi mahdollinen vaihtoehto on testitapausten automaattisen tuottamisen yhteydessä tutkittu symbolinen suorittaminen, jossa ohjelma käydään läpi keräten symbolisiin argumentteihin kunkin polun niille asettamat rajoitteet. Näin voidaan tuottaa myös energiankulutuksen arvioinnin käyttöön teoriassa kaikki sovelluksessa mahdolliset kontrollivuot, mutta ei kuitenkaan tietoa niiden yleisyydestä erilaisissa käyttötapauksissa (Mirzaei ym. 2012). Staattista analyysiä rajoittavat kuitenkin kaikki sovelluksen ja ympäristön vuorovaikutuksesta seuraavat haasteet, jotka on tunnistettu dynaamisesta toiminnasta kerättyjä tallenteita analysoitaessa. Lisäksi kaikki sovelluksen suorituksen oikeellisuuteen vaikuttavat ohjelmistokehyksen ja kirjastometodien toiminnot on käsiteltävä esimerkiksi toiminnallisilla tynkäluokilla, joita ei ole saatavilla valmiina (Mirzaei ym. 2012).

4.2 Menetelmien teoreettinen vertailu

Edeltävissä alaluvuissa energiankulutuksen määritysmenetelmiä on käsitelty lähinnä niihin liittyvän teorian, erityispiirteiden ja kirjallisuudessa julkaistujen toteutusten yksityiskohtien näkökulmasta. Tässä luvussa niitä on kuvattu tiiviimmin, keskittyen erityisesti niiden soveltamiseen liittyviin piirteisiin, kuten menetelmille ominaisiin rajoituksiin ja etuihin, arvion mahdolliseen ohjelmakomponenttitasoiseen granulariteettiin ja valmiiden toteutusten saatavuuteen.

Sähkötehoa mittaavat menetelmät tuottavat tietoa mobiililaitteen todellisesta energiankulutuksesta. Näistä suora tehonmittaus ulkoisella mittalaitteella on myös sovellettavissa käytännössä kaikkiin mobiililaitteisiin, vaikkakin sisäänrakennettuihin akkuihin tarvittavia järjestelyjä on pidetty haastavina (esim. Oliner ym. 2013; Rice ja Hay 2010; “Mobile Device Power Monitor Manual - Monsoon Solutions” 2008). Toisaalta on myös esitetty, ettei enemmistöllä ohjelmistokehittäjistä ole välttämättä riittävää asiantuntemusta mittaamiseen liittyvistä järjestelyistä ja samalla mittalaitteiden korkea hankintahinta rajoittaa menetelmän vain harvojen saataville (Maker, Amirtharajah ja Akella 2013). Näistä rajoituksista jälkimmäistä ei kuitenkaan voida enää välttämättä pitää merkityksellisenä, koska Arduinon kaltaiset (“Arduino - Home” 2016), mikro-ohjaimen sisältävät laitteistonkehitysalustat kaupallisine lisämoduuleineen ovat tehneet energiankulutuksen mittaamiseen tarvittavasta laitteistosta edullisen ja yksinkertaisen (Holleis ym. 2013; Hindle ym. 2014; “INA219 High Side DC Current Sensor Breakout” 2016). Ulkoisiin mittalaitteisiin perustuvat järjestelyt eivät kuitenkaan yleensä ole kuljetettavia, mutta myös mobiililaitteiden akkurajapinnat voivat tukea mittaamista ja tehdä ainakin toteutusratkaisuja vertailevassa käytössä erilliset mittausjärjestelyt tarpeettomiksi (Maker, Amirtharajah ja Akella 2013; Höpfner, Schirmer ja Bunse 2012). Osa mobiililaitteista myös tukee tätä mittaustapaa jo aikaisempaa korkeammalla, yli 5 Hz:n näytenopeudella (“Measuring Device Power | Android Open Source Project” 2016), mutta kattavaa tai aktiivisesti ylläpidettyä listaa eri laitteiden mittausmahdollisuuksista ei todennäköisesti ole missään tarjolla.

Energiankulutusta tuottavien ohjelmisto- tai laitteistokomponenttien ja kulutuksen välistä korrelaatiota ei ole mahdollista muodostaa pelkkien mittausten perusteella. Ilman korrelaatiota tehdyt mittaukset ovat silti sellaisenaan käyttökelpoisia, jos tarkasteltavat tapahtumat

ovat tunnistettavissa mittaustallenteesta. Näin saavutettava granulariteetti voi käytännössä olla yksittäisten testitapausten kumulatiivisen energiankulutuksen tasolla (Hindle ym. 2014). Virheenhaun seurantamenetelmät mahdollistavat kuitenkin useimmilla alustoilla sovelluksen suorituksen seuraamisen, joka tukee yksinkertaisen korrelaation ja siten granulariteetiltaan yksityiskohtaisemman analyysin muodostamista (esim. Brouwers, Zuniga ja Langendoen 2014)

Modernien mobiililaitteiden monikerroksiset ohjelmistopinot hankaloittavat kuitenkin energian kokonaiskulutuksesta saatavien mittaustulosten käyttöä sovellusten arvioinnissa, koska ne käyttävät usein hankalasti seurattavia taustaprosesseja, jotka voivat aiheuttaa merkittäviä energiankulutusjaksoja (Ding Li ym. 2013; Hao ym. 2013). Jos näitä tapahtumia ei voida tunnistaa, on ne käsiteltävä esimerkiksi toistamalla mittausta ja karsimalla tilastollisesti poikkeavat tulokset (Brouwers, Zuniga ja Langendoen 2014). Julkaistuista menetelmistä osa pystyy kuitenkin jo tunnistamaan nämä tapahtumat ja huomioimaan niiden vaikutuksen, sekä tarjoamaan rutiini- ja lähdekoodirivitasoista energiankulutuksen korrelaatiota (Ding Li ym. 2013; Chung, Lin ja King 2011). Näin jopa yksittäisestä testikerrasta voidaan muodostaa yksityiskohtainen energiankulutusarvio sovelluksen suoritetuille osille.

Mitatun energiankulutuksen ja komponenttien korrelaatiota käsittelevät menetelmät ovat kuitenkin vielä harvinaisia ja rajoittuneet kokonaan Android-alustalle. Niissä käytettyjä toteutusperiaatteita sen sijaan on pidetty ensisijaisesti siirrettävinä, joskin esimerkiksi käyttöjärjestelmän ytimestä kerättyjä tapahtumia ei ole mahdollista käyttää suljetuilla alustoilla. Tämä rajoittaa erityisesti energiankulutuksen ja sitä tuottavien laitteistokomponenttien välisen korrelaation tarkastelua, jota on muutenkin käsitelty vielä suhteellisen vähän (Ding Li ym. 2013; Brouwers, Zuniga ja Langendoen 2014). Lisäksi tähänastisista toteutuksista todennäköisesti ainoastaan yksi on julkaistu ohjelmistokehittäjien käyttöön (“neat-power-toolkit Mobile power analysis toolkit for Android smartphones” 2013), eikä sitä ole tiettävästi enää ylläpidetty. Kaupallisista ratkaisuista puolestaan luultavasti yksikään ei tarjoa korrelaatiotietoja, vaikkakin osa suorituskyvyn arviointisovelluksista lukee myös tehonkulutuksen sitä tukevasta mobiililaitteesta matalalla päivitysnopeudella (“Trepn Profiler” 2016). Näin ohjelmistokehittäjien on ensisijaisesti toimittava pelkkien mittausten pohjalta, tai toteutettava itse energiankulutusmittausten korrelaatiota riittävällä granulariteetilla käsittelevät työkalut.

Epäsuoralla tehonmittauksella ja **akun tilaa hyödyntävillä menetelmillä** on pyritty poistamaan riippuvuutta sähkövirtaa mittaavasta laitteistosta. Pelkkiin mobiililaitteiden yleisesti tarjoamiin akun varaustila- ja jännitetietoihin perustuvina nämä lähestymistavat ovat suhteellisen hyvin siirrettäviä. Erillisestä mittalaitteesta riippumattomina ne myös yksinkertaistavat useilla, mahdollisesti erilaisilla laitteilla tapahtuvien mittausten järjestelyjä, tukevat kenttämittauksia ja mahdollistavat arvion lähes reaaliaikaisen hyödyntämisen mobiililaitteella (Xu ym. 2013; Zhang ym. 2010).

Ohjelmistopinojen tuottamat energiankulutusjaksot rajoittavat myös näiden menetelmien tarkkuutta sovellusten energiankulutuksen arvioinnissa, koska ne mittaavat laitteen energian kokonaiskulutusta (Ding Li ym. 2013). Yhtä lailla energiankulutuksen ja sen aiheuttajan korrelaation käsitteleminen vaatii suoran mittauksen tavoin myös tilatietojen seurantaa. Epäsuoria ja akun tilaa hyödyntäviä menetelmiä ei kuitenkaan ole ensisijaisesti tutkittu energiankulutuksen itsenäisen mittaamisen kannalta, vaan ensisijaisesti mallipohjaisen arvioinnin mallien muodostamisen välineenä (Xu ym. 2013; Zhang ym. 2010). Näin ympäristöihin liittyviä tekijöitä tai korrelaatiota ei ole käsitelty kirjallisuudessa. Lisäksi näillä mittausmenetelmillä saavutettu mittausnopeus on parhaimmillaankin ollut vain yksi näyte sekunnissa (Xu ym. 2013). Tämä nopeus on liian matala ohjelmistopinon kulutustapahtumien vaikutusten poistamiseen ja rajaa ohjelmistokomponenttien korrelaation käsittelyä huomattavasti (Ding Li ym. 2013). Niinpä mittauksen granulariteetti voi olla testitapauksen kokonaisenergian tasolla, mutta testiä on voitava suorittaa toistuvasti ohjelmistopinon satunnaisvaikutusten poistamiseksi ja riittävän keston saavuttamiseksi.

Epäsuoraa tehonmittausta ja akun tilaa hyödyntäviä menetelmiä on tutkittu vasta suppeasti, eikä yhtäkään toteutusta ole julkaistu vapaaseen käyttöön. Kehitetyt ratkaisut ovat kuitenkin verraten yksinkertaisia ja ne on kuvattu tarkkuudella, joka myös mahdollistaa tarvittaessa niiden toteuttamisen ja käyttöönoton. Molemmat lähestymistavat riippuvat kuitenkin täysin mobiililaitteen akun yksityiskohtaisista ominaisuuksista ja vaikka tämä herkkyys on tunnistettu, sen vaikutuksen suuruutta ei ole tutkittu (Xu ym. 2013; Zhang ym. 2010). Yhtenä selvänä lyhyen aikavälin virhelähteenä on pidetty akun lämpötilavaihtelua, joka rajoittaa mallien tarkkuutta muun muassa kenttäolosuhteissa (Zhang ym. 2010). Pitkäaikaisessa käytössä myös akun kulumisesta tukee merkittävä tekijä, joka vaatii akkua kuvaavan mal-

lin aktiivista päivittämistä (Lee, Chon ja Cha 2015). Lisäksi laitevalmistajien yritys helpottaa energiankulutuksen haasteita akun latausaikaa lyhentämällä on todennäköisesti pysyvästi johtamassa akkujen sisäisen resistanssitason laskemiseen ("Qualcomm Quick Charge FAQs" 2016; "NCP4371 Product Preview" 2016). Tämä voi rajoittaa entisestään virranmittauksessaan sisäisestä resistanssista riippuvan epäsuoran tehonmittauksen mahdollistamaa mittaustarkkuutta.

Energiankulutusmalleihin perustuvat menetelmät käyttävät ainoastaan järjestelmän tilaa kuvaavia tietoja energiankulutuksen arviointiin, jonka ansiosta lähestymistapa on mobiililaitteilla toimivista menetelmistä laajimmin siirrettävä ja kokonaan ohjelmallisena myös yksinkertaisimmin käyttöön otettava vaihtoehto. Koska arvion muodostaminen perustuu järjestelmän loogisten laitteiden mallintamiseen yleensä erillään, mahdollistaa se muita menetelmiä luontevammin kulutuksen ja siihen vaikuttavien laitteistokomponenttien välisen korrelaation käsittelyn. Lisäksi myös ohjelmakomponenttien ja niiden energiankulutuksen välillä voidaan saavuttaa prosessi-, säie- ja rutiinitasoinen korrelaatio (Pathak, Hu ja Zhang 2012). Tähän tarvittava korkea, 1 kHz:n tasoinen näytenopeus rajaa kuitenkin menetelmälle käyttökelpoiset tilatietolähteet käyttöjärjestelmän ytimen tasolle, josta tietoja ei välttämättä voida kerätä kaupallisilla, suljetuilla ohjelmistopinoilla toimittaessa. Näillä alustoilla voidaan silti käyttää käyttäjäsovellusten saavutettavissa olevia tilatietoja. Näin suoritettavan arvioinnin käyttökelpoinen näytenopeus on kuitenkin selvästi matalampi, noin 1 Hz, joka rajoittaa ohjelmakomponenttien ja energiankulutuksen välisen korrelaation granulariteetin pidempien testitapausten tasolle (K. Kim ym. 2014; Shye, Scholbrock ja Memik 2009).

Muodostettavien energiankulutusarvioiden tarkkuuden määrää ensisijaisesti energiankulutusmallin kyky kuvata tarkasteltavan laiteyksilön todellisia energiankulutusominaisuuksia. Edeltävien sukupolvien yksiytimisiä järjestelmäpiirejä käyttävien mobiililaitteiden on todettu olevan käsiteltävissä yksinkertaisilla malleilla, jotka riippuvat pienestä joukosta tilaa kuvaavia muuttujia tai tapahtumia (esim. Jung ym. 2012). Suorituskyvyn ja virranhallinnan kehitys on kuitenkin monimutkaistanut laitteistokomponenttien toimintaa niin, että energiankulutuksen mallintamisen kannalta tärkeitä tietoja niiden sisäisestä toiminnasta ei välttämättä voida enää lainkaan havainnoida käyttöjärjestelmän tasolta saatavista tilatiedoista (McCullough ym. 2011). Tämä on lähestymistavalle perustavanlaatuinen haaste, johon ei

vielä esimerkiksi moniytimisten järjestelmäpiirien osalta ole kehitetty kattavia mallinnusratkaisuja, joiden virhetaso olisi täysin laitteiston kuormituksen luonteesta riippumaton (K. Kim ym. 2014; McCullough ym. 2011). Osalle laitteistokomponenteista on mallissa huomioitava myös niiden sisäisen tilan lisäksi vuorovaikuttava ympäristö, kuten verkkorajapintojen tapauksessa signaalinvoimakkuus, tarkan arvion muodostamiseksi (Ding ym. 2013). Malleihin kohdistuvat vaatimukset ovat myös entisestään monimutkaistumassa, kun muun muassa järjestelmäpiirien homogeenisten ytimien lukumäärä edelleen kasvaa ja heterogeenisiä ydinkokonaisuuksia sisältävät toteutukset leviävät laajempaan käyttöön ("big.LITTLE Technology - ARM" 2016; "Snapdragon 810 Mobile Processor (8 Core) | Qualcomm" 2016). Näiden tekniikoiden tai esimerkiksi erilaisten laitteistokiihdytinten vaikutuksesta menetelmän käyttökelpoisuuteen tai tarkkuuteen ei kuitenkaan ole vielä saatavilla tietoa.

Toinen olennaisesti tarkkuuteen ja menetelmän käytettävyyteen liittyvä kysymys on mahdollisuus energiankulutusmallin uudelleenkäyttämisestä useammilla samantyyppisillä laiteyksilöillä. Käytännössä laitteistokomponenttien satunnaisvaihtelun ja ohjelmiston erojen on todettu rajoittavan uudelleenkäyttöä niin, että hyvän tarkkuuden saavuttamiseksi malli on yleensä optimoitava vastaamaan testattavaa laiteyksilöä vähintään käyttöönottovaiheessa (Xu ym. 2013), mutta todennäköisesti myös pidempiaikaisen käytön jatkuessa (Dong ja Zhong 2011). Tämä tekee menetelmän edelleen riippuvaiseksi energiankulutuksen mittausmenetelmistä, joista tarkkuudeltaan heikommat epäsuoran mittaamisen, akun tilapohjaisten menetelmien ja laitteiden akkurajapintojen tarjoamat mittaustulokset voivat riittää mallin päivittämiseen (Dong ja Zhong 2011). Nämä mittausmenetelmät voivat kuitenkin rajoittaa niillä muodostettavien mallien mahdollista tarkkuutta, joka tukee myös suoran, ulkoisiin mittalaitteisiin perustuvan mittauksen tarvetta mallinmuodostuksessa (Dong ja Zhong 2011; Maker, Amirtharajah ja Akella 2013).

Laajasta tutkimuspohjastaan huolimatta energiankulutusmalleihin perustuvat menetelmät eivät myöskään vaikuta saavuttaneen asemaa, jossa mobiililaitteiden valmistajat julkaisisivat laitteistoa kuvaavia tietoja tai valmiita energiankulutusmalleja. Menetelmän toteutuksista on lisäksi julkaistu ohjelmistokehittäjien käyttöön ainoastaan muutamia toteutusratkaisuiltaan yksinkertaisimpia, ja käytännössä poikkeuksetta niistäkin on tarjottu vain energiankulutuksen arviointiin liittyvää osuutta ja muutamia energiankulutusmalleja julkaisuaikaisille lait-

teille. Jatkokäytön kannalta olennaisia työkaluja uusien laitteiden mallintamiseen ei ole julkaistu, eikä menetelmiä ole muutenkaan ylläpidetty (esim. “Appscope” 2012; “PowerTutor” 2011). Lisäksi toteutukset ovat rajoittuneet kokonaan Android-käyttöjärjestelmälle, kuten valtaosa menetelmään liittyvästä tutkimuksestakin. Tälle alustalle on myös julkaistu kaupallinen, energiankulutusmalleihin perustuva määritysmenetelmä (“Mobile Enerlytics” 2016).

Emuloituun laitteistoon perustuvissa menetelmissä energiankulutusta arvioidaan edelleen laitteistoa kuvaavien mallien avulla, mutta niissä käsiteltävät tilat ovat tavallisesti mobiililaitteilla käytettäviin malleihin verrattuna yksityiskohtaisempia. Näillä menetelmillä energiankulutuksen arviointi on mahdollista tehdä kehitysympäristön osaksi ja suorittaa kokonaan ilman mobiililaitteita. Sovelluskehittäjän kannalta tämä yksinkertaistaa ja nopeuttaa testaamisen työnkulkua, koska siirtymää laitteiden välillä ei tarvita. Lisäksi mobiililaitteen ja sen luonnollisen käyttöympäristön poistaminen vähentää testauspaikasta esimerkiksi verkkoyhteyden laatuun aiheutuvaa vääristymää (Mittal, Kansal ja Chandra 2012). Kun nämä osat korvataan simulaatiomallilla, voidaan käsitellä keskimääräistä tai heikointa odotettavaa yhteydenlaatua ja samalla poistaa ympäristön satunnaisvaihtelua, joka voi peittää testikohtaisia eroja (Mittal, Kansal ja Chandra 2012).

Sovelluksen suorituksesta emuloidussa järjestelmässä kerättävä tieto käsitellään kokonaan sen ulkopuolella, joten tiedonkeruu tai arviointimallit eivät vaikuta tarkasteltavan sovelluksen toimintaan tai muodostettavaan energiankulutusarvioon (Tu ym. 2014). Näin arviointia voidaan suorittaa jopa käskytasoisesti ja erottaa analysoitavan koodin energiankulutuksesta kokonaan muun ohjelmistopinon vaikutus (Mittal, Kansal ja Chandra 2012; Lin ym. 2015). Tämä mahdollistaa aiemmin käsiteltyjä menetelmiä pienempien energiankulutusmuutosten havainnoimisen ja tukee granulariteetiltaan yksityiskohtaista, jopa lähdekoodirivitasoista energiankulutuksen käsittelemistä (Brandolese, Corbetta ja Fornaciari 2011; Lin ym. 2015). Arvioinnin suorittaminen kehitysympäristössä tukee myös muita vaihtoehtoja paremmin palautteesta riippuvia, kehitysaikaisia energiankulutuksen optimointimenetelmiä (Tu ym. 2014).

Sensoreiden käsitteleminen emuloidussa ympäristössä vaatii niitä kuvaavia simulaatiototeutuksia. Yksinkertaisimmillaan ne voivat perustua laitteella tehtäviin tallenteisiin, jotka tarvitsee kerätä jokaiselle testille vain kerran (Austin, Larson ja Ernst 2002). Tätä hankalampia

ovat erilaiset kiihdytinlaitteistot, kuten DSP:t ja GPU, joita kuvaavia energiankulutusmalleja ja simulaatiototeutuksia ei välttämättä ole helposti saatavilla (Varma ym. 2008). Lisäksi yksityiskohtaiset, kattavasti alustan laitteistokomponentteja käsittelevät simulaatiototeutukset voivat olla laitteistoon verrattuna hitaita (Chidambaram Nachiappan ym. 2014). Tämä vääristää sovelluksen ja muun ohjelmistopinon sekä käyttäjän ja ulkoisten resurssien vuorovaikutusta, joka voi heikentää arvion tarkkuutta (Tu ym. 2014).

Emulaatioon perustuvat menetelmät integroivat jo toiminnallisen testaamisen osaksi kehitysympäristöjä ja voisivat tarjota myös yksityiskohtaisia energiankulutusarvioita, mutta kaupallisia toteutuksia ei ole vielä julkaistu. Menetelmän soveltamista mobiililaitteiden ohjelmistopinoihin on käsitelty muutamissa julkaisuissa ja energiankulutusarvioiden mahdollisen virhetason on todettu olevan alle 10 % (Mittal, Kansal ja Chandra 2012). Ainakin yhden toteutuksen lähdekoodi Android-sovellusten analyysiin on myös jo julkaistu, mutta sitä ei ole ylläpidetty tai dokumentoitu kattavasti (“Downloads - hmavp” 2013).

Ohjelma-analyyttiset menetelmät pyrkivät arviointitapaan, joka ei riipu mobiililaitteista tai sen emulaatiosta. Niissä arvio muodostetaan mallilla, joka kuvaa sovelluksen lähdekoodin tai sen johdannaisten, kuten käännöksen, elementtien energiankulutusta (Hao ym. 2012). Tämän periaatteen ansiosta menetelmä on täysin siirrettävä ja soveltuu hyvin esimerkiksi sovelluskehitysympäristön osaksi. Arviot ovat myös hyvin vertailtavia, koska ne eivät riipu ohjelmistopinon tai suoritusympäristön vaihteluista. Sovelluksen tiloihin perustuvana lähestymistapa tukee myös hyvin lähdekoodin yksityiskohtaisella, rivitasoisella granulariteetilla suoritettavaa arviointia (Hao ym. 2013). Lisäksi tämä lähestymistapa tarjoaa myös monia muita mallipohjaisten menetelmien eduista, kuten uudelleenanalysoinnin useammilla erilaisilla energiankulutusmalleilla (Hao ym. 2012).

Mobiililaitteiden piirissä julkaistut toteutukset perustuvat kuitenkin yleensä edelleen laitealustalla kerättäviin suoritustallenteisiin (Hao ym. 2013, 2012). Lähestymistavan staattiset, kokonaan suorituksesta riippumattomat toteutukset ovat harvinaisia ja niiden haasteena on pidetty sovelluksen edustavien kontrollivoiden tunnistamista (Mirzaei ym. 2012). Ohjelma-analyyttiselle menetelmälle merkittävä haaste on myös sovelluksen ympäristöstä, johon kuuluvat laitteistokomponentit ja kaikki ulkopuoliset resurssit, jotka on käsiteltävä joko mittaamalla, simuloimalla tai esimerkiksi heuristisesti arvioimalla. Vaaditut energiankulutusmallit

ovat myös monimutkaisia, ja niiden toteutuksessa on yhtymäkohtia emulaatiopohjaisten menetelmien mallien kanssa. (esim. Li ja John 2003; Lin ym. 2015).

Menetelmän toimintaperiaate näyttää soveltuvan kuitenkin hyvin tarkkuusvaatimukseltaan matalamman, suhteellisen energiankulutuksen määrittämiseen, jossa tarvitut mallit voivat olla yksinkertaisempia, eikä ympäristöä ole välttämättä tarpeellista käsitellä (Mirzaei ym. 2012). Energiankulutuksen arvioinnin lisäksi ohjelma-analyyttisellä lähestymistavalla voi tulevaisuudessa olla merkitystä ylimääräistä energiankulutusta tuottavien ohjelmointivirheiden automaattisessa tunnistamisessa ja toteutusten automaattisessa optimoinnissa (Alam ym. 2014; J. Wang ym. 2012). Lähestymistavan toteutuksia on kuitenkin käsitelty vasta vähän ja ainoastaan Android-alustalla, eikä toteutuksia ole julkaistu kehittäjien saataville (Hao ym. 2013, 2012). Myöskään kaupallisia toteutuksia ei todennäköisesti ole.

Yhteenvetona voidaan todeta, että menetelmätutkimusta on tehty viimeisten viiden vuoden aikana paljon, mutta se on johtanut vain harvoihin julkaistuihin työkaluihin. Lisäksi markkinoille on tullut useita kaupallisia toimijoita, jotka tarjoavat testaamiseen keskittyviä ratkaisuja (“Mobile App Testing | Keynote” 2016; “Robotium Tech” 2016), tai esimerkiksi alustariippumattoman ohjelmakehityksen työkaluja, joiden osana on testausympäristö (“Xamarin” 2016). Kaupallisten välineiden runsaudesta huolimatta ne eivät vielä yleensä käsittele lainkaan energiankulutuksen määrittämiseen liittyviä kysymyksiä.

4.3 Mobiililaiteympäristön asettamat haasteet

Mobiililaiteympäristössä energiankulutuksen määrittämistä ja siihen liittyvää testaamista hankaloittavia ominaisuuksia on jo sivuttu määritysmenetelmien yhteydessä, niistä kullekin merkittävien piirteiden osalta. Osa tällaisista ominaisuuksista vaikuttaa laajemmin, jopa määritysmenetelmästä riippumatta. Tässä luvussa käsitellään näistä piirteistä testauksen suorittamisen ja tulosten luotettavuuden kannalta tärkeimpiä. Tarkastelu on jaoteltu hierarkkisesti kolmelle tasolle laitteiston, ohjelmistopinon ja ulkoisen ympäristön vaikutuksista. Mobiilisovellusten testaamisen liittyy luonnollisesti myös muita kysymyksiä, joista osa voisi tukea energiatehokkuuden saavuttamista esimerkiksi kehitys- ja testausmenetelmien muodossa. Tämä tarkastelu rajoittuu kuitenkin sellaisiin testaukseen liittyviin haasteisiin, jotka suoraan vaikuttavat energiankulutuksen arviointiin.

4.3.1 Laitteistoalustan vaikutukset

Mobiililaitteet perustuvat järjestelmäpiireihin (SoC), joissa samaan piiriin yhdistyvät useimmat järjestelmän komponenteista. Tavallisesti se sisältää vähintään suoritinytimet, GPU:n, radiorajapintoja, laitteistokiihdyttimiä ja rajapinnat ulkoisille komponenteille. Erillisillä laitteistokomponenteilla on yleensä toteutettu lähinnä flash-muisti, RAM-muisti, näyttö, GPS-vastaanotin ja sensorit (“Snapdragon 805 Processor Product Brief” 2016; Chidambaram Nachiappan ym. 2014). Eräälle nykyaikaiselle älypuhelimelle suoritetuissa mittauksissa todettiin, että sen laitteistokomponenteista eniten sähköenergiaa voivat kuluttaa suoritinytimet ja GPU. Näitä seurasivat järjestyksessä näyttö, 3G-radio, kamera, GPS ja RAM-muisti. Muiden komponenttien, kuten flash-muistin ja sensoreiden, vaikutukset olivat kertaluokan tai kaksi edellisiä matalampia (Carroll ja Heiser 2013).

Tehojärjestyksessä kolmea merkittävintä komponenttia, suoritinta, GPU:ta ja näyttöä, on pidetty energiankulutuksen määrittämisen kannalta helposti käsiteltävinä. Näistä kaikkien energiankulutus seuraa käyttöastetta ilman merkittävää viivettä ja on arvioitavissa energiankulutusmalleilla, myös GPU:n tapauksessa (Jin, He ja Liu 2015), suhteellisen hyvällä tarkkuudella (esim. Pathak ym. 2011; Chen ym. 2013). Langattomien yhteystekniikoiden radiorajapinnat on yleisesti tunnistettu edellisiä hankalammiksi käsitellä. Yksi niihin liittyvistä

haasteista on tiedon sisäinen puskurointi, joka voi viivästää pyynnön ja energiankulutuksen ajallista suhdetta (Hao ym. 2013). Radiorajapintoja ei myöskään sammuteta heti niiden käytön jälkeen, joten ne kuluttavat energiaa sekä yhteyden avaamiseen ennen lähetystä, että myös tiedonsiirron jälkeen, joka vääristää siirrettävän tiedon määrän ja energiankulutuksen muuten varsin lineaarista suhdetta. Lisäksi tiedonsiirtoa seuraava aktiivisuusaika riippuu mobiilidataverkoissa operaattorin tekemistä valinnoista (Mittal, Kansal ja Chandra 2012). Mallintavissa menetelmissä käyttöä seuraava aktiivisuusaika on määritettävä malliin muodostusvaiheessa mittausten kautta ja kaikissa menetelmissä sen energia on jaettava jollain säännöllä aktivaatioon osallistuville prosesseille (Pathak, Hu ja Zhang 2012). Eri yhteystekniikoiden (2/3/4G ja WLAN) energiankulutusprofiilit eroavat myös merkittävästi toisistaan ja mobiililaitteet vaihtavat tekniikoita niiden saatavuuden mukaan (Balasubramanian, Balasubramanian ja Venkataramani 2009). Tämä voi aiheuttaa erityisesti kenttämittausten tuloksiin vaihtelua, joka voi olla tarpeen välttää estämällä tekniikan vaihtaminen.

Suoritinta ja näyttöä on kuvattu tavallisiksi laitteiksi vastakohtana eksoottisille laitteille, joihin kuuluvat muun muassa sensorit (Pathak ym. 2012). Sensoreista selvästi eniten energiaa kuluttaa kamera (D. Li ym. 2014; Carroll ja Heiser 2013), jonka merkitys on luultavasti kasvamassa lisätyn todellisuuden sovellusten myötä. Energiankulutusta mallintavissa menetelmissä tämä laite on käsitelty vasta harvoin, todennäköisesti sen lyhyiden käyttöjaksojen takia (esim. Pathak ym. 2011). Kameran selvänä ongelmana on pidetty sen asynkronista energiankulutusta, joka alkaa käyttöönotosta ja päättyy sen sammuttavaan tapahtumaan. Näin sillä ei ole määrällistä, käyttöä kuvaavaa tilaa, vaikka sen energiankulutus riippuu yleensä merkittävästi asetuksista, joilla se on otettu käyttöön (Pathak ym. 2011; Chen ym. 2013). Kamera hallitsee kuvaamisen energiankulutusta, mutta sen aktivoiminen aiheuttaa energiankulutusta myös kuvaa välittävissä ja käsittelevissä kiihdyttimissä, eikä tätä voida helposti tunnistaa tai eritellä (Chen ym. 2013; Chidambaram Nachiappan ym. 2014). Lisäksi kameraan liittyvä ohjelmointimalli voi ainakin osittain perustua erilliseen kamerasovellukseen, johon siirrytään kuvaamisen alkaessa. Tällöin sovelluksen kuluttamasta energiasta osa kirjataan helposti kamerasovellukselle, eikä sitä kutsuneelle, analysoitavalle prosessille (Gao ym. 2016).

Muita sensoreita, kuten kiihtyvyysantureita, on käsitelty vasta vähän kirjallisuudessa ja niiden energiankulutus on myös selvästi kameraa pienempi (Carroll ja Heiser 2013). Ohjel-

mointimalliltaan ne ovat kameran tavoin asynkronisia, ja päivitykset saadaan rekisteröitymisen jälkeen tapahtumina, jotka käyttöjärjestelmä on esikäsitellyt. Sensorit tarjoavat tavallisesti useampia toimintatiloja, joissa tietoa saadaan eri päivitysnopeudella tai tarkkuudella ja joiden energiankulutus on erilainen (esim. “Sensors Overview | Android Developers” 2016; “Motion Events” 2016). Sovellus voi myös odottaa tai kerätä sensoritapahtumia taustalla olematta aktiivinen, mutta järjestelmä kuluttaa silti energiaa (D. Li ym. 2014). Yhdessä nämä monimutkaiset toimintatavat tekevät sensoreiden seurannasta ja energiankulutuksen osittamisesta oikeille prosesseille hankalaa (Ravindranath ym. 2012). Lisäksi sensoreiden käyttäminen testauksessa voi vaatia mittaamista mobiililaitteella, koska läheskään kaikki emulaattorit eivät tarjoa niiden toimintoja. Energiankulutuksen kannalta edustavaan arviointiin ei välttämättä riitä edes staattinen laboratoriotestaus, koska sensoreiden oma energiankulutus ja niiden tuottamat tapahtumat voivat tiedon esikäsittelyn takia riippua täysin niiden keräämästä mittausdatasta (“MPU-6500 | InvenSense” 2016).

GPS-vastaanotin on radiorajapintojen jälkeen yksi eniten ohjelmointikäytänteiden ja energiankulutuksen kannalta tutkittu laite (esim. D. H. Kim ym. 2010; Liu, Xu ja Cheung 2013). Sen suhteellinen energiankulutus on merkittävä ja ohjelmointimalliltaan se vastaa muita sensoreita, toimien rekisteröidyillä tapahtumilla ilman käyttöastetta kuvaavaa muuttujaa (D. Li ym. 2014; Pathak ym. 2011). GPS-vastaanottimeen aktivaatio ei liity suoraan sen tuottamiin tuloksiin, vaan sen käyttöönotto vaatii satelliittien hakemista, joka voi kestää ensimmäisellä kerralla satoja sekunteja, mutta myöhemmin vain kymmeniä sekunteja (Carroll ja Heiser 2013). Käytön aikana vastaanottimen energiankulutus vaihtelee paikkatietoja laskettaessa ja käytön loppuessa sillä on sulkemista seuraavaa aktivaatiota (Brouwers, Zuniga ja Langendoen 2014). Muiden sensoreiden tavoin ei ole varmaa, että laboratoriomittaukset GPS:n toiminnasta ovat edustavia, sillä vaikka se toimisi sisätiloissa, pysyminen paikallaan voi vaikuttaa osassa tapauksista energiankulutuskäyttäytymiseen (Liu, Xu ja Cheung 2013; “BCM47521” 2016).

4.3.2 Käyttöjärjestelmän ja suoritusympäristön vaikutukset

Mobiilisovelluksen eivät pääse suoraan käyttämään energiaa kuluttavaa laitteistoa. Esimerkiksi Android-ohjelmistopinossa sovelluksen vuorovaikuttavat ohjelmointirajapinnan kans-

sa, joka käyttää laitteistoa Androidin järjestelmäpalvelujen kautta IPC:n avulla. Järjestelmäpalvelut puolestaan ovat yhteydessä laiteajureihin HAL:in kautta (“Android Interfaces and Architecture | Android Open Source Project” 2016). Tämä kerroksittaisuus monimutkaistaa energiankulutuksen tulkintaa, koska pyyntöjä voidaan puskuroida ja käsitellä useilla kerroksilla. Näin suoritettava toiminto voi muuttua ja energiankulutus viivästyä, jos sovelluksen toimintaa seurataan helpommin kerättävien, korkean tason pyyntöjen kautta (K. Kim ym. 2014).

Analysoitavaa sovellusta suoritetaan aina samanaikaisesti muiden mahdollisten sovellusten ja järjestelmäpalveluiden kanssa. Prosessinvaihtoa käyttöjärjestelmään tai esimerkiksi suoritusympäristön roskienkeruuta ei ole tavallisesti mahdollista hallita. Tällaisten tapahtumien kesto on yleensä lyhyt, mutta teho suhteessa suuri, joten ne voivat vaikuttaa selvästi testin energiankulutukseen (Hao ym. 2013). Sovelluksen eivät voi modifioimattomilla ohjelmistopinoilla yleensä seurata näitä tapahtumia, joten tavallisena ratkaisuna on käytetty testien automaattista toistamista ja niiden energiankulutuksen keskiarvon määrittämistä (Hao ym. 2013; Corral ym. 2013). Ohjelmistopinon taustaprosesseista osaan, kuten langattomien yhteystekniikoiden ylläpitoon ja push-viestintään, voi kuitenkin vaikuttaa määrittämällä energiankulutusta lentokonetilassa, jos verkkoyhteyttä ei tarvita (Corral ym. 2015).

4.3.3 Laitteen käyttöympäristön vaikutukset

Radiorajapinnat ja GPS-vastaanotin ovat merkittäviä yksittäisiä energiankuluttajia mobiililaitteissa (Carroll ja Heiser 2013). Testauspaikalla ei voida odottaa vallitsevan keskimääräisen käyttäjän signaalintasoa vastaavia olosuhteita tai edes tasaisia olosuhteita, jotka eivät vaihtelisi eri vuorokaudenaikojen tai päivien välillä. Käyttäjien todelliset olosuhteet myös vaihtelevat merkittävästi, mutta tavallisesti vallitsevia testiolosuhteita ei ole mahdollisuutta hallita (Vallina-Rodriguez ym. 2010). Signaalintason vaikutus langattomien verkkotekniikoiden energiankulutukseen on merkittävä ja sen vaikutus erityisesti heikon signaalintason tilanteessa voi olla yli 50 % (Ding ym. 2013). Lisäksi heikko tai riittämätön signaalintaso nostaa myös GPS-vastaanottimen energiankulutusta (Liu, Xu ja Cheung 2013; Chon ym. 2011). Kuten sensorien tapauksessa, laitteen liikkuminen vaikuttaa myös mobiilidatan toimintaan. Usein toistuvat solunvaihdot ja tekniikan vaihtaminen lisäävät energiankulutusta

paikallaan tehtävään mittaukseen verrattuna (Balasubramanian, Balasubramanian ja Venkataramani 2009)

Mobiililaitteiden käyttöympäristöjen on todettu vaihtelevan merkittävästi esimerkiksi lämpötilan osalta (Vallina-Rodriguez ym. 2010). Lämpötilan vaihtelun tiedetään vaikuttavan muun muassa mobiililaitteiden akkujen kapasiteettiin ja sähköisiin ominaisuuksiin (Zhang ym. 2010). Toinen, tunnistettu mutta mobiililaitteiden energiankulutuksen kannalta vähemmän käsitelty tekijä on lämpötilan noustessa kasvava puolijohteiden vuotovirta, joka nostaa niiden energiankulutusta (Gupta ym. 2008). Tämän ilmiön mahdollista vaikutusta vielä korostaa kuormitustilanteissa mobiililaitteiden rajallinen jäähdytys ja suorittimen tehokkaiden kellotaajuudensäätötilojen käyttäminen.

5 Energiatehokkaan mobiiliohjelmoinnin käytänteet

Suorituskyky on yksi tärkeistä tietojärjestelmien laatuun liittyvistä vaatimuksista. Sen piiriin kuuluvat sekä järjestelmän ulkoiset ominaisuudet, kuten mahdollinen transaktio- tai käyttäjälukumäärä, että sisäiset piirteet, kuten suoritinajan tai muistitilan varauksen tehokkuus. Näihin kysymyksiin keskittyvän tutkimuksen lähtökohdiksi on luonnehdittu varhaisten laitteistojen rajallista suorituskykyä, ja sen jatkuvaa merkitystä puoltaa ohjelmistojen laitteiston suorituskykyä nopeammin kasvava vaativuus. Suorituskykyyn liittyvien tavoitteiden tärkeys onkin tunnistettu laajasti ja siihen liittyviä kysymyksiä käsitellään ohjelmistotekniikan vakiintuneella osa-alueella *software performance engineering* (Dumke 2001; Woodside, Franks ja Petriu 2007). Tämän alan laajuutta kuvastaa hyvin, että sen yksittäisistä osa-alueista, kuten olio-ohjelmoinnin suorituskyvyn mittaamisesta, on julkaistu tuhansia artikkeleita 2000-luvulla (Maplesden ym. 2015). Lisäksi suorituskykyisten ohjelmien kehitystä tukevista välineistä jo monet, kuten kääntäjäoptimoinnit tai koodin kokoa optimoivat työkalut, ovat aktiivisessa käytössä (esim. “ProGuard” 2016). Vaikka perustellusti voidaankin esittää, että energiankulutus on ohjelmiston muistinvarauksen tehokkuuden tavoin sen sisäisen suorituskyvyn piirre, ei sillä vaikuta vielä olevan perinteisempien suorituskykyulottuvuuksien tapaista, vakiintunutta asemaa.

Suorituskykyä painottavan tutkimuksen tavoitteet ovat mittaamiseen, analysointiin ja kehittämiseen liittyvät välineet ja aktiviteetit, jotka tukevat ohjelmistokehitysprosessia (Woodside, Franks ja Petriu 2007). Energiankulutuksen tapauksessa mittaaminen on nostettu vastaavalla tavalla sen ymmärtämisen ja näin parantamisen yhdeksi edellytykseksi (Esmaeilzadeh ym. 2011). Arvokkaina on pidetty myös energiatehokkaan ohjelmoinnin hyviä käytänteitä (Linares-Vasquez ym. 2014). Tällaisia käytänteitä on kuitenkin käsitelty mobiilisovellusohjelmoinnissa vasta vähän ja ne ovat yleensä kapea-alaisia, algoritmin tai resurssin käyttötapaa kuvaavia malleja (mm. Metri, Shi ja Brockmeyer 2015; Corral ym. 2014; Datta, Bonnet ja Nikaein 2013).

Käytänteiden rajallista määrää ja abstraktiotasoa voi selittää ainakin osittain modernien mobiilisovellusten lyhyt historia, mutta osa tutkijoista on myös pitänyt niiden mahdollista yleistettävyyttä heikkona ja arvoa ainoastaan rajallisena (Ding Li ym. 2013; Hao ym. 2012). Oh-

jelmien energiankulutukseen vaikuttavien tekijöiden monimutkaisuudesta on saatu alustavaa näyttöä muun muassa kirjastovalintojen tapauskohtaisuuden kautta (mm. Pinto 2013; Pereira ym. 2016; Sahin, Pollock ja Clause 2014). Lisäksi mobiililaitteiden ohjelmistopinot kehittyvät nopeasti ja niiden keskinäiset erot ovat aina ohjelmointikielistä alkaen merkittäviä. Näin on mahdollista, että yleisesti hyödylliset käytänteet eivät yleensä ole kääntäjäoptimointien kaltaisia, granulariteetiltaan hyvin yksityiskohtaisia tai arkkitehtuurimallien tavoin korkealla tasolla, vaan sijoittuvat näiden välille.

Tähän lukuun on pyritty keräämään tällaisia käytänteitä kattavasti ja myös niissä tapauksissa, joissa ne on esitetty menetelmän tai sen toteutuksen osana. Käytänteiden enemmistöä ei ole testattu siirrettävyyden osalta esimerkiksi toisilla alustoilla, joten niitä ei ole jätetty tämänkään kokonaisuuden ulkopuolelle. Käsittelyn ulkopuolelle on kuitenkin jätetty muun muassa versiohallintajärjestelmistä kommenttien avulla tunnistetuista käytänteistä ne, joita ei ole muotoiltu yleiseksi ratkaisuksi (esim. Bao ym. 2016). Lisäksi pois on rajattu sellaiset käytänteet, joita ei ole mahdollista hyödyntää käyttäjäsovellusten tasolla. Käsiteltävistä käytänteistä monet sisältyvät myös jo mobiililaitevalmistajien uusimpiin ohjelmointioppaisiin, tosin usein suorituskykyoptimointeina ja ilman arvioita mahdollisista energiankulutusvaikutuksista (esim. “Performance Tips | Android Developers” 2016; “Performance Tips | App programming guide for iOS” 2016).

5.1 Yleiset ohjelmointikäytänteet

IO-laitteiden käyttö kuluttaa useimpia muita käskyjä enemmän energiaa ja siihen liittyy staattista kulutusta, johon suoritettavan työn määrä ei vaikuta. Niinpä näiden laitteiden hyötysuhdetta on mahdollista parantaa kokoamalla peräkkäin, samaan aktivaatioon, useampia operaatioita. Jos esimerkiksi tiedostoja tarvitaan todennäköisesti lisää tulevaisuudessa, niiden lataaminen kerralla keskusmuistiin säästä energiaa toistuviin, lyhyempiin latauksiin verrattuna (J. Wang ym. 2012). Esiladattava tieto on kuitenkin rajoitettava ainoastaan todennäköisesti tarpeelliseen, tai sen käsittely voi heikentää lopullista energiankulutusta ja suorituskykyä (Metri, Shi ja Brockmeyer 2015). Kiinteän flash-muistin ja muistikortin energiankulutus on suhteessa muuhun laitteistoon usein niin matala (Carroll ja Heiser 2013), että sen käytön tehostamisen merkitys korostuu kuitenkin vasta usein toistuvissa operaatioissa.

Sovellus joutuu usein odottamana laitteiston, ympäristön tai käyttäjän toimintoja. Muutoksiin on mahdollista reagoida aktiivisesti tilaa seuraamalla, pollaamalla, mutta yleensä myös tapahtumaperusteisesti. Pollaaminen pitää suoritinta aktiivisena ja estää energiansäästötilojen käytön, varaa muiden prosessien mahdollista aikaa ja lisää energiankulutusta seurattavissa laitteistokomponenteissa. Lisäksi pollaamisen energiankulutus kasvaa tarkkuuden mukana, joten sitä on yritettävä välttää (Pinto, Castor ja Liu 2014). Jos tapahtumat ovat harvinaisia tai sovelluksen toimintaa on muuten viivästettävä pidempiä aikoja, ohjelmointikielten tavalliset, tarkat ajastimet eivät yleensä ole energiatehokas vaihtoehto. Tätä käyttökelpoisempi rakenne ovat epätarkat ajastimet, joiden aikana suoritin voi pysyä lepotilassa (Datta, Bonnet ja Nikaein 2013; "AlarmManager | Android Developers" 2016). Jos odotusaikana on mahdollista suorittaa muita operaatioita, ne kannattaa sijoittaa omiin säikeisiinsä ja pyrkiä näin nopeuttamaan tehtävien valmistumista. Suorittimen lyhyemmät, mutta intensiivisemmät kuormitusjaksot lisäävät lepoaikaa ja samalla säästävät usein myös energiaa (Pinto, Castor ja Liu 2014).

Osa sovelluksista käsittelee ja vaihtaa ympäristön kanssa paljon tietoa. REST-rajapinnoissa vakiintuneita viestiformaatteja ovat muun muassa JSON ja XML, mutta myös muita formaatteja, kuten Googlen *Procol Buffers*, on otettu käyttöön ("Protocol Buffers | Google Developers" 2016). Saman tiedon koko eri esitysmuodoissa vaihtelee merkittävästi, mikä vaikuttaa niiden käsittelemisen, tallentamisen ja siirtämisen energiankulutukseen. Tavallisesti energiatehokkaimpana vaihtoehtona on pidetty tiiviin lopputuloksen tuottavaa JSON:ia, jota käsitellään virtatyyppisellä parserilla (Datta, Bonnet ja Nikaein 2013; Gil ja Trezentos 2011). Lisäksi sarjallistetut tietueet kannattaa pakata ennen lähettämistä esimerkiksi GZip:illä, koska verkkoliikenteessä säästettävä energia on yleensä pakkaamiseen kuluvaa suurempi. Pienten viestien tai tekstipohjaisia tiiviimpien binääriformaattien kanssa pakkaaminen ei kuitenkaan välttämättä säästä energiaa (Gil ja Trezentos 2011).

Tietoa voidaan tallentaa paikallisesti tiedostoihin, mutta monet alustoista tarjoavat niille vaihtoehtona myös sovelluskohtaisen relaatiotietokannan, johon tietue-avain-parien tallentaminen on yksinkertaista. Relaatiotietokannan operaatiot kuluttavat kuitenkin suhteessa paljon energiaa ja sen hallintaan, kuten avaamiseen ja sulkemiseen, sekä kokonaisiin tauluihin kohdistuvat operaatiot ovat erityisen kalliita. Jos relaatiokannan tietomallia ei tarvita tai

operaatioita on tarpeen suorittaa paljon, suositellaan sen korvaamista yksinkertaisemmalla tietorakenteella, joka sarjallistetaan tarvittaessa tiedostoon (Linares-Vasquez ym. 2014).

Vastaavaa selaimessa suoritettavaa ja alustalla suoraan suoritettavaa sovellusta verrattaessa on selainsovelluksen energiankulutus todennäköisesti suurempi (Metri, Shi ja Brockmeyer 2015). Energiatehokkuutta tavoiteltaessa sovellukset voidaan toteuttaa kokonaan alustakohtaisesti tai käyttää alustariippumatonta kehitystä tukevia ympäristöjä, jotka tuottavat samasta projektista julkaisuversion useammalle alustalle (esim. “Xamarin” 2016). Osalla mobiililaitteista on myös mahdollista suorittaa useammalla ohjelmointikielellä toteutettuja sovelluksia. Androidin tapauksessa mahdolliset kielet ovat Java ja C/C++, joista Javan suorittaminen virtuaalikoneella laskee selvästi sen suhteellista energiatehokkuutta (Corral ym. 2014). Natiivikoodia ei kuitenkaan suoritusympäristön rajoitusten takia suositella suorituskyvyn parantamiseen ja nykyään Android-alusta kääntää Java-ohjelmat asennusvaiheessa natiivibinääreiksi (“ART and Dalvik | Android Open Source Project” 2016; “Performance Tips | Android Developers” 2016). Tämä muutos on todennäköisesti tuonut Javan ja C:n energiankulutuksen selvästi lähemmäksi toisiaan.

Yksittäisten olioiden luominen, tuhoaminen ja roskienkeruu kuluttavat vähän resursseja ja energiaa, mutta niiden vaikutuksen merkitys korostuu mobiililaitteilla (Bhattacharya ym. 2012; Datta, Bonnet ja Nikaein 2013). Olioiden lukumäärää kannattaa rajoittaa käyttämällä primitiivityyppejä ja niiden taulukoita yksinkertaisten koosteolioiden sijasta. Monimutkaisesta alustajasta riippuvia olioita on usein mahdollista kierrättää ja lyhytikäisiä väliaikaisolioita välttää (Datta, Bonnet ja Nikaein 2013). Lisäksi sopivat argumentti- ja paluuarvotyypit valitsemalla olioiden kopioimista ja tyyppimuutoksia esimerkiksi merkkijonojen käsittelyssä on mahdollista vähentää (“Performance Tips | Android Developers” 2016).

Granulariteetiltaan yksityiskohtaisempina, energiaa säästävinä keinoina on esitetty myös liukulukulaskennan korvaamista kokonaislukulaskennalla ja tulosten muuntamista tarvittaessa takaisin liukuluvuiksi (Datta, Bonnet ja Nikaein 2013). Staattisten metodien kutsuminen Javassa on virtuaalisia yksinkertaisempaa ja kuluttaa vähemmän energiaa. Näin virtuaaliset metodit kannattaa muuttaa mahdollisuuksien mukaan staattisiksi. Olioiden tiedon käsittelemiseen käytettävät välittäjämetodit ovat myös virtuaalisia ja lisäävät energiankulutusta muuttujien suoraan käyttöön verrattuna, joten niiden poistaminen voi osassa tapauksista ol-

la perusteltua tiedon suojaamisen heikkenemisestä huolimatta (“Performance Tips | Android Developers” 2016; Li ja Halfond 2014). Lisäksi Javan taulukoiden pituusmuuttujan käyttäminen silmukan loppuehtona johtaa sen tarkistamiseen jokaisella kierroksella ja kuluttaa selvästi ylimääristä energiaa. Jos pituus ei muutu, se kannattaa tallentaa ehtona käytettävään väliaikaismuuttujaan ennen toistorakenteen alkua (Li ja Halfond 2014). Monet alustat tarjoavat sovelluksille myös mahdollisuuden estää puhelimen automaattinen siirtymä lepotilaan, jos niiden tarvitsee säilyä taustalla aktiivisena (mm. “idleTimerDisabled - UIApplication | Apple Developer Documentation” 2016; “Keeping the Device Awake | Android Developers” 2016). Tämä mekanismi voi lisätä ylimääräistä kulutusta, jos siirtymä estetään tarpeettomasti tai estoa ei muisteta vapauttaa (Pathak ym. 2012).

5.2 Käyttöliittymään liittyvät käytänteet

Käyttöliittymissä tehdyt valinnat voivat vaikuttaa sovellusten energiankulutukseen välillisesti monilla tavoilla, mutta niillä on siihen myös suora vaikutus. Mobiililaitteissa näytöt ovat yksi eniten energiaa kuluttavista laitteistokomponenteista ja useimmat sovellukset käyttävätkin suuren osan ajasta ja jopa 60 % energiastaan lepotilassa, suorittamatta aktiivisesti koodia (D. Li ym. 2014). Näytön asetuksilla ja kuvasisällöllä voi näin olla muun toteutuksen tasoinen vaikutus sovelluksen energiankulutukseen. Mobiililaitteissa käytetään kuitenkin kahta, toimintaperiaatteeltaan täysin erilaista näyttötekniikkaa. Näistä LCD-näytöissä taustavalo kuluttaa suurimman osan energiasta ja tuottaa valkoista valoa tasaisesti, lähes kuvasisällöstä riippumatta. Sen edessä on nestekidekerros, joka tuottaa taustavaloa moduloimalla näkyvän kuvan ja kuluttaa hyvin vähän energiaa (Kawamoto 2002). OLED-näytöissä pikselit tuottavat puolestaan itse valoa, joten näytön energiankulutus riippuu käytännössä kokonaan sen esittämästä kuvasisällöstä (Mittal, Kansal ja Chandra 2012).

LCD-näytöille yksinkertainen suositus on vähentää taustavalon kirkkautta ja aikaa, jonka se on päällä (Datta, Bonnet ja Nikaein 2013). Näytön kirkkauden vähentäminen heikentää luettavuutta hyvässä valaistuksessa, ja yleiskäyttöisempänä ratkaisuna on ehdotettu kuvasisällön kirkkauden lisäämistä ja taustavalon himmentämistä. Näiden tekijöiden yhteisvaikutuksesta kuvan kontrasti vähenee, mutta sen havaittu kirkkaus ja siten luettavuus säilyy lähellä alkuperäistä tasoa, vaikka energiankulutusta voidaan selvästi laskea. Kuvasisällön kirkkautta

voidaan kuitenkin käsitellä energiatehokkaasti lähinnä GPU:n gamma-muunnoksella, joka ei yleensä ole sovellettavissa tavallisissa, käyttäjätasoisissa sovelluksissa (Anand ym. 2011). Periaatetta on silti mahdollista hyödyntää valitsemalla käyttöliittymän komponenteille vaaleat värit ja korkea kontrasti, jotka vähentävät hyvään näkyvyyteen riittävää taustavalon kirkkautta.

OLED-näyttöjen energiankulutus riippuu sekä kuvan kirkkaudesta että sen värisisällöstä. Kirkkaat värit kuluttavat tummia enemmän ja osaväreistä sinisen vaikutus on muita suurempi. Näin energiankulutusta voidaan vähentää sekä kuvan kirkkautta laskemalla että säätämällä sen värisisältöä (esim. Dong, Choi ja Zhong 2009; K. W. Tan ym. 2013). Koko kuva-alan kirkkauden laskeminen heikentää helposti kuvan laatua niin merkittävästi, että sen sijasta on suositeltu vain vähemmän merkityksellisten osien tummentamista (K. W. Tan ym. 2013). Lisäksi käyttöliittymän elementtien väreissä voidaan suosia tummia, vähän sinistä sisältäviä sävyjä (Dong, Choi ja Zhong 2009).

Molempien optimointien yhteinen ongelma on, että ollessaan tehokkaita ne myös vaikuttavat sovelluksen ulkoasuun suhteellisen paljon. Lisäksi eri näyttötekniikoille sopivat ratkaisut ovat käytännössä vastakkaisia, joka rajoittaa niistä saatavan edun aina vain osalle käyttäjistä. Kattavin ratkaisu on toteuttaa molemmat vaihtoehdot ja päätellä näyttötekniikka asennusvaiheessa laitteen mallinumerosta. Jos käyttöliittymän ulkoasuratkaisuja ei haluta tehdä kokonaan energiansäästön tavoitteiden pohjalta, yksi vaihtoehto on myös käyttää energiatehokasta ulkoasua ainoastaan akun ollessa lähes tyhjä (K. W. Tan ym. 2013).

5.3 Verkkoliikenteeseen liittyvät käytänteet

Radiorajapintojen vaikutus hallitsee sovellusten aktiivisen suoritusajan energiankulutusta. Analysoitaessa kattavaa otantaa Android-sovelluksista todettiin, että pelkästään HTTP API-kutsujen osuus energiasta on kolmessa neljästä sovelluksesta 89 % tai enemmän (D. Li ym. 2014), joten käytänteillä voi olla merkittävä kokonaisvaikutus. Eri verkkotekniikoilla on erilainen energiakulutusprofiili, mutta tärkeimpänä yleisratkaisuna on pidetty muun IO:n tavoin useampien pyyntöjen suorittamista kerralla, joka säästää energiaa vähentämällä rajapinnan käyttöä seuraavien aktivaatiojaksojen lukumäärää (mm. J. Wang ym. 2012; Liu, Zhang ja

Zhou 2011; Balasubramanian, Balasubramanian ja Venkataramani 2009). Yhdessä viestissä käsiteltävän datan määrää on myös hyödyllistä kasvattaa, koska esimerkiksi yhden tavun ja yhden kilon HTTP-pyynnöt kuluttavat saman määrän energiaa alempien tasojen protokollakerrosten kiinteiden kustannusten takia (Li ja Halfond 2014). Pelkkien lataus- tai lähetystapahtumien yhdistämisen ohella on suositeltu myös latausten suorittamista yhtä aikaa lähetysten kanssa, koska niiden energiankulutus on jo valmiiksi vastaanottoa korkeampi, eikä rinnalla tapahtuva vastaanotto enää merkittävästi lisää tätä tasoa (Ou ym. 2013). Lisäksi HTTPS-protokollan korvaaminen HTTP-protokollalla voi säästää pienen osuuden verkkoliikenteen tarvitsemasta energiasta (Naylor ym. 2014).

Jokainen verkkotekniikka kuluttaa eri määrän energiaa yhden tavun siirtämiseen, mutta vertailua monimutkaistavat eri pituiset, käyttöä seuraavat aktivaatiojaksot ja kuormitustason kasvaessa muuttuvat kustannukset (Huang ym. 2012; Balasubramanian, Balasubramanian ja Venkataramani 2009). WWAN-tekniikoista GPRS voi olla energiatehokkain ladattaessa pieniä, alle 100 kt viestejä, mutta häviää tätä selvästi suuremmilla 3G/HSPA:lle (Balasubramanian, Balasubramanian ja Venkataramani 2009). LTE-tekniikka on vaihtoehdoista suorituskykyisin, mutta siihen liittyy muita enemmän käytön jälkeistä aktivaatiota, joten keskimääräisissä sovelluksissa se kuluttaa energiaa HSPA:ta enemmän. Suurempia, useiden satojen kilojen yhtenäisiä tietomääriä siirrettäessä LTE on WWAN-tekniikoista energiatehokkain (Huang ym. 2012). Kaikista vaihtoehdoista energiatehokkaimpia ovat kuitenkin WLAN-tekniikat, jotka voivat kuluttaa todellisissa sovelluksissa vain alle kymmenesosan WWAN-tekniikoiden tarvitsemasta energiasta (Huang ym. 2012; Balasubramanian, Balasubramanian ja Venkataramani 2009).

Käyttäjätasoisissa mobiilisovelluksissa ei tavallisesti ole mahdollisuutta hyödyntää WWAN-tekniikoiden eroja energiankulutuksen optimointiin, ei edes muita avoimemmalla Android-alustalla (“TelephonyManager.java - Git at Google” 2016; “Manifest.permission | Android Developers” 2016). WLAN-yhteyden tilan seuraaminen ja käyttöönotto on kuitenkin usein sallittu tavallisille sovelluksille. Kiireettömän tiedonsiirron tapauksessa on hyödyllistä odottaa WLAN-verkon käyttömahdollisuutta, tai ainakin viivästää tiedonsiirtoa ja käyttää muita tekniikoita vasta aikarajan umpeutuessa (Ra ym. 2010).

5.4 Sensoreihin liittyvät käytänteet

Sensoreiden energiankulutuksenhallinnan yleisohje on käyttää niitä matalimmalla riittävällä päivitysnopeudella ja sammuttaa ne heti, kun dataa ei enää käytetä (mm. “Performance Tips | App programming guide for iOS” 2016; Liu, Xu ja Cheung 2013). Sensoreista useimmat, kuten kiihtyvyysanturi tai magnetometri, kuluttavat itse erittäin vähän energiaa ja valtaosa niiden vaikutuksesta johtuu dataa sisältäviä tapahtumia vastaanottavan ja käsittelevän suorittimen energiankulutuksesta (Carroll ja Heiser 2013). Näin niiden käytöstä on annettu vain vähän yksityiskohtaisempaa ohjeistusta. GPS-vastaanotin erottuu kuitenkin muista sensoreista selvästi suuremmalla energiankulutuksellaan, mutta perusohje päivitysnopeuden minimoimisesta pätee myös siihen (Chon ym. 2011). Jos matalalla päivitysnopeudella saatavien paikkatietojen tarkkuus riittää sovellukseen, on mahdollista, että GPS-sijainti on korvattavissa myös selvästi energiatehokkaammalla, radioverkkoihin perustuvalla paikannuksella (Paek, Kim ja Govindan 2010).

Pitkäaikaisesti kerättävien paikkatietojen sovelluskohteet ovat ajaneet kehittämään myös vaihtoehtoja pelkälle GPS-vastaanottimen päivitysnopeuden laskemiselle, joka rajoittaa tietojen tarkkuutta liikenopeudesta riippuvalla tavalla (Paek, Kim ja Govindan 2010). Vaihtoehdoista ensimmäinen on ollut GPS-sijainnin päivitystarpeen arvioiminen muiden sensoreiden, erityisesti kiihtyvyysanturin, avulla. Kiihtyvyysdatasta voidaan arvioida liikkuuko laite ja rajallisesti myös sen liikenopeus (Paek, Kim ja Govindan 2010; D. H. Kim ym. 2010). Liikkumisen tunnistamista monimutkaisempi arviointi vaatii kuitenkin yleensä niin korkeaa näytenopeutta, että se rajoittaa mahdollisuutta säästää energiaa harvoin kerättyyn GPS-sijaintiin verrattuna (Paek, Kim ja Govindan 2010). Toinen GPS-sijainnin päivitysvälin arviointivaihtoehto on käyttää historiatietoja ja yrittää ennustaa millaista laitteen liike voi tulevaisuudessa olla, tai milloin laite seuraavan kerran todennäköisesti liikkuu (Chon ym. 2011; Paek, Kim ja Govindan 2010; D. H. Kim ym. 2010). Historiatiedot soveltuvat kuitenkin vain niihin sovelluskohteisiin, joissa liikkeiden voidaan odottaa olevan luonteeltaan toistuvia. Osalla alustoista paikannuskirjastot ovat jo yhdistäneet verkkopaikannuksen, GPS-paikannuksen ja muita sensoreita toisiaan tukevaksi kokonaisuudeksi, joka mahdollistaa valitun kompromissin tarkkuuden ja energiankulutuksen välillä ilman sensoreita yhdistelevien menetelmien kehittämistä (esim. “Making Your App Location-Aware | Android Developers” 2016).

6 Tutkimusmenetelmä

Empiirisessä osassa on sovellettu konstruktiivista, suunnittelutieteellistä tutkimusmenetelmää. Tässä menetelmässä olennaista on tekniikka, joka nähdään ensisijaisesti toiminnan mahdollistajana ja apuvälineenä (Hevner ym. 2004). Monista muista menetelmistä poiketen tekniikkaa käsitellään tavoitteena ja tutkimuksen tuloksena, ei kuvailun tai ennustuksen kohteena. Niinpä tekniikan käsittely ei rajoitu teoreettisen tiedon muodostamiseen, vaan menetelmän tavoite on tiedon soveltaminen (March ja Smith 1995). Suunnittelutiede on siten vahvasti ongelmanratkaisumenetelmä, jonka tavoite on toteuttaa relevantteja ongelmia ratkaisevia artefakteja (Hevner ym. 2004). Näin sen tärkeimpiä kysymyksiä ovat toteutettujen artefaktien sovellettavuus ja tehokkuus, mutta ei niinkään niiden toiminnan selittäminen ympäristössä (March ja Smith 1995; Hevner ym. 2004).

Kaikki teknisten ongelmien ratkaiseminen ei kuitenkaan ole suunnittelutieteellistä tutkimusta. Siihen kuulumattomina rutiinitapauksina on nähty tilanteet, joissa on saatavilla yleisesti tunnettuja hyviä käytänteitä, joita voidaan soveltaa sellaisenaan (Hevner ym. 2004). Suunnittelutieteellisen tutkimuksen tavoitteet puolestaan liittyvät kokonaan uusien ongelmien ratkaisemiseen tai aikaisempien toteutusten kehittämiseen. Tämä tuottaa tietoa sekä tuloksena kehitetyn artefaktin että siihen johtaneen prosessin muodossa (Hevner ym. 2004).

Tutkimuksen tuloksiksi on laajasti nähty kaikki tutkimustyötä suunnittelussaan sisältävät toteutukset. Tarkemmin on tunnistettu neljä tyypillistä, abstraktiotasoltaan erilaista artefaktityyppiä, jotka ovat käsitteistöt, mallit, metodit ja realisaatiot (March ja Smith 1995; Hevner ym. 2004; Peffers ym. 2007). Näistä käsitemallit mahdollistavat kohdealueen kuvaamisen ja tukevat siihen liittyvää ajattelua ja viestimistä, mallit kuvaavat ongelmaa ja sen ympäristöä sekä ratkaisuja, metodit antavat askelittaisia ohjeita ratkaisujen saavuttamiseksi ja realisaatiot ovat edellisiä sisältäviä toteutuksia ympäristössään (March ja Smith 1995). Realisaatioihin kuuluvat valmiit tietojärjestelmät, mutta myös prototyypit ja osatoteutukset (Hevner ym. 2004).

Suunnittelutieteellisen tutkimuksen tärkeänä ulottuvuutena on pidetty suunnittelun tunnistamista prosessiksi ja tulokseksi. Näin siihen kuuluva iteratiivinen toimintatapa, jossa toteutusta arvioidaan sen edetessä, tuottaa palautetta sekä tuloksen että prosessin kehittämisen tueksi (Hevner ym. 2004). Toteuttamista edeltää ongelman tunnistaminen ja sitä seuraa tulosten toiminnallisuuden osoittaminen. Kattavammin tutkimusprosessin tärkeimpiä vaiheita on muotoiltu muun muassa eräässä jaossa kuuteen askeleeseen (Peffers ym. 2007). Näitä ovat ongelman tunnistaminen ja oikeuttaminen, ratkaisun tavoitteiden määritteleminen, suunnittelu ja toteuttaminen, käyttökelpoisuuden osoittaminen, tuloksen arviointi ja tulosten raportoiminen. Vaiheet eivät ole puhtaasti sekventiaalisia, ja niiden järjestys voi muuttua työn alkaessa esimerkiksi valmiista artefaktista (Peffers ym. 2007). Niistä kukin riippuu jo vakiintuneista menetelmistä, kuten vaatimusmäärittelystä, analyysimenetelmistä ja testausmenetelmistä.

Tämän työn teoriaosuudessa on käsitelty energiankulutuksen määrittämiseen ja energiatehokkaaseen ohjelmointiin liittyviä käytänteitä, luonteeltaan metodeja, joista osasta on julkaistu myös esimerkkitoteutuksessa realisaatio. Empiirisessä osassa joitain kootuista energiatehokkaan ohjelmoinnin käytänteistä sovelletaan reittitietojen keräämiseen ja siirtämiseen liittyvään osakokonaisuuteen, komponenttiin. Vaikka sovellettavat metodit näin ovat jo julkaistua tietoa, tehtävää työtä voidaan pitää suunnittelutieteellisenä tutkimuksena, koska kehitettävä realisaatio on uusi kokonaisuus. Tätä puoltaa uuden tavoitteen toteuttaminen, jota yksikään sen osana vaikuttavista metodeista ei kata ja myös se, että tarkasteltava suoritusympäristö on sovellettaville metodeille uusi. Empiirisen luvun rakenne perustuu suunnittelutieteelliselle tutkimukselle esitettyihin kuuteen vaiheeseen (Peffers ym. 2007), joista osa on yhdistetty saman alaluvun alle, eikä tulosten raportoimista käsitellä erillään.

7 Välineiden soveltaminen

Tässä luvussa perehdytään energiatehokkaan mobiiliohjelmoinnin välineiden soveltamiseen pienen Android-alustalle kehitettävän ohjelmakomponentin puitteissa. Alaluvuista ensimmäisessä motivoidaan komponentin tarvetta ja määritellään sen tavoitteet. Toisessa esitellään kehitetty, testauksessa käytettävä energiankulutuksen mittauslaitteisto. Kolmannessa kuvataan toteutusympäristö, toteutettu komponentti ja kehityksen yhteydessä suoritettu energiankulutukseen liittyvä testaus. Neljännessä käydään läpi kehitystulokset, tarkastellaan tavoitteiden toteutumista komponentissa ja arvioidaan testattujen energiatehokkaan ohjelmoinnin käytänteiden käyttökelpoisuutta tässä sovelluksessa.

7.1 Komponentin tavoitteet

Navigaatio-, kartoitus ja liikuntasovellukset riippuvat tarkoista paikkatiedoista, joita kerätään reittien muodostamiseksi suhteellisen korkealla päivitysnopeudella (D. H. Kim ym. 2010). Näistä sovelluksista monet tallentavat tietoja paikallisesti, mutta myös jakavat niitä reaaliaikaisesti esimerkiksi sosiaalisen vuorovaikutuksen tukemiseksi (esim. “Runkeeper Go” 2016). GPS-vastaanotin ja verkkoliikenne voivat molemmat kuluttaa paljon sähköenergiaa ja tällaisille sovelluksille ominaista on jatkuva, pitkäaikainen käyttö. Yhdessä nämä piirteet tekevät reittimuotoisten paikkatietojen käsittelystä energiankulutusvaikutukseltaan haasteellisen ja samalla myös mielenkiintoisen kehitys- ja mittauskohteen sekä tiedonsiirtoon että sensoreihin liittyvien käytänteiden testaamiseen.

Reittien käsitteleminen tapahtuu taustalla, erillään sovelluksen vuorovaikutteisista näkymistä, joten sen toiminnot on toteutettava niistä riippumattomana, palvelutyyppisenä komponenttina. Muita keskeisiä vaatimuksia ovat paikkatietojen kerääminen, tallentaminen ja lähettäminen palvelimelle. Komponentin mahdolliseksi sovellusalueeksi on valittu tässä tarkastelussa liikuntasuorituksia seuraavat ja maastossa liikkumiseen liittyvät sovellukset, jotka voivat hyötyä paikkatietojen lisäksi myös mobiililaitteiden muista sensoreista saatavista tiedoista. Kehitettävässä komponentissa näistä käsitellään barometrisen korkeuden ja keskimääräisen kiihtyvyyden seuraamista. Komponentissa toteutettavat vaatimukset, testattavat

toteutusvaihtoehdot ja sovellettavat ohjelmointikäytänteet on käsitelty alaluvussa 7.3.

7.2 Energiankulutuksen mittausjärjestely

Energiankulutuksen määrittämismenetelmäksi valittiin mobiililaitteen sähkötehon mittaaminen, koska se kertoo todellisesta kulutuksesta ja on yleisesti käytetty vertailukohta muille määritysmenetelmille (mm. Hao ym. 2012; Dong ja Zhong 2011). Osaltaan valintaan vaikutti myös vaihtoehtoisen määritysmenetelmien toteutusten erittäin heikko saatavuus. Mobiililaitteiden energiankulutuksen luotettava mittaaminen vaatii kuitenkin korkeaa, 100 Hz:n – 1 kHz:n näytenopeutta (Maker, Amirtharajah ja Akella 2013), jota tukevat kaupallisista mittalaitteista vain suhteellisen kalliit mallit (esim. “Monsoon Solution Power Monitor” 2016). Hyvän älypuhelimen hintainen tai sitä kalliimpi mittalaite ei ole useimmille aiheesta kiinnostuneista kehittäjistä realistinen vaihtoehto, joten osana empiiristä työtä päätettiin toteuttaa myös yksinkertainen ja edullinen prototyyppi mobiililaitteen sähkötehon mittaamiseen soveltuvasta laiteesta.

Sähkötehon mittaaminen toteutettiin akkujännitteen suoralla mittaamisella ja akkupiirin virran mittaamiseen käytettävällä mittausvastuksella. Näin saatavien signaalien käsittelemiseen voidaan käyttää tavallisia A/D-muuntimia, joita on saatavilla jopa valmiina, edullisina USB-liitentäisinä kortteina (mm. “8 Channel USB GPIO Module With Analog inputs” 2016). Yleiskäyttöisten muuntimien rajoitus on kuitenkin korkea muunnettava jännite, joka vaatii virranmittausvastukselta saatavan signaalin vahvistamista ja lisää samalla toteutukseen tarvittavien komponenttien lukumäärää (Rice ja Hay 2010). Toinen vaihtoehto ovat tehonmittauspiirit, jotka sisältävät sekä esivahvistimet että muuntimet samassa piirissä ja vaativat vain muutamia lisäkomponentteja. Toteutuksen yksinkertaistamiseksi mittalaitteeseen valittiin INA219-tehonmittauspiiri, joka sisältää 12 bitin A/D-muuntimet jännitteen ja virran mittaamiseen 1 kHz:n näytenopeudella ilman erillisiä mittausvahvistimia. Lisäksi tämän piirin sisäinen näytteenottonopeus on 500 kHz, joten tulokset kattavat myös mobiililaitteiden lyhytkestoisimmat energiankulutustapahtumat (“INA219 Datasheet - Texas Instruments” 2016; Maker, Amirtharajah ja Akella 2013).

Toteutukseen valittu tehonmittauspiiri lähettää mittausnäytteet I2C-väylän välityksellä lait-

teen pohjana toimivalle Atmel ATMega88 -mikro-ohjaimelle, jolle kehitettiin C-kielinen ohjelma tulosten esikäsittelyyn ja tietokoneelle lähettämiseen (liite B). Mikro-ohjaimen A/D-muuntimen avulla mittalaitteeseen lisättiin myös radiotaajuisen signaalinvoimakkuuden mittaus mobiililaitteen lähetysjaksojen tunnistamiseksi (kytkentälaavio, liite A). Lisäksi mittausten tallentamiseen, energian kokonaiskulutuksen analysointiin ja näytteiden visualisointiin kehitettiin Java-ohjelma. Teknisiltä ominaisuuksiltaan ja valmistuskustannuksiltaan käytännössä vastaava mittalaite on mahdollista toteuttaa myös valmiista moduuleista esimerkiksi Arduino-alustan pohjalta (“INA219 High Side DC Current Sensor Breakout” 2016; “Arduino Uno” 2016; Hindle ym. 2014). Laitteiston komponenttitasoinen toteutustapa valittiin lähinnä sen suuremman muunneltavuuden takia, mutta se antoi myös hyvän mahdollisuuden tutustua tarkemmin mikro-ohjaimiin. Useimmille ohjelmistokehittäjille käytännöllisin ja käyttöönottokynnykseltään matalin vaihtoehto ovat todennäköisesti valmiisiin moduuleihin perustuvat ratkaisut, joilla vältetään piirilevyjen valmistaminen tai tilaaminen. Lisäksi näille alustoille on olemassa valmiita kirjastoja, joilla mittalaite on erittäin yksinkertaista toteuttaa (“Arduino - Libraries” 2016; “INA219 High Side DC Current Sensor Breakout” 2016).

Mittalaitteessa käytetty INA219-tehonmittauspiiri ja virranmittausvastus määräävät saavutettavan mittausvirheen. Testit suoritettiin puhelimella, jonka virranmittausvastus oli toleranssiltaan 0,5 % ja huomioimalla myös INA219:n syötön poikkeamajännite, CMRR ja PSRR, saadaan huoneenlämpötilassa mittauksen maksimivirheeksi 350 mA:n virralla 2 % (“A Current Sensing tutorial - Part III: Accuracy” 2016; “INA219 Datasheet - Texas Instruments” 2016; “LVK Series - Ohmite” 2016). Virhearvioinnin virtatasona käytettiin keskiarvoa Samsung Galaxy S5 -puhelimella kerätystä 100 sekunnin testijaksosta, jossa ladattiin sekä selattiin verkkosivuja HSPA-yhteydellä ja näytön kirkkaus oli 15 %. Tämä virta vastaa hyvin aikaisemmista, vastaavantyyppisistä puhelimista mitattua tasoa (esim Carroll ja Heiser 2013). Laitteen mittaustarkkuus heikkenee virran laskiessa, mutta maksimivirhe on 90 mA:n virralla, joka vastaa idle-tilaa näyttö päällä, edelleen alle 6 % (“A Current Sensing tutorial - Part III: Accuracy” 2016; “INA219 Datasheet - Texas Instruments” 2016).

7.3 Komponentin toteuttaminen ja testaaminen

Komponentti toteutettiin Java-kielellä Androidin Service-luokka perimällä ja sitä suoritetaan sen pitkäaikaisen säilymisen varmistamiseksi etualalla (startForeground) (“Service | Android Developers” 2016). Lisäksi komponentin ympärille toteutettiin sovellus, jossa on yksinkertainen käyttöliittymä testien suorittamiseen. Sovelluksen kehittämiseen käytettiin Android Studio -kehitysympäristön versiota 2.2.3 ja se toteutettiin tukemaan API-versiota 15 tai sitä uudempaa käyttäviä laitteita. Sovelluksen ja komponentin lähdekoodit on listattu liitteessä E. Komponentissa toteutetut vaatimukset olivat seuraavat:

1. GPS-paikkatieto
    (a) kerätään vähintään 5 sekunnin välein
    (b) aikaleimataan
2. muut sensorit
    (a) ilmanpaine korkeusmääritykseen kerätään vähintään 2 sekunnin välein
    (b) kiihtyvyysarvo kerätään vähintään sekunnin välein
    (c) data aikaleimataan
3. tietojen lähettäminen
    (a) lähetetään korkeintaan puolen minuutin viiveellä palvelimelle
4. tietojen tallentaminen
    (a) paikkatiedot tallennetaan paikallisesti
    (b) sensoridata tallennetaan paikallisesti
    (c) tallennus suoritetaan korkeintaan 10 sekunnin viiveellä.

Nämä vaatimukset toteuttavan komponentin ja sille suoritettavan testauksen kautta pyrittiin varmistamaan seuraavien käytänteiden tai yleisten periaatteiden paikkansapitävyyttä ja merkitystä tässä sovelluksessa:

1. GPS-vastaanottimen näytenopeuden laskeminen säästää selvästi energiaa (Paek, Kim ja Govindan 2010).
2. Sensoreiden näytenopeuden laskeminen säästää energiaa (Liu, Xu ja Cheung 2013).

3. Sensorikuuntelijat kannattaa poistaa kun tietoa ei enää tarvita (Liu, Xu ja Cheung 2013). Sama periaate voi säästää energiaa myös tarvittua nopeammin tapahtumia tuottavilla sensoreilla, kun kuuntelija poistetaan säännöllisesti nopeuden rajoittamiseksi.
4. Tiedosto-operaatiot kannattaa koota suuremmiksi kokonaisuuksiksi (J. Wang ym. 2012).
5. Verkkoliikennettä kannattaa viivästää ja yhdistellä suuremmiksi kokonaisuuksiksi (Balasubramanian, Balasubramanian ja Venkataramani 2009).
6. JSON on XML:ää tiiviimpänä myös energiatehokkaampi tiedonsiirtoformaatti (Gil ja Trezentos 2011).
7. Verkossa lähetettävien tietojen pakkaaminen säästää energiaa (Gil ja Trezentos 2011).
8. HTTP voi olla HTTPS:ää energiatehokkaampi tiedonsiirtoprotokolla (Naylor ym. 2014).
9. Laitteen lepotilan estäminen lisää merkittävästi sen energiankulutusta (Pathak ym. 2012).

Kysymyksiin vastaamiseksi kunkin vaatimuksen toteuttamiseen testattiin useampia vaihtoehtoja, joiden energiankulutusta verrattiin niiden toimiessa osana komponenttia. Tämä lähestymistapa valittiin, koska se varmistaa edustavan testausympäristön ja käsittelee kaikki vuorovaikutukset. Lisäksi osien vaihtaminen kokonaisuuteen on lähempänä tyypillistä ohjelmistokehityksen työnkulkua, kuin kattavien, irrallisten esitestien suorittaminen ja kustannusfunktioiden muodostaminen eri osille. Kokonaisuutena testaamisen riskinä on kuitenkin suhteelliselta vaikutukseltaan merkittävämpien osien energiankulutuksen satunnaisvaihtelu, joka voi peittää tarkasteltavat muutokset. Tämän riskin vähentämiseksi testausta suoritettiin riittävän pitkinä jaksoina ja useita kertoja jokaiselle erilaiselle konfiguraatiolle (esim. Chung, Lin ja King 2011; Rice ja Hay 2010). Lisäksi esitesteissä ilman komponentin muita funktioita mitattiin sensoreiden energiankulusta eri päivitysnopeuksilla.

Komponentissa testattavat, vaihtoehtoiset ratkaisut toteutettiin kaikki sen osiksi ja niiden valitseminen käytettiin muistiin ladattavaa konfiguraatiota, joka ohjaa ehtolauseita ja tehdasmetodeja. Nämä kontrollirakenteet vaikuttavat lopulliseen energiankulutukseen, mutta koska ne suoritetaan jokaiselle keskenään vertailtavalle vaihtoehdolle, ne eivät muuta niiden keskinäistä suhdetta. Toteutettu komponentti tukee näytetapahtumien vastaanottamista eri nopeuksilla GPS-vastaanottimelta, kiihtyvyysanturilta ja ilmanpaineanturilta. Nämä tapahtumat kirjoitetaan millisekuntiajan kanssa olioon ja tallennetaan jonoon, josta erillinen

sarjallistajasäie ottaa ne ja tuottaa niistä JSON, XLM tai merkkierotellun muodon. Saadut merkkijonot siirretään seuraaviin jonoihin, joista ne eri aikavälein tallennetaan tiedostoon paikalliselle flash-muistille ja lähetetään palvelimelle käyttäen HTTP(S) sovellusprotokollan POST-metodia tai pelkkiä TCP- tai UDP-kuljetusprotokollia. Sovellus lähettää vain datan, eikä käytä paluuviestintää ja UDP:n osalta siirtotien kadottamista paketeista ei välitetä.

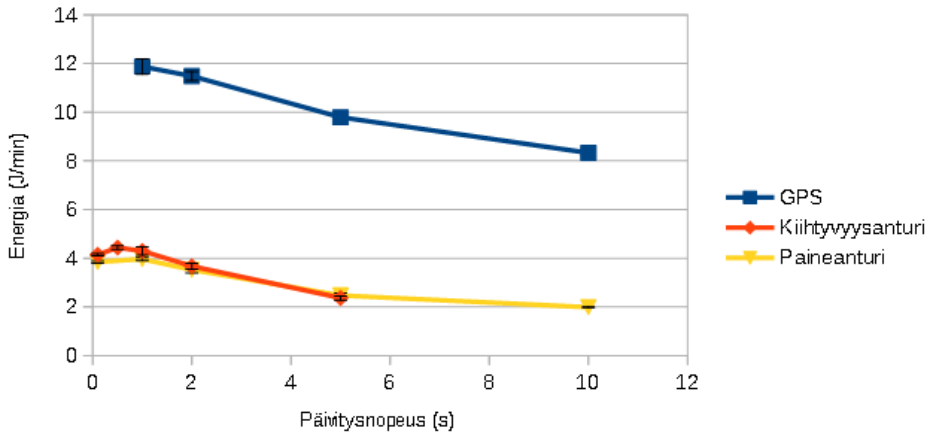
Komponentin energiankulutusta mitattiin Samsung Galaxy S5 -älypuhelimella (malli SM-G900F), jatkossa GS5. Laitetta ei kytketty testien aikana USB-kaapelilla tietokoneeseen, koska se lataa akkua. Mittausten aikana sovellusten ilmoitukset oli otettu pois käytöstä satunnaisvaihtelun vähentämiseksi. Lisäksi verkkoyhteydestä riippumattomat testit suoritettiin lentokonetilassa ja WLAN pidettiin aina pois käytöstä. Verkkoyhteytenä käytettiin 3G HSPA:ta Elisan verkossa hyvällä signaalinvoimakkuudella (keskiarvo -85 dBm). Laitteen akku pidettiin kaikissa testeissä ladattuna vähintään neljään volttiin, eli 75 %:n varaustilaan. Näyttö ei ollut testien aikana päällä, sen automaattinen kierto oli kytketty pois ja mitkään energiansäästötilat eivät olleet aktiivisia.

Kaikki mittaukset toistettiin viisi kertaa ja tuloksena käytetään niiden keskiarvoa. Toistojen välissä sovellus suljettiin ja käynnistettiin uudestaan. Tehonkulutusnäytteitä kerättiin niin, että hitain sovelluksessa toistuva tapahtuma ehtii sisältyä niihin kymmenen kertaa, jolloin puolen minuutin toistovälillä tapahtumia sisältävää konfiguraatiota mitattiin yli 5 minuuttia. Tämä vastaa aiemmissa tutkimuksissa usein käytettyä kymmenen toistokerran tasoa (esim. McCullough ym. 2011). Kahdessa testissä, joissa palvelinlähetysten toistotiheys oli 60 tai 120 sekuntia, kerättiin kuitenkin vain viisi toistoa kattavat 300 ja 600 sekunnin kestoiset mittaukset. Lisäksi ennen jokaista mittausta sovellusta suoritettiin yksi minuutti.

Edustavaa analysoitavaa mittausjaksoa ei voitu aina valita kerätystä datasta systemaattisesti, esimerkiksi minuutti jakson alusta, koska useimmissa testeissä mittauksiin sisältyi satunnaisia tapahtumia, jotka vaikuttavat olevan sovelluksesta riippumattomia ja energialtaan selvästi keskikulutusta suurempia. Erityisesti sensoreiden esitesteissä, joissa keskikulutus oli vain 1 – 14 J/min, aiheuttivat joulen sekunnissa kuluttavat tapahtumat merkittävän satunnaisvaihtelun rinnakkaisten mittaustulosten välille. Tällaiset tapahtumat poistamalla vaihtelu rinnakkaisten tulosten välillä oli suhteellisen vähäistä ja aiemmissa töissä ne on liitetty roskienkeruuseen ja muihin suoritusympäristön prosesseihin (esim. Hao ym. 2013).

7.4 Tulokset ja johtopäätökset

Käytänteiden testaamiseksi muusta komponentista erillään mitattiin kiihtyvyysanturin, paineanturin ja GPS-vastaanottimen lukemista ja näytteiden kirjoittamista tiedostoon eri nopeuksilla. Ehdotettu GPS-vastaanottimen ja sensoreiden päivitysnopeuden rajoittaminen todettiin toimivaksi, sillä testattujen sensoreiden energiankulutus riippui näytenopeudesta lähes lineaarisesti (kaavio 2). Testauksessa käytetyssä GS5-puhelimessa kiihtyvyys- ja paineanturin matalin tuettu päivitysnopeus oli noin 200 ms, joten sitä rajoitettiin poistamalla sensorikuuntelija käytöstä hitaammilla tavoitenopeuksilla. Rekisteröinnin poistaminen kuluttaa energiaa, joka selittää kaaviossa sekuntia lyhyemmillä, mutta 100 millisekuntia hitaammilla päivitysnopeuksilla näkyvän mutkan. Vertaamalla kuuntelijan poistamista ja tarpeettomien näytteiden hylkäämistä todettiin, että sekunnin päivitysnopeudella kuuntelijaa ei kannata poistaa käytöstä tai sillä ei ole merkitystä, mutta sitä pidemmillä ajoilla sen poistaminen säästää energiaa. Kymmenen sekunnin päivitysnopeudella ylimääräisten tapahtumien vaikutus jopa kaksinkertaisti tarvittavan energian, joten kuuntelija kannattaa poistaa jaksoittain käytöstä hitaiden mittauksen aikana.

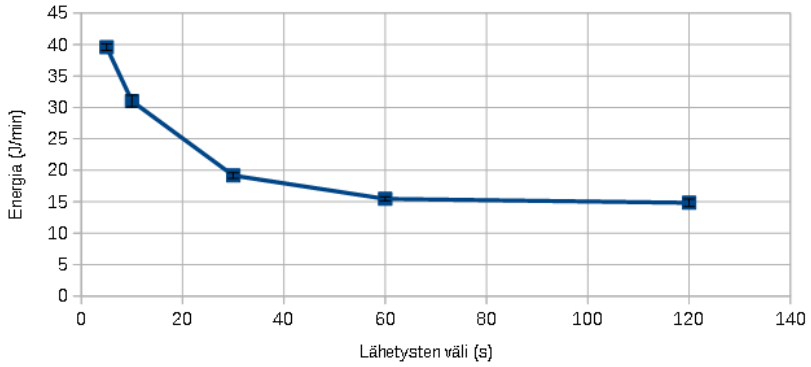


Kuvio 2: GPS-vastaanottimen, kiihtyvyysanturin ja paineanturin energiankulutus eri päivitysnopeuksilla ilman komponentin muita toimintoja. Kiihtyvyys- ja paineanturin sensorikuuntelija on poistettu yli 200 ms:n taukojen aikana käytöstä.

Tiedosto-operaatioiden yhdistelemisellä ja yhdellä kerralla tallennettavan tiedon määrällä ei tässä sovelluksessa tutkituissa rajoissa todettu olevan mitään vaikutusta. Kirjoitettavan tiedon

määrä oli pieni (1 – 5 kt) ja kirjoituksia suoritettiin harvoin (5, 10 tai 20 sekunnin välein), eikä tiedostonkäsittelyn tuottamaa vaikutusta erotettu missään testissä ilman sitä suoritetusta vertailumittauksesta.

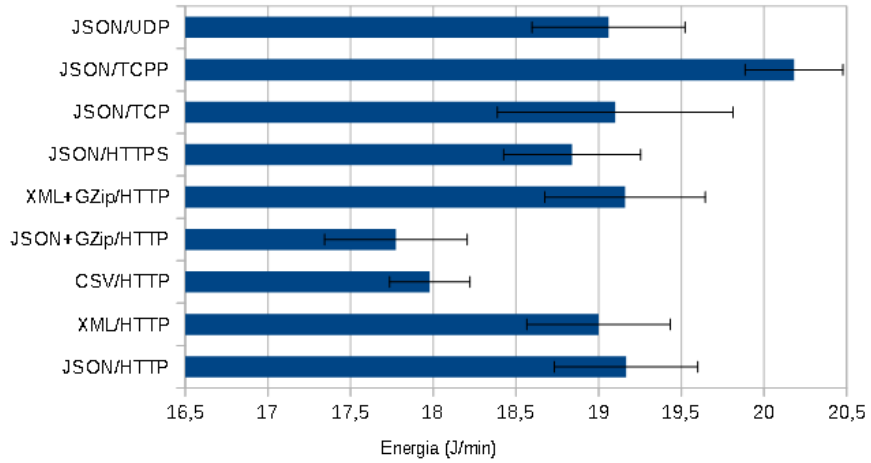
Verkkoliikenteen vaikutus komponentin kokonaisenergiankulutukseen oli merkittävä ja tiedon lähettäminen harvemmin, suurempina kokonaisuuksina, säästi suositusten mukaisesti energiaa. Jos reittitiedot lähetettiin palvelimelle viiden sekunnin välein, komponentin energiankulutus kaksinkertaistui 30 sekunnin lähetysnopeutta käyttävään konfiguraatioon verrattuna (kaavio 3). Minuutin ja kahden minuutin lähetysnopeuksien välillä energiankulutus ei kuitenkaan enää muuttunut merkittävästi. Energiankulutustallenteita vertaamalla havaittiin, että vaikka lähetysjaksokohtainen datamäärä 30 ja 120 sekunnin välillä nelinkertaistui, kesti radiorajapinnan aktivaatio molemmissa noin viisi sekuntia ja energiankulutus oli lähes vastaava. Radiorajapinnan käyttöä seuraava aktivaatio hallitsee siis tässä sovelluksessa tiedonsiirron energiankulutusta. Tämä havainto vastaa aiempien tutkimusten tuloksia 3G-verkoissa (esim. Balasubramanian, Balasubramanian ja Venkataramani 2009).



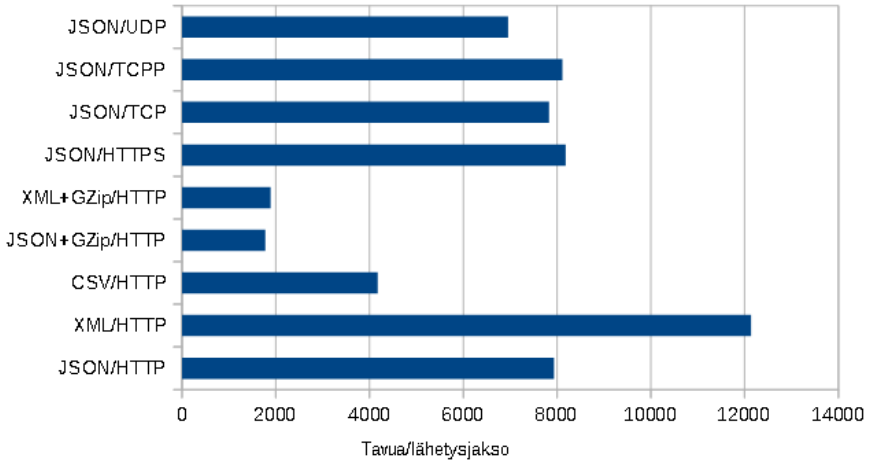
Kuvio 3: Komponentin energiankulutus lähetettäessä näytteitä palvelimelle JSON-formaatissa, HTTP-protokollalla eri nopeuksilla. GPS-vastaanottimen päivitysnopeus 5 s, kiihtyvyysanturin 1 s ja paineanturin 2 s.

Tiedonsiirtoformaattien, lähetettävän tiedon pakkaamisen ja tiedonsiirtoprotokollien mahdollistamat säästöt eivät tässä sovelluksessa olleet yhtä merkittäviä. Verrattaessa HTTP-, HTTPS-, TCP- ja UDP-protokollia ei niiden välillä havaittu eroja, jotka ylittäisivät GPS- ja radiorajapinnan tuottaman satunnaisvaihtelun vaikutuksen kerätyssä aineistossa (kaavio 4). Tämä johtui todennäköisesti radiorajapinnan käyttöä seuraavan aktivaation energiankulutus-

ta hallitsevasta roolista ja siitä, että kaikissa näissä yhdistelmissä siirrettiin lähetysjaksossaan vastaava määrä dataa (kaavio 5). Vähiten tietoa siirtävät GZip-pakattu JSON ja CSV-formaatti voivat kuitenkin luultavasti säästää vähän energiaa. GZip-pakatun XML:n siirtämän tiedon määrään verrattuna odottamattoman heikko suorituskyky johtui todennäköisesti sen JSON:ia suuremmasta pakattavasta tietomäärästä ja näin pakkauksen energiankulutuksesta. Lisäksi vaihtoehdoista odotetusti heikoimmin suoriutui ylläpitoon liittyvän liikenteensä takia TCP-yhteys, jossa sokettia pidettiin aina auki.



Kuvio 4: Komponentin energiankulutus lähetettäessä näytteitä palvelimelle 30 sekunnin välein. GPS-vastaanottimen päivitysnopeus 5 s, kiihtyvyysanturin 1 s ja paineanturin 2 s. TCPP on TCP-yhteys, jossa soketti pidetään avoinna. Virhepalkit kuvaavat keskivirhettä.



Kuvio 5: Lähetettävän datan määrä lähetysjaksossaan eri formaatti- ja protokollayhdistelmillä lähetettäessä 30 sekunnin välein. TCPP on TCP-yhteys, jossa soketti pidetään avoinna.

Testeissä käytetyn GS5-puhelimen akun kapasiteetti on 10,78 Wh, eli noin 39 kJ. Jos kehitetty palvelukomponentti on ainoa laitteella suoritettava sovellus, olisi sitä mittaustulosten perusteella mahdollista käyttää vaatimusten mukaisessa konfiguraatiossa yli 30 tuntia. Vastaavasti todettiin myös, että laitteen lepotilan estäminen (PARTIAL_WAKE_LOCK) ilman muita toimintoja ei aiheuttanut merkittävää energiankulutusta, vaan siitä huolimatta laitteen akku kestäisi teoriassa noin 400 tunnin käytön ("Keeping the Device Awake | Android Developers" 2016). Suoritettu testaus kattaa kuitenkin vain yhden ympäristön ja kenttämittaukset voisivat tarjota paremman arvion komponentin käyttäytymisestä realistisemmissa käyttöolosuhteissa. Lisäksi esimerkiksi eri sensoreita yhdistelevien kirjastojen, kuten Androidin paikannusrajapinnan, testaaminen voi vaatia laitteen liikuttamista realistisen toiminnan mittaamiseksi ("Making Your App Location-Aware | Android Developers" 2016). Tässä työssä kehitetty USB-mittalaite voi olla hyödyllinen myös kenttätesteissä, jos mittaustulokset tallennetaan esimerkiksi toisella Android-mobiililaitteella. Jatkokehityksen kannalta kenttämittaustarve voisi olla järkevää huomioida lisäämällä mahdollisuus kerätä mittauksia myös suoraan muistikortille, joka on liitettävissä yksinkertaisesti sekä pelkkään mikro-ohjaimeen että esimerkiksi Arduino-alustaan perustuvaan mittalaitteeseen.

Testatuista käytänteistä useimmat voitiin osoittaa tulosten perusteella myös tässä sovelluksessa käyttökelpoisiksi. Tiedostojen kirjoittamiseen ja HTTP:n sekä HTTPS:n eroihin liittyviä oletuksia ei kuitenkaan voida arvioida, mutta niiden mahdollisen vaikutuksen odotettiinkin olevan pieni ja sovelluksessa käsitellään suhteellisen vähän dataa (Naylor ym. 2014; J. Wang ym. 2012). Tiedonsiirrossa käytettäviin formaatteihin ja GZip-pakkaukseen liittyvät käytänteet saivat osittaista tukea, mutta siirrettävän tiedon pieni määrää ja verkko-olosuhteiden vaihtelu peittivät suuren osan niiden mahdollisista, pienistä eroista. Jos niitä haluttaisiin verrata tämän sovelluksen tapauksessa tarkemmin, pitäisi testausta suorittaa näiden konfiguraatioiden osalta käsiteltyä, yli 25 minuutin (50 toistoa) mittausaineistoa enemmän. Tämä näytemäärä osoittautui kuitenkin riittäväksi useimpien vertailujen suorittamiseen, vaikka niissä käytettiin sekä GPS-vastaanotinta että verkkorajapintaa, jotka molemmat lisäsivät niitä käyttävien testien mitatun energiankulutuksen hajontaa.

Erot esimerkiksi eri verkkoyhteysprotokollien energiankulutusten välillä olivat suoritetuissa testeissä ainoastaan alle 10 %. Samoja testejä pidemmällä aikavälillä toistettaessa todettiin,

että kunkin testin keskimääräinen, absoluuttinen energiankulutus voi muuttua ympäristön vaikutuksesta testauskertojen välillä jopa tätä enemmän, vaikka testien väliset suhteelliset erot säilyisivät ensisijaisesti vastaavina. Näin toisiinsa verrattavat testit on pyrittävä suorittamaan samalla kerralla, mahdollisimman vastaavissa olosuhteissa. Verkkorajapinnasta riippumattomissa testeissä tällaista vaikutusta ei todettu.

Nyt testejä suoritettiin käsin, yksi kerrallaan, eikä siihen käytetty mitään ympäristöä. Testauksen automatisointiin on kuitenkin olemassa käytännössä kaikille mobiilialustoille useampia työkaluja, jotka tukevat myös käyttöliittymien testaamista. Useimmat niistä riippuvat tietokoneesta, vaikka testejä suoritettaisiin mobiililaitteella, mutta osa tukee myös testien itsenäistä suorittamista. Manuaalisesti, yksi konfiguraatio kerrallaan, suoritetussa testauksessa mittaustallenteet oli yksinkertaista kerätä käsin, mutta automaattisessa testauksessa vähintään testien rajat on merkittävä tallenteisiin. Tämän kannalta kokeiltiinkin kirjallisuudessa esitettyä, näytön päälle ja pois kytkemistä, jonka todettiin tuottavan myös tällä laitteella hyvin erottuvan muutoksen energiankulutukseen (Rice ja Hay 2010).

Suoritetut testit osoittivat, että toteutetun kaltainen, edulliseen tehonmittauspiiriin perustuva mittalaite voi tarkkuutensa rajoituksista huolimatta tarjota yksityiskohtaista ja hyödyllistä tietoa energiankulutuksesta sovelluskehityksessä tehtävien päätösten tueksi. Lisäksi mittaukseen tarvittavien muutosten tekeminen modernien mobiililaitteiden akkuihin osoittautui mahdolliseksi suhteellisen pienellä vaivalla (liite C). Mittaustulosten käsittely testien kokonaiskulutuksen tasolla vaikuttaa myös käyttökelpoiselta vaihtoehdolta, jos erot ovat riittävän suuria suhteessa testien satunnaisvaihteluun tai tarkasteltavia kokonaisuuksia voidaan testata erillään.

8 Keskustelu ja mahdollinen jatkotutkimus

Mobiilisovellusten energiankulutus on merkittävä käytettävyyskysymys ja siihen liittyvät ongelmat voivat haitata myös sovellusten menestymistä kilpailluilla markkinoilla, joilla niille on aina tarjolla vaihtoehtoja (C. Wilke ym. 2013). Käyttäjien mielenkiinnosta huolimatta mobiililaitteiden energiankulutukseen liittyvät kysymyksiä eivät kuitenkaan ole vielä niin lajaasti tunnistettuja, että niitä kirjattaisiin usein vaatimuksiksi (Pang ym. 2016). Toisaalta mobiililaitteiden energiankulutuksen vaikutusta on pyritty vähentämään myös laitteistotasolla, erityisesti nopeuttamalla akkujen lataamista ja kehittämällä energiatiheydeltään parempia akkutekniikoita ("Qualcomm Quick Charge FAQs" 2016; "Doubling battery power of consumer electronics | MIT News" 2016).

Laitteiden suorituskyky ja energiankulutus kasvaa kuitenkin akkutekniikan kehitystä nopeammin, joten laitteistotasoiset keinot eivät todennäköisesti tule yksinään riittämään ongelman ratkaisuksi (Carroll ja Heiser 2013). Mobiililaitteiden kehittyessä niiden käyttökohteet ovat myös laajenemassa ja esimerkiksi virtuaalitodellisuuden sekä lisätyn todellisuuden sovellukset tulevat arkipäiväistymään. Nämä ovat molemmat sovellusryhmiä, joissa käytetään paljon energiaa kuluttavia laitteistokomponentteja ja raskasta laskentaa. Tämä voi tulla lisäämään mielenkiintoa myös energiatehokkaita ohjelmointiratkaisuja kohtaan.

Vaikka energiankulutukseen liittyvä tutkimus on jatkuvasti laajentunut 2010-luvulla, se on suorituskykyyn liittyvään kenttään verrattuna vielä selvästi heikommin vakiintunut. Tämän taustalla vaikuttaa todennäköisesti energiankulutuksen vaikutusten jääminen lähes näkymättömäksi suurimpia palvelimia lukuun ottamatta. Mobiililaitteisiin keskittyvän tutkimuksen tämänhetkistä tilannetta kuvaa puolestaan hyvin se, että aiheesta on julkaistu vasta vain yksittäisiä katsausartikkeleita tai kirjoja (esim. Tarkoma ym. 2014; Hoque ym. 2015).

8.1 Energiankulutuksen määritysmenetelmät

Tässä työssä tehtiin laaja katsaus mobiililaitteiden energiankulutusta käsittelevään kirjallisuuteen ja todettiin, että erilaisia energiankulutuksen määrittämismenetelmiä on käsitelty jo suhteellisen kattavasti. Näiden menetelmien käyttäjille julkaistut toteutukset osoittautuivat

kuitenkin harvinaisiksi, eikä energiankulutuksen määrittämistä ole lisätty minkään merkittävän sovelluskehitysympäristön osaksi. Näin edelleen avoimeksi jää kysymys siitä, miten energiankulutukseen liittyvä tehtäväkokonaisuus kannattaa yhdistää muun sovelluskehitysprosessin osaksi sen eri vaiheissa. Joitakin ajatuksia on esitetty sen tuomisesta muun jatkuvan testauksen osaksi ja tulosten esittämisestä koodieditorissa rivi- tai metoditasoisesti (Ding Li ym. 2013).

Mobiilisovellusten testaaminen on myös laajenemassa yksittäisistä laitteista ja emulaattoreista palveluihin, jotka tarjoavat sadoilla tai jopa tuhansilla erilaisilla laitteilla suoritettavaa rinnakkaista testausta (Rojas, Meireles ja Dias-Neto 2016; "Xamarin Test Cloud" 2016). Nämä ratkaisut ovat vielä käyttäjilleen suhteellisen kalliita, mutta yleistyessään ne voivat tulla merkittäväksi osaksi myös energiankulutuksen määrittämistä ja tarjota testauksessa käytettävän laitepopulaation kattavuuden ansiosta edustavampia tuloksia. Vielä tällä hetkellä energiankulutuksen määrittäminen ei kuitenkaan kuulu minkään testauspilven palveluvalikoimaan.

Koska energiankulutuksen määritysmenetelmien saatavuus todettiin kirjallisuuskatsauksessa heikoksi ja kaupalliset mittalaitteet poikkeuksetta kalliiksi, työn empiirisessä osassa suoritetun testauksen tueksi kehitettiin edullinen, tehonmittauspiiriin perustuva mittalaite. Sillä suoritetut testit osoittivat, että yksinkertainen mittalaite ja pelkän testikohtaisen kokonaiskulutuksen määrittäminen voivat olla hyödyllisiä välineitä energiatehokkaita toteutusvalintoja haettaessa. Näin olisikin toivottavaa, että muiden arviointimenetelmien ollessa edelleen harvinaisia edes mittalaitteet leviäisivät laajemmin sovelluskehittäjien käyttöön esimerkiksi valmiista, kaupallisista moduuleista koottavassa muodossaan. Tämä kehitys riippuu kuitenkin myös mittaustallenteiden analyysiin käytettävistä ohjelmista, joiden tarjonta on, kaupallisia lukuun ottamatta, varsin heikkoa.

8.2 Energiatehokkaan ohjelmoinnin käytänteet

Kirjallisuuskatsauksessa todettiin myös, että energiatehokkaan mobiiliohjelmoinnin käytänteitä on käsitelty vasta suhteellisen suppeasti ja korkealla tasolla. Tavallisesti niissä on keskitytty energiankulutukseltaan merkittävimpiin laitteistokomponentteihin, kuten radioraja-

pintoihin ja GPS-vastaanottimeen. Näitä käytänteitä yhteisesti kuvaava periaate on vähentää käyttöjaksojen lukumäärää ja kestoa. Mobiililaitealustojen julkaisijoiden omat ohjeet ovat myös jatkuvasti kehittyneet ja niistä osa vastaa kattavuudeltaan vähintäänkin viimeisimpien tutkimusartikkeleiden tasoa, joskaan ohjeissa ei yleensä kuvata käytänteiden mahdollisen vaikutuksen suuruusluokkaa (esim. “Optimizing battery life | Android Developers” 2016).

Suoraan laitteistokomponentteihin liittymättömiä ohjelmointikäytänteitä on esitetty vasta vähän. Lisäksi ohjelmistoalustojen erojen ja nopean kehityksen takia yksityiskohtaisempia käytänteitä on pidetty vain rajallisesti yleistettävinä, joka vähentää niiden arvoa (Ding Li ym. 2013; Hao ym. 2012). Yksi mahdollinen ratkaisu tähän haasteeseen voi tulevaisuudessa kuitenkin olla energiatehokkaiden ohjelmointikäytänteiden tunnistaminen versiohallintajärjestelmien sisältämien artefaktien automaattisen testauksen ja analyysin avulla (esim. Bao ym. 2016; Hindle ym. 2014). Parhaimmillaan tämä menetelmä voisi mahdollistaa laajasti eri alustoilla soveltuvien, ajantasaisten käytänteiden keräämisen lähes kokonaan ilman testitoteutuksia tai muuta käsityötä.

Työn empiirisessä osassa joitain kerätyistä käytänteistä testattiin reittimuotoisten paikkatietojen käsittelemiseen tarkoitetussa taustaprosessissa. Tässä kehyksessä erityisesti verkkorajapinnan aktivaatioiden lukumäärän ja sensoreista GPS-vastaanottimen lukunopeuden rajoittamisen todettiin säästävän kymmeniä prosentteja energiaa. Monet muut käytänteet, kuten flash-muistin aktivaatioiden rajoittaminen tai verkkoprotokollan valinta, eivät sen sijaan tuottaneet merkittävää vaikutusta tässä sovelluksessa. Lisäksi sovelletuilla käytänteillä todettiin selvät rajat, joiden jälkeen energiatehokkuus ei enää parane, mutta palvelun laatu heikkenee. Tämä tukee yhdessä ratkaisujen ympäristöön liittyvän herkkyyden kanssa näkemystä, jonka mukaan pelkät käytänteet ovat hyödyllisiä, mutta valintojen tekemistä on tarpeellista tukea myös todellisen kulutuksen määrittämisellä (Ding Li ym. 2013; Hao ym. 2012; Esmaeilzadeh ym. 2011).

8.3 Kysymyksiä jatkotutkimukseen

Yksi merkittävä, edelleen varsin vähän käsitelty kenttä ovat automaattiset menetelmät. Näistä energiankulutuksen kääntäjäoptimointeja on jo rajallisesti tutkittu ja niiden mahdollisuuk-

sien on pidetty vakiintuneiden, suorituskykyä tukevien kääntäjäoptimointien tasoisina. Lisäksi energiankulutuksen lähdekooditasoisten mikro-optimointien automatisointia on pidetty tärkeänä niiden suhteessa suuren työmäärän ja mahdollisesti pienen vaikutuksen takia (Kambadur ja Kim 2014). Mobiilisovellusten käyttäjille näkyvä, automaattisten työkalujen sovelluskohde voi tulevaisuudessa olla myös sovelluskauppojen suorittama energiankulutuksen arviointi, josta annettaisiin sovellukselle sen suhteellista energiankulutusta kuvaava arvosana (Claas Wilke ym. 2013).

Työtä tehdessä lähes avoimeksi jäi myös kysymys siitä, kuinka laajasti sovelluskehittäjät määrittävät energiankulutusta tai soveltavat jo tunnettuja ohjelmointikäytänteitä kehitystyötä tehdessään. Sekä näistä tekijöistä että käyttäjien suhtautumisesta mobiilisovellusten energiankulutukseen vaikuttaa olevan vasta vähän ja varsin alustavia tutkimustuloksia (mm. Pang ym. 2016; Pinto, Castor ja Liu 2014; C. Wilke ym. 2013). Käyttäjien toiveiden ja sovelluskehittäjien tarpeiden nykytilan kartoittaminen voisi auttaa jatkokehityksen tunnistamisessa ja motivoimisessa sekä edistää vielä hyvin rajallisen välinetarjonnan tulevaa kehitystä.

Lähteet

"2014 US Mobile Phone sales fall by 15% and handset replacement cycle lengthens to historic high". 2015. Viitattu 6. marraskuuta 2016. http://reconanalytics.com/2015/02/2014-us-mobile-phone-sales-fall-by-15-and-handset-replacement-cycle-lengthens-to-historic-high/.

"8 Channel USB GPIO Module With Analog inputs". 2016. Viitattu 8. joulukuuta 2016. http://numato.com/8-channel-usb-gpio-module-with-analog-inputs/.

"A Current Sensing tutorial - Part III: Accuracy". 2016. Viitattu 9. joulukuuta 2016. http://www.eetimes.com/document.asp?doc_id=1279451.

"About Simulator". 2016. Viitattu 2. helmikuuta 2016. https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html.

Alam, Faisal, Preeti Ranjan Panda, Nikhil Tripathi, Namita Sharma ja Sanjiv Narayan. 2014. "Energy Optimization in Android Applications Through Wakelock Placement". Teoksessa *Proceedings of the Conference on Design, Automation & Test in Europe,* 88:1–88:4. DATE '14. Dresden, Germany: European Design / Automation Association. ISBN: 978-3-9815370-2-4. http://dl.acm.org/citation.cfm?id=2616606.2616714.

"AlarmManager | Android Developers". 2016. Viitattu 26. marraskuuta 2016. https://developer.android.com/reference/android/app/AlarmManager.html.

Anand, Bhojan, Karthik Thirugnanam, Jeena Sebastian, Pravein G. Kannan, Akhihebbal L. Ananda, Mun Choon Chan ja Rajesh Krishna Balan. 2011. "Adaptive Display Power Management for Mobile Games". Teoksessa *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services,* 57–70. MobiSys '11. Bethesda, Maryland, USA: ACM. ISBN: 978-1-4503-0643-0. doi:10.1145/1999995.2000002. http://doi.acm.org/10.1145/1999995.2000002.

Andrae, Anders S. G., ja Tomas Edler. 2015. “On Global Electricity Usage of Communication Technology: Trends to 2030”. *Challenges* 6 (1): 117. ISSN: 2078-1547. doi:10.3390/challe6010117. http://www.mdpi.com/2078-1547/6/1/117.

“Android Interfaces and Architecture | Android Open Source Project”. 2016. Viitattu 18. marraskuuta 2016. https://source.android.com/devices/.

“Apple - Batteries - Maximizing Performance”. 2015. Viitattu 20. huhtikuuta 2015. https://www.apple.com/batteries/maximizing-performance/#ios.

“Appscope”. 2012. Viitattu 22. toukokuuta 2016. http://css3.yonsei.ac.kr/appscope.

Arai, N., T. Okuno, Y. Ishiyama ja K. Hirose. 2014. “NTT group’s approaches to energy reduction”. Teoksessa *2014 IEEE 36th International Telecommunications Energy Conference (INTELEC),* 1–5. Syyskuu. doi:10.1109/INTLEC.2014.6972183.

“Arduino - Home”. 2016. Viitattu 10. toukokuuta 2016. https://www.arduino.cc/.

“Arduino - Libraries”. 2016. Viitattu 9. joulukuuta 2016. https://www.arduino.cc/en/Reference/Libraries.

“Arduino Uno”. 2016. Viitattu 8. joulukuuta 2016. http://www.arduino.org/products/boards/arduino-uno.

“ART and Dalvik | Android Open Source Project”. 2016. Viitattu 26. marraskuuta 2016. https://source.android.com/devices/tech/dalvik/.

“ATMega88”. 2016. Viitattu 1. joulukuuta 2016. http://www.atmel.com/devices/ATMEGA88.aspx.

“Atmel Studio 7”. 2016. Viitattu 1. joulukuuta 2016. http://www.atmel.com/microsite/atmel-studio/.

Austin, T., E. Larson ja D. Ernst. 2002. “SimpleScalar: an infrastructure for computer system modeling”. *Computer* 35, numero 2 (helmikuu): 59–67. ISSN: 0018-9162. doi:10.1109/2.982917.

"AVR Libc reference manual". 2016. Viitattu 1. joulukuuta 2016. http://www.atmel.com/webdoc/AVRLibcReferenceManual/.

"AVRISP mkII". 2016. Viitattu 1. joulukuuta 2016. http://www.atmel.com/tools/AVRISPMKII.aspx.

Balasubramanian, Niranjan, Aruna Balasubramanian ja Arun Venkataramani. 2009. "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications". Teoksessa *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference,* 280–293. IMC '09. Chicago, Illinois, USA: ACM. ISBN: 978-1-60558-771-4. doi:10.1145/1644893.1644927. http://doi.acm.org/10.1145/1644893.1644927.

Bao, Lingfeng, David Lo, Xin Xia, Xinyu Wang ja Cong Tian. 2016. "How Android App Developers Manage Power Consumption?: An Empirical Study by Mining Power Management Commits". Teoksessa *Proceedings of the 13th International Workshop on Mining Software Repositories,* 37–48. MSR '16. Austin, Texas: ACM. ISBN: 978-1-4503-4186-8. doi:10.1145/2901739.2901748. http://doi.acm.org/10.1145/2901739.2901748.

Barroso, L. A., ja U. Holzle. 2007. "The Case for Energy-Proportional Computing". *Computer* 40, numero 12 (joulukuu): 33–37. ISSN: 0018-9162. doi:10.1109/MC.2007.443.

"Battery: making it last on your Windows Phone". 2015. Viitattu 20. huhtikuuta 2015. http://www.windowsphone.com/en-us/how-to/wp8/phones-and-hardware/battery-making-it-last.

"BCM47521". 2016. Viitattu 18. marraskuuta 2016. https://www.broadcom.com/products/wireless-connectivity/gps/bcm47521.

Behrouz, R.J., A. Sadeghi, J. Garcia, S. Malek ja P. Ammann. 2015. "EcoDroid: An Approach for Energy-Based Ranking of Android Apps". Teoksessa *Green and Sustainable Software (GREENS), 2015 IEEE/ACM 4th International Workshop on,* 8–14. Toukokuu. doi:10.1109/GREENS.2015.9.

Benini, L., R. Hodgson ja P. Siegel. 1998. “System-level power estimation and optimization”. Teoksessa *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on,* 173–178. Elokuu.

Bhattacharya, Suparna, Karthick Rajamani, K. Gopinath ja Manish Gupta. 2012. “Does Lean Imply Green?: A Study of the Power Performance Implications of Java Runtime Bloat”. *SIGMETRICS Perform. Eval. Rev.* (New York, NY, USA) 40, numero 1 (kesäkuu): 259–270. ISSN: 0163-5999. doi:10.1145/2318857.2254789. http://doi.acm/10.1145/2318857.2254789.

“big.LITTLE Technology - ARM”. 2016. Viitattu 24. toukokuuta 2016. https://www.arm.com/products/processors/technologies/biglittleprocessing.php.

Brandolese, Carlo, Simone Corbetta ja William Fornaciari. 2011. “Software Energy Estimation Based on Statistical Characterization of Intermediate Compilation Code”. Teoksessa *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design,* 333–338. ISLPED ’11. Fukuoka, Japan: IEEE Press. ISBN: 978-1-61284-660-6. http://dl.acm/citation.cfm?id=2016802.2016877.

Brouwers, Niels, Marco Zuniga ja Koen Langendoen. 2014. “NEAT: A Novel Energy Analysis Toolkit for Free-roaming Smartphones”. Teoksessa *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems,* 16–30. SenSys ’14. Memphis, Tennessee: ACM. ISBN: 978-1-4503-3143-2. doi:10.1145/2668332.2668337. http://doi.acm/10.1145/2668332.2668337.

Capra, Eugenio, Chiara Francalanci ja Sandra A. Slaughter. 2012. “Is Software "Green"? Application Development Environments and Energy Efficiency in Open Source Applications”. *Inf. Softw. Technol.* (Newton, MA, USA) 54, numero 1 (tammikuu): 60–71. ISSN: 0950-5849. doi:10.1016/j.infsof.2011.07.005. http://dx.doi/10.1016/j.infsof.2011.07.005.

Carroll, Aaron, ja Gernot Heiser. 2010. “An Analysis of Power Consumption in a Smartphone”. Teoksessa *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference,* 21–21. USENIXATC’10. Boston, MA: USENIX Association. http://dl.acm.org/citation.cfm?id=1855840.1855861.

———. 2013. “The Systems Hacker’s Guide to the Galaxy Energy Usage in a Modern Smartphone”. Teoksessa *Proceedings of the 4th Asia-Pacific Workshop on Systems,* 5:1–5:7. APSys ’13. Singapore, Singapore: ACM. ISBN: 978-1-4503-2316-1. doi:10.1145/2500727.2500734. http://doi.acm/10.1145/2500727.2500734.

Chang, Jichuan, Justin Meza, Parthasarathy Ranganathan, Amip Shah, Rocky Shih ja Cullen Bash. 2012. “Totally Green: Evaluating and Designing Servers for Lifecycle Environmental Impact”. *SIGARCH Comput. Archit. News* (New York, NY, USA) 40, numero 1 (maaliskuu): 25–36. ISSN: 0163-5964. doi:10.1145/2189750.2150980. http://doi.acm.org/10.1145/2189750.2150980.

Chatzigeorgiou, Alexander, ja George Stephanides. 2002. “Evaluating Performance and Power of Object-Oriented Vs. Procedural Programming in Embedded Processors”. Teoksessa *Proceedings of the 7th Ada-Europe International Conference on Reliable Software Technologies,* 65–75. Ada-Europe ’02. London, UK, UK: Springer-Verlag. ISBN: 3-540-43784-3. http://dl.acm/citation.cfm?id=646581.697920.

Chen, Xiang, Yiran Chen, Zhan Ma ja Felix C. A. Fernandes. 2013. “How is Energy Consumed in Smartphone Display Applications?” Teoksessa *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications,* 3:1–3:6. HotMobile ’13. Jekyll Island, Georgia: ACM. ISBN: 978-1-4503-1421-3. doi:10.1145/2444776.2444781. http://doi.acm/10.1145/2444776.2444781.

Chetty, Marshini, A.J. Bernheim Brush, Brian R. Meyers ja Paul Johns. 2009. “It’s Not Easy Being Green: Understanding Home Computer Power Management”. Teoksessa *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 1033–1042. CHI ’09. Boston, MA, USA: ACM. ISBN: 978-1-60558-246-7. doi:10.1145/1518701.1518860. http://doi.acm/10.1145/1518701.1518860.

Chidambaram Nachiappan, Nachiappan, Praveen Yedlapalli, Niranjan Soundararajan, Mahmut Taylan Kandemir, Anand Sivasubramaniam ja Chita R. Das. 2014. “GemDroid: A Framework to Evaluate Mobile Platforms”. *SIGMETRICS Perform. Eval. Rev.* (New York, NY, USA) 42, numero 1 (kesäkuu): 355–366. ISSN: 0163-5999. doi:10.1145/2637364.2591973. http://doi.acm/10.1145/2637364.2591973.

Chon, Yohan, Elmurod Talipov, Hyojeong Shin ja Hojung Cha. 2011. “Mobility Prediction-based Smartphone Energy Optimization for Everyday Location Monitoring”. Teoksessa *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems,* 82–95. SenSys ’11. Seattle, Washington: ACM. ISBN: 978-1-4503-0718-5. doi:10.1145/2070942.2070952. http://doi.acm.org/10.1145/2070942.2070952.

“Chrome V8 | Google Developers”. 2016. Viitattu 6. marraskuuta 2016. https://developers.google.com/v8/.

Chung, Yi-Fan, Chun-Yu Lin ja Chung-Ta King. 2011. “ANEPROF: Energy Profiling for Android Java Virtual Machine and Applications”. Teoksessa *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on,* 372–379. Joulukuu. doi:10.1109/ICPADS.2011.28.

Cignetti, Todd L., Kirill Komarov ja Carla Schlatter Ellis. 2000. “Energy Estimation Tools for the Palm”. Teoksessa *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems,* 96–103. MSWIM ’00. Boston, Massachusetts, USA: ACM. ISBN: 1-58113-304-9. doi:10.1145/346855.346869. http://doi.acm/10.1145/346855.346869.

“Convert an Android Device to Linux”. 2015. Viitattu 10. maaliskuuta 2015. http://www.linux-magazine.com/Online/Features/Convert-an-Android-Device-to-Linux.

Corral, Luis, Anton B. Georgiev, Andrea Janes ja Stefan Kofler. 2015. “Energy-aware Performance Evaluation of Android Custom Kernels”. Teoksessa *Proceedings of the Fourth International Workshop on Green and Sustainable Software,* 1–7. GREENS ’15. Florence, Italy: IEEE Press. http://dl.acm.org/citation.cfm?id=2820158.2820160.

Corral, Luis, Anton B. Georgiev, Alberto Sillitti ja Giancarlo Succi. 2013. “A Method for Characterizing Energy Consumption in Android Smartphones”. Teoksessa *Proceedings of the 2Nd International Workshop on Green and Sustainable Software,* 38–45. GREENS ’13. San Francisco, California: IEEE Press. ISBN: 978-1-4673-6267-2. http://dl.acm.org/citation.cfm?id=2662693.2662701.

———. 2014. “Method Reallocation to Reduce Energy Consumption: An Implementation in Android OS”. Teoksessa *Proceedings of the 29th Annual ACM Symposium on Applied Computing,* 1213–1218. SAC ’14. Gyeongju, Republic of Korea: ACM. ISBN: 978-1-4503-2469-4. doi:10.1145/2554850.2555064. http://doi.acm.org/10.1145/2554850.2555064.

Datta, S.K., C. Bonnet ja N. Nikaein. 2012. “Android power management: Current and future trends”. Teoksessa *Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012 First IEEE Workshop on,* 48–53. Kesäkuu. doi:10.1109/ETSIoT.2012.6311253.

———. 2013. “Minimizing energy expenditure in smart devices”. Teoksessa *Information Communication Technologies (ICT), 2013 IEEE Conference on,* 712–717. Huhtikuu. doi:10.1109/CICT.2013.6558187.

Ding, Ning, Daniel Wagner, Xiaomeng Chen, Abhinav Pathak, Y. Charlie Hu ja Andrew Rice. 2013. “Characterizing and Modeling the Impact of Wireless Signal Strength on Smartphone Battery Drain”. *SIGMETRICS Perform. Eval. Rev.* (New York, NY, USA) 41, numero 1 (kesäkuu): 29–40. ISSN: 0163-5999. doi:10.1145/2494232.2466586. http://doi.acm/10.1145/2494232.2466586.

Dong, Mian, Yung-Seok Kevin Choi ja Lin Zhong. 2009. “Power-saving Color Transformation of Mobile Graphical User Interfaces on OLED-based Displays”. Teoksessa *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design,* 339–342. ISLPED ’09. San Fancisco, CA, USA: ACM. ISBN: 978-1-60558-684-7. doi:10.1145/1594233.1594317. http://doi.acm.org/10.1145/1594233.1594317.

Dong, Mian, ja Lin Zhong. 2011. “Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems”. Teoksessa *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services,* 335–348. MobiSys ’11. Bethesda, Maryland, USA: ACM. ISBN: 978-1-4503-0643-0. doi:10.1145/1999995.2000027. http://doi.acm.org/10.1145/1999995.2000027.

“Doubling battery power of consumer electronics | MIT News”. 2016. Viitattu 18. marraskuuta 2016. http://news.mit.edu/2016/lithium-metal-batteries-double-power-consumer-electronics-0817.

“Downloads - hmavp”. 2013. Viitattu 14. marraskuuta 2016. http://www.openfoundry.org/of/projects/1621/download.

“DS2784 1-Cell Stand-Alone Fuel Gauge IC with Li+ Protector and SHA-1 Authentication - Maxim”. 2015. Viitattu 10. huhtikuuta 2015. http://www.maximintegrated.com/en/products/power/battery-management/DS2784.html.

Al-Dulaimy, Auday, Wassim Itani, Ahmed Zekri ja Rached Zantout. 2016. “Power management in virtualized data centers: state of the art”. *Journal of Cloud Computing* 5 (1): 1.

Dumke, R. 2001. *Performance Engineering: State of the Art and Current Trends.* Lecture Notes in Computer Science. Springer. ISBN: 9783540421450. https://books.google.fi/books?id=Bac51dM5IvIC.

Eeckhout, L., S. Nussbaum, J.E. Smith ja K. De Bosschere. 2003. “Statistical simulation: adding efficiency to the computer designer’s toolbox”. *Micro, IEEE* 23, numero 5 (syyskuu): 26–38. ISSN: 0272-1732. doi:10.1109/MM.2003.1240210.

“Energy-Efficient Software Guidelines - Intel Developer Zone”. 2011. Viitattu 4. marraskuuta 2016. https://software.intel.com/sites/default/files/m/3/c/b/Energy_Efficient_Software_Guidelines_v3_4_10_11.pdf.

Eom, Seung-Wook, Min-Kyu Kim, Ick-Jun Kim, Seong-In Moon, Yang-Kook Sun ja Hyun-Soo Kim. 2007. “Life prediction and reliability assessment of lithium secondary batteries”. 13th International Meeting on Lithium Batteries, *Journal of Power Sources* 174 (2): 954–958. ISSN: 0378-7753. doi:http://dx.doi.org/10.1016/j.jpowsour.2007.06.208. http://www.sciencedirect.com/science/article/pii/S0378775307013389.

Esmaeilzadeh, Hadi, Ting Cao, Yang Xi, Stephen M. Blackburn ja Kathryn S. McKinley. 2011. “Looking Back on the Language and Hardware Revolutions: Measured Power, Performance, and Scaling”. *SIGARCH Comput. Archit. News* (New York, NY, USA) 39, numero 1 (maaliskuu): 319–332. ISSN: 0163-5964. doi:10.1145/1961295.1950402. http://doi.acm/10.1145/1961295.1950402.

Falaki, Hossein, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan ja Deborah Estrin. 2010. “Diversity in Smartphone Usage”. Teoksessa *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services,* 179–194. MobiSys ’10. San Francisco, California, USA: ACM. ISBN: 978-1-60558-985-5. doi:10.1145/1814433.1814453. http://doi.acm.org/10.1145/1814433.1814453.

Gao, Xing, Dachuan Liu, Daiping Liu ja Haining Wang. 2016. “On Energy Security of Smartphones”. Teoksessa *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy,* 148–150. CODASPY ’16. New Orleans, Louisiana, USA: ACM. ISBN: 978-1-4503-3935-3. doi:10.1145/2857705.2857738. http://doi.acm.org/10.1145/2857705.2857738.

“Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013”. 2014. Viitattu 16. lokakuuta 2014. http://www.gartner.com/newsroom/id/2665715.

“Gartner Says Worldwide Smartphone Sales to Slow in 2016”. 2016. Viitattu 6. marraskuuta 2016. http://www.gartner.com/newsroom/id/3339019.

Gerin, Patrice, Mian Muhammad Hamayun ja Frédéric Pétrot. 2009. “Native MPSoC Co-simulation Environment for Software Performance Estimation”. Teoksessa *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis,* 403–412. CODES+ISSS ’09. Grenoble, France: ACM. ISBN: 978-1-60558-628-1. doi:10.1145/1629435.1629490. http://doi.acm/10.1145/1629435.1629490.

Gil, Bruno, ja Paulo Trezentos. 2011. “Impacts of Data Interchange Formats on Energy Consumption and Performance in Smartphones”. Teoksessa *Proceedings of the 2011 Workshop on Open Source and Design of Communication,* 1–6. OSDOC ’11. Lisboa, Portugal: ACM. ISBN: 978-1-4503-0873-1. doi:10.1145/2016716.2016718. http://doi.acm.org/10.1145/2016716.2016718.

“Google unlocks once-secret server”. 2009. Viitattu 4. marraskuuta 2016. https://www.cnet.com/news/google-uncloaks-once-secret-server-10209580/.

Gough, Corey, Ian Steiner ja Winston A. Saunders. 2015. *Energy Efficient Servers: Blueprints for Data Center Optimization.* 1st. Berkely, CA, USA: Apress. ISBN: 1430266376, 9781430266372.

Gupta, A., A. Djahromi, A. Eltawil, N. Dutt ja F. Kurdahi. 2008. “Managing leakage power and reliability in hot chips using system floorplanning and SRAM design”. Teoksessa *2008 14th International Workshop on Thermal Inveatigation of ICs and Systems,* 37–42. Syyskuu. doi:10.1109/THERMINIC.2008.4669875.

Hao, Shuai, Ding Li, William G. J. Halfond ja Ramesh Govindan. 2012. “Estimating Android Applications’ CPU Energy Usage via Bytecode Profiling”. Teoksessa *Proceedings of the First International Workshop on Green and Sustainable Software,* 1–7. GREENS ’12. Zurich, Switzerland: IEEE Press. ISBN: 978-1-4673-1832-7. http://dl.acm/citation.cfm?id=2663779.2663780.

———. 2013. “Estimating Mobile Application Energy Consumption Using Program Analysis”. Teoksessa *Proceedings of the 2013 International Conference on Software Engineering,* 92–101. ICSE ’13. San Francisco, CA, USA: IEEE Press. ISBN: 978-1-4673-3076-3. http://dl.acm.org/citation.cfm?id=2486788.2486801.

Heddeghem, Ward Van, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet ja Piet Demeester. 2014. “Trends in worldwide ICT electricity consumption from 2007 to 2012”. Green Networking, *Computer Communications* 50:64–76. ISSN: 0140-3664. doi:http://dx.doi.org/10.1016/j.comcom.2014.02.008. http://www.sciencedirect.com/science/article/pii/S0140366414000619.

Heikkinen, Mikko V.J., Jukka K. Nurminen, Timo Smura ja Heikki Hämmäinen. 2012. “Energy efficiency of mobile handsets: Measuring user attitudes and behavior”. Green Information Communication Technology, *Telematics and Informatics* 29 (4): 387–399. ISSN: 0736-5853. doi:http://dx.doi.org/10.1016/j.tele.2012.01.005. http://www.sciencedirect.com/science/article/pii/S0736585312000068.

Hevner, Alan R., Salvatore T. March, Jinsoo Park ja Sudha Ram. 2004. “Design Science in Information Systems Research”. *MIS Q.* (Minneapolis, MN, USA) 28, numero 1 (maaliskuu): 75–105. ISSN: 0276-7783. http://dl.acm.org/citation.cfm?id=2017212.2017217.

“HHVM”. 2016. Viitattu 6. marraskuuta 2016. http://hhvm.com/.

Hindle, Abram, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell ja Stephen Romansky. 2014. “GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework”. Teoksessa *Proceedings of the 11th Working Conference on Mining Software Repositories,* 12–21. MSR 2014. Hyderabad, India: ACM. ISBN: 978-1-4503-2863-0. doi:10.1145/2597073.2597097. http://doi.acm/10.1145/2597073.2597097.

“History for core/java/com/android/internal/os/BatteryStatsImpl.java - android/platform _frameworks_base GitHub”. 2015. Viitattu 20. huhtikuuta 2015. https://github.com/android/platform_frameworks_base/commits/master/core/java/com/android/internal/os/BatteryStatsImpl.java.

Holleis, Paul, Marko Luther, Gregor Broll ja Bertrand Souville. 2013. “A DIY Power Monitor to Compare Mobile Energy Consumption in Situ”. Teoksessa *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services,* 416–421. MobileHCI ’13. Munich, Germany: ACM. ISBN: 978-1-4503-2273-7. doi:10.1145/2493190.2494087. http://doi.acm/10.1145/2493190.2494087.

Höpfner, Hagen, Maximilian Schirmer ja Christian Bunse. 2012. “On Measuring Smartphones’ Software Energy Requirements.” Teoksessa *ICSOFT,* 165–171.

Hoque, Mohammad Ashraful, Matti Siekkinen, Kashif Nizam Khan, Yu Xiao ja Sasu Tarkoma. 2015. “Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices”. *ACM Comput. Surv.* (New York, NY, USA) 48, numero 3 (joulukuu): 39:1–39:40. ISSN: 0360-0300. doi:10.1145/2840723. http://doi.acm.org/10.1145/2840723.

Hsu, Chung-Hsing, ja Ulrich Kremer. 2003. “The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction”. Teoksessa *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation,* 38–48. PLDI ’03. San Diego, California, USA: ACM. ISBN: 1-58113-662-5. doi:10.1145/781131.781137. http://doi.acm/10.1145/781131.781137.

Hsu, Wen-Chang, Shih-Hao Hung ja Chia-Heng Tu. 2010. “A Virtual Timing Device for Program Performance Analysis”. Teoksessa *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on,* 2255–2260. Kesäkuu. doi:10.1109/CIT.2010.389.

Huang, Junxian, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen ja Oliver Spatscheck. 2012. “A Close Examination of Performance and Power Characteristics of 4G LTE Networks”. Teoksessa *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services,* 225–238. MobiSys ’12. Low Wood Bay, Lake District, UK: ACM. ISBN: 978-1-4503-1301-8. doi:10.1145/2307636.2307658. http://doi.acm.org/10.1145/2307636.2307658.

Hübert, H., ja B. Stabernack. 2010. “Energy analysis of embedded software based on a cycle-accurate processor power model”. Teoksessa *Industrial Electronics Applications (ISIEA), 2010 IEEE Symposium on,* 602–607. Lokakuu. doi:10.1109/ISIEA.2010.5679393.

“idleTimerDisabled - UIApplication | Apple Developer Documentation”. 2016. Viitattu 14. marraskuuta 2016. https://developer.apple.com/reference/uikit/uiapplication/1623070-idletimerdisabled.

Imielinski, Tomasz, ja B. R. Badrinath. 1994. “Mobile Wireless Computing: Challenges in Data Management”. *Commun. ACM* (New York, NY, USA) 37, numero 10 (lokakuu): 18–28. ISSN: 0001-0782. doi:10.1145/194313.194317. http://doi.acm/10.1145/194313.194317.

“INA219 Datasheet - Texas Instruments”. 2016. Viitattu 1. joulukuuta 2016. http://www.ti.com/lit/ds/symlink/ina219.pdf.

“INA219 High Side DC Current Sensor Breakout”. 2016. Viitattu 10. toukokuuta 2016. https://www.adafruit.com/product/904.

“Issue 61975 - android - Undo removal of access to BATTERY_STATS permission for apps - Android Open Source Project - Issue Tracker - Google Project Hosting”. 2015. Viitattu 20. huhtikuuta 2015. https://code.google.com/p/android/issues/detail?id=61975.

Jin, Tianxing, Songtao He ja Yunxin Liu. 2015. “Towards Accurate GPU Power Modeling for Smartphones”. Teoksessa *Proceedings of the 2Nd Workshop on Mobile Gaming,* 7–11. MobiGames ’15. Florence, Italy: ACM. ISBN: 978-1-4503-3499-0. doi:10.1145/2751496.2751502. http://doi.acm.org/10.1145/2751496.2751502.

Jung, Wonwoo, Chulkoo Kang, Chanmin Yoon, Donwon Kim ja Hojung Cha. 2012. “DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components”. Teoksessa *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* 353–362. CODES+ISSS ’12. Tampere, Finland: ACM. ISBN: 978-1-4503-1426-8. doi:10.1145/2380445.2380502. http://doi.acm/10.1145/2380445.2380502.

Kambadur, Melanie, ja Martha A. Kim. 2014. “An Experimental Survey of Energy Management Across the Stack”. *SIGPLAN Not.* (New York, NY, USA) 49, numero 10 (lokakuu): 329–344. ISSN: 0362-1340. doi:10.1145/2714064.2660196. http://doi.acm/10.1145/2714064.2660196.

Kapetanakis, K., ja S. Panagiotakis. 2012. “Efficient Energy Consumption’s Measurement on Android Devices”. Teoksessa *Informatics (PCI), 2012 16th Panhellenic Conference on,* 351–356. Lokakuu. doi:10.1109/PCi.2012.29.

Kawamoto, H. 2002. “The history of liquid-crystal displays”. *Proceedings of the IEEE* 90, numero 4 (huhtikuu): 460–500. ISSN: 0018-9219. doi:10.1109/JPROC.2002.1002521.

“Keeping the Device Awake | Android Developers”. 2016. Viitattu 14. marraskuuta 2016. https://developer.android.com/training/scheduling/wakelock.html.

Kim, Donnie H., Younghun Kim, Deborah Estrin ja Mani B. Srivastava. 2010. “SensLoc: Sensing Everyday Places and Paths Using Less Energy”. Teoksessa *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems,* 43–56. SenSys ’10. Zürich, Switzerland: ACM. ISBN: 978-1-4503-0344-6. doi:10.1145/1869983.1869989. http://doi.acm.org/10.1145/1869983.1869989.

Kim, Kitae, Donghwa Shin, Qing Xie, Yanzhi Wang, Massoud Pedram ja Naehyuck Chang. 2014. “FEPMA: Fine-grained Event-driven Power Meter for Android Smartphones Based on Device Driver Layer Event Monitoring”. Teoksessa *Proceedings of the Conference on Design, Automation & Test in Europe,* 367:1–367:6. DATE ’14. Dresden, Germany: European Design / Automation Association. ISBN: 978-3-9815370-2-4. http://dl.acm/citation.cfm?id=2616606.2617122.

Kistowski, Joakim v., Hansfried Block, John Beckett, Klaus-Dieter Lange, Jeremy A. Arnold ja Samuel Kounev. 2015. “Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive Workloads”. Teoksessa *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering,* 223–234. ICPE ’15. Austin, Texas, USA: ACM. ISBN: 978-1-4503-3248-4. doi:10.1145/2668930.2688057. http://doi.acm/10.1145/2668930.2688057.

Kjærgaard, MikkelBaun, ja Henrik Blunck. 2012. “Unsupervised Power Profiling for Mobile Devices” [kielellä English]. Teoksessa *Mobile and Ubiquitous Systems: Computing, Networking, and Services,* toimittanut Alessandro Puiatti ja Tao Gu, 104:138–149. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg. ISBN: 978-3-642-30972-4. doi:10.1007/978-3-642-30973-1_12. http://dx.doi.org/10.1007/978-3-642-30973-1_12.

Kong, Fanxin, ja Xue Liu. 2014. “A Survey on Green-Energy-Aware Power Management for Datacenters”. *ACM Comput. Surv.* (New York, NY, USA) 47, numero 2 (marraskuu): 30:1–30:38. ISSN: 0360-0300. doi:10.1145/2642708. http://doi.acm.org/10.1145/2642708.

Kwon, Young-Woo, ja E. Tilevich. 2013. “Reducing the Energy Consumption of Mobile Applications Behind the Scenes”. Teoksessa *Software Maintenance (ICSM), 2013 29th IEEE International Conference on,* 170–179. Syyskuu. doi:10.1109/ICSM.2013.28.

Lee, Jaejin, Junghyun Kim, Choonki Jang, Seungkyun Kim, Bernhard Egger, Kwangsub Kim ja SangYong Han. 2008. “FaCSim: A Fast and Cycle-accurate Architecture Simulator for Embedded Systems”. *SIGPLAN Not.* (New York, NY, USA) 43, numero 7 (kesäkuu): 89–100. ISSN: 0362-1340. doi:10.1145/1379023.1375670. http://doi.acm/10.1145/1379023.1375670.

Lee, Jaeseong, Yohan Chon ja Hojung Cha. 2015. “Evaluating Battery Aging on Mobile Devices”. Teoksessa *Proceedings of the 52Nd Annual Design Automation Conference,* 135:1–135:6. DAC ’15. San Francisco, California: ACM. ISBN: 978-1-4503-3520-1. doi:10.1145/2744769.2744838. http://doi.acm/10.1145/2744769.2744838.

Lee, Seokjun, Wonwoo Jung, Yohan Chon ja Hojung Cha. 2015. “EnTrack: A System Facility for Analyzing Energy Consumption of Android System Services”. Teoksessa *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing,* 191–202. UbiComp ’15. Osaka, Japan: ACM. ISBN: 978-1-4503-3574-4. doi:10.1145/2750858.2807531. http://doi.acm/10.1145/2750858.2807531.

“Lenovo A10 Multimode Laptop”. 2015. Viitattu 10. maaliskuuta 2015. http://shop.lenovo.com/fi/fi/laptops/lenovo/a-series/a10/.

Li, D., S. Hao, J. Gui ja W. G. J. Halfond. 2014. “An Empirical Study of the Energy Consumption of Android Applications”. Teoksessa *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on,* 121–130. Syyskuu. doi:10.1109/ICSME.2014.34.

Li, Ding, ja William G. J. Halfond. 2014. “An Investigation into Energy-saving Programming Practices for Android Smartphone App Development”. Teoksessa *Proceedings of the 3rd International Workshop on Green and Sustainable Software,* 46–53. GREENS 2014. Hyderabad, India: ACM. ISBN: 978-1-4503-2844-9. doi:10.1145/2593743.2593750. http://doi.acm.org/10.1145/2593743.2593750.

Li, Ding, Shuai Hao, William G. J. Halfond ja Ramesh Govindan. 2013. “Calculating Source Line Level Energy Information for Android Applications”. Teoksessa *Proceedings of the 2013 International Symposium on Software Testing and Analysis,* 78–89. ISSTA 2013. Lugano, Switzerland: ACM. ISBN: 978-1-4503-2159-4. doi:10.1145/2483760.2483780. http://doi.acm.org/10.1145/2483760.2483780.

Li, Tao, ja Lizy Kurian John. 2003. “Run-time Modeling and Estimation of Operating System Power Consumption”. Teoksessa *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems,* 160–171. SIGMETRICS ’03. San Diego, CA, USA: ACM. ISBN: 1-58113-664-1. doi:10.1145/781027.781048. http://doi.acm/10.1145/781027.781048.

Lin, Cheng-Yen, Chung-Wen Huang, Chi-Bang Kuan, Shi-Yu Huang ja Jenq-Kuen Lee. 2015. “The Design and Experiments of A SID-Based Power-Aware Simulator for Embedded Multicore Systems”. *ACM Trans. Des. Autom. Electron. Syst.* (New York, NY, USA) 20, numero 2 (maaliskuu): 22:1–22:27. ISSN: 1084-4309. doi:10.1145/2699834. http://doi.acm/10.1145/2699834.

Linares-Vasquez, Mario, Gabriele Bavota, Carlos Bernal-Cardenas, Rocco Oliveto, Massimiliano Di Penta ja Denys Poshyvanyk. 2014. “Mining Energy-greedy API Usage Patterns in Android Apps: An Empirical Study”. Teoksessa *Proceedings of the 11th Working Conference on Mining Software Repositories,* 2–11. MSR 2014. Hyderabad, India: ACM. ISBN:

978-1-4503-2863-0. doi:10.1145/2597073.2597085. http://doi.acm.org/10.1145/2597073.2597085.

Liu, Hao, Yaoxue Zhang ja Yuezhi Zhou. 2011. “TailTheft: Leveraging the Wasted Time for Saving Energy in Cellular Communications”. Teoksessa *Proceedings of the Sixth International Workshop on MobiArch,* 31–36. MobiArch ’11. Bethesda, Maryland, USA: ACM. ISBN: 978-1-4503-0740-6. doi:10.1145/1999916.1999925. http://doi.acm.org/10.1145/1999916.1999925.

Liu, Yepang, Chang Xu ja S.C. Cheung. 2013. “Where has my battery gone? Finding sensor related energy black holes in smartphone applications”. Teoksessa *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on,* 2–10. Maaliskuu. doi:10.1109/PerCom.2013.6526708.

“LM3641 Lithium-Ion Battery Pack Protection Circuit”. 2016. Viitattu 1. joulukuuta 2016. http://www.ti.com/lit/ds/symlink/lm3641.pdf.

“LTC5582 Datasheet - Linear Rechnology”. 2016. Viitattu 1. joulukuuta 2016. http://www.linear.com/docs/29427.

Luiz Sartor, A., U. Brisolara Correa ja A.C. Schneider Beck. 2013. “AndroProf: A Profiling Tool for the Android Platform”. Teoksessa *Computing Systems Engineering (SBESC), 2013 III Brazilian Symposium on,* 23–28. Joulukuu. doi:10.1109/SBESC.2013.15.

“LVK Series - Ohmite”. 2016. Viitattu 3. joulukuuta 2016. http://www.ohmite.com/cat/res_lvk.pdf.

“Major Mobile Milestones - The Last 15 Years, and the Next Five”. 2016. Viitattu 4. marraskuuta 2016. http://blogs.cisco.com/sp/mobile-vni-major-mobile-milestones-the-last-15-years-and-the-next-five.

Maker, Frank, Rajaveen Amirtharajah ja Venkatesh Akella. 2013. “Update Rate Tradeoffs for Improving Online Power Modeling in Smartphones”. Teoksessa *Proceedings of the 2013 International Symposium on Low Power Electronics and Design,* 114–119. ISLPED ’13. Beijing, China: IEEE Press. ISBN: 978-1-4799-1235-3. http://dl.acm/citation.cfm?id=2648668.2648696.

"Making Your App Location-Aware | Android Developers". 2016. Viitattu 2. joulukuuta 2016. https://developer.android.com/training/location/index.html.

"Manifest.permission | Android Developers". 2016. Viitattu 30. marraskuuta 2016. https://developer.android.com/reference/android/Manifest.permission.html#MODIFY_PHONE_STATE.

Manotas, Irene, Lori Pollock ja James Clause. 2014. "SEEDS: A Software Engineer's Energy-optimization Decision Support Framework". Teoksessa *Proceedings of the 36th International Conference on Software Engineering,* 503–514. ICSE 2014. Hyderabad, India: ACM. ISBN: 978-1-4503-2756-5. doi:10.1145/2568225.2568297. http://doi.acm/10.1145/2568225.2568297.

Manousakis, Ioannis, ja Dimitrios S. Nikolopoulos. 2012. "EPC: A Power Instrumentation Controller for Embedded Applications". *SIGBED Rev.* (New York, NY, USA) 9, numero 2 (kesäkuu): 28–32. ISSN: 1551-3688. doi:10.1145/2318836.2318841. http://doi.acm/10.1145/2318836.2318841.

Maplesden, D., E. Tempero, J. Hosking ja J. C. Grundy. 2015. "Performance Analysis for Object-Oriented Software: A Systematic Mapping". *IEEE Transactions on Software Engineering* 41, numero 7 (heinäkuu): 691–710. ISSN: 0098-5589. doi:10.1109/TSE.2015.2396514.

March, Salvatore T., ja Gerald F. Smith. 1995. "Design and natural science research on information technology". *Decision Support Systems* 15 (4): 251–266. ISSN: 0167-9236. doi:http://dx.doi.org/10.1016/0167-9236(94)00041-2. http://www.sciencedirect.com/science/article/pii/0167923694000412.

"MAX17050 ModelGauge m3 Fuel Gauge - Maxim". 2015. Viitattu 10. huhtikuuta 2015. http://www.maximintegrated.com/en/products/power/battery-management/MAX17050.html.

McCullough, John C., Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren ja Rajesh K. Gupta. 2011. “Evaluating the Effectiveness of Model-based Power Characterization”. Teoksessa *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference,* 12–12. USENIXATC’11. Portland, OR: USENIX Association. http://dl.acm.org/citation.cfm?id=2002181.2002193.

“Measuring Device Power | Android Open Source Project”. 2016. Viitattu 24. toukokuuta 2016. https://source.android.com/devices/tech/power/device.html.

Metri, Grace, Weisong Shi ja Monica Brockmeyer. 2015. “Energy-Efficiency Comparison of Mobile Platforms and Applications: A Quantitative Approach”. Teoksessa *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications,* 39–44. HotMobile ’15. Santa Fe, New Mexico, USA: ACM. ISBN: 978-1-4503-3391-7. doi:10.1145/2699343.2699358. http://doi.acm.org/10.1145/2699343.2699358.

Mirzaei, Nariman, Sam Malek, Corina S. Păsăreanu, Naeem Esfahani ja Riyadh Mahmood. 2012. “Testing Android Apps Through Symbolic Execution”. *SIGSOFT Softw. Eng. Notes* (New York, NY, USA) 37, numero 6 (marraskuu): 1–5. ISSN: 0163-5948. doi:10.1145/2382756.2382798. http://doi.acm/10.1145/2382756.2382798.

Mittal, Radhika, Aman Kansal ja Ranveer Chandra. 2012. “Empowering Developers to Estimate App Energy Consumption”. Teoksessa *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking,* 317–328. Mobicom ’12. Istanbul, Turkey: ACM. ISBN: 978-1-4503-1159-5. doi:10.1145/2348543.2348583. http://doi.acm.org/10.1145/2348543.2348583.

Moberg, Asa, Clara Borggren, Christine Ambell, Goran Finnveden, Fredrik Guldbrandsson, Anna Bondesson, Jens Malmodin ja Pernilla Bergmark. 2014. “Simplifying a life cycle assessment of a mobile phone”. *The International Journal of Life Cycle Assessment* 19 (5): 979–993. ISSN: 1614-7502. doi:10.1007/s11367-014-0721-6. http://dx.doi.org/10.1007/s11367-014-0721-6.

"Mobile App Testing | Keynote". 2016. Viitattu 18. marraskuuta 2016. http://www.keynote.com/solutions/testing/mobile-testing.

"Mobile Device Power Monitor Manual - Monsoon Solutions". 2008. Viitattu 10. toukokuuta 2016. https://www.msoon.com/LabEquipment/PowerMonitor/downloads/PowerMonitor_ManualVer1.3.pdf.

"Mobile Enerlytics". 2016. Viitattu 22. toukokuuta 2016. http://mobileenerlytics.com/index.php.

"Monitor the battery life, memory usage, CPU usage, and storage space on your device". 2015. Viitattu 20. huhtikuuta 2015. http://docs.blackberry.com/en/smartphone_users/deliverables/61420/amc1384977549398.jsp.

"Monsoon Solution Power Monitor". 2016. Viitattu 7. joulukuuta 2016. https://www.msoon.com/LabEquipment/PowerMonitor/.

"Motion Events". 2016. Viitattu 23. marraskuuta 2016. https://developer.apple.com/library/content/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/motion_event_basics/motion_event_basics.html.

"MPU-6500 | InvenSense". 2016. Viitattu 18. marraskuuta 2016. https://www.invensense.com/products/motion-tracking/6-axis/mpu-6500/.

Murmuria, Rahul, Jeffrey Medsger, Angelos Stavrou ja Jeffrey M. Voas. 2012. "Mobile Application and Device Power Usage Measurements". Teoksessa *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability,* 147–156. SERE '12. Washington, DC, USA: IEEE Computer Society. ISBN: 978-0-7695-4742-8. doi:10.1109/SERE.2012.19. http://dx.doi.org/10.1109/SERE.2012.19.

Muttreja, Anish, Anand Raghunathan, Srivaths Ravi ja Niraj K. Jha. 2005. "Hybrid Simulation for Embedded Software Energy Estimation". Teoksessa *Proceedings of the 42Nd Annual Design Automation Conference,* 23–26. DAC '05. Anaheim, California, USA: ACM. ISBN: 1-59593-058-2. doi:10.1145/1065579.1065590. http://doi.acm/10.1145/1065579.1065590.

Naylor, David, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki ja Peter Steenkiste. 2014. “The Cost of the "S"in HTTPS”. Teoksessa *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies,* 133–140. CoNEXT ’14. Sydney, Australia: ACM. ISBN: 978-1-4503-3279-8. doi:10.1145/2674005.2674991. http://doi.acm.org/10.1145/2674005.2674991.

“NCP4371 Product Preview”. 2016. Viitattu 15. toukokuuta 2016. http://www.onsemi.com/pub/Collateral/NCP4371-D.PDF.

“neat-power-toolkit Mobile power analysis toolkit for Android smartphones”. 2013. Viitattu 10. toukokuuta 2016. https://code.google.com/archive/p/neat-power-toolkit/.

Noureddine, Adel, Aurelien Bourdon, Romain Rouvoy ja Lionel Seinturier. 2012. “A Preliminary Study of the Impact of Software Engineering on GreenIT”. Teoksessa *Proceedings of the First International Workshop on Green and Sustainable Software,* 21–27. GREENS ’12. Zurich, Switzerland: IEEE Press. ISBN: 978-1-4673-1832-7. http://dl.acm/citation.cfm?id=2663779.2663783.

Oliner, Adam J., Anand P. Iyer, Ion Stoica, Eemil Lagerspetz ja Sasu Tarkoma. 2013. “Carat: Collaborative Energy Diagnosis for Mobile Devices”. Teoksessa *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems,* 10:1–10:14. SenSys ’13. Roma, Italy: ACM. ISBN: 978-1-4503-2027-6. doi:10.1145/2517351.2517354. http://doi.acm/10.1145/2517351.2517354.

Olsson, M., C. Cavdar, P. Frenger, S. Tombaz, D. Sabella ja R. Jantti. 2013. “5GrEEn: Towards Green 5G mobile networks”. Teoksessa *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob),* 212–216. Lokakuu. doi:10.1109/WiMOB.2013.6673363.

“Optimizing battery life | Android Developers”. 2016. Viitattu 14. joulukuuta 2016. https://developer.android.com/training/monitoring-device-state/index.html.

Ou, Zhonghong, Shichao Dong, Jiang Dong, Jukka K. Nurminen, Antti Ylä-Jääski ja Ren Wang. 2013. “Characterize Energy Impact of Concurrent Network-intensive Applications on Mobile Platforms”. Teoksessa *Proceedings of the Eighth ACM International Workshop on Mobility in the Evolving Internet Architecture,* 23–28. MobiArch ’13. Miami, Florida, USA: ACM. ISBN: 978-1-4503-2366-6. doi:10.1145/2505906.2505909. http://doi.acm.org/10.1145/2505906.2505909.

Paek, Jeongyeup, Joongheon Kim ja Ramesh Govindan. 2010. “Energy-efficient Rate-adaptive GPS-based Positioning for Smartphones”. Teoksessa *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services,* 299–314. MobiSys ’10. San Francisco, California, USA: ACM. ISBN: 978-1-60558-985-5. doi:10.1145/1814433.1814463. http://doi.acm.org/10.1145/1814433.1814463.

Pang, Candy, Abram Hindle, Bram Adams ja Ahmed E. Hassan. 2016. “What Do Programmers Know About Software Energy Consumption?” *IEEE Softw.* (Los Alamitos, CA, USA) 33, numero 3 (toukokuu): 83–89. ISSN: 0740-7459. doi:10.1109/MS.2015.83. http://dx.doi.org/10.1109/MS.2015.83.

Pathak, Abhinav, Y. Charlie Hu ja Ming Zhang. 2012. “Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof”. Teoksessa *Proceedings of the 7th ACM European Conference on Computer Systems,* 29–42. EuroSys ’12. Bern, Switzerland: ACM. ISBN: 978-1-4503-1223-3. doi:10.1145/2168836.2168841. http://doi.acm.org/10.1145/2168836.2168841.

Pathak, Abhinav, Y. Charlie Hu, Ming Zhang, Paramvir Bahl ja Yi-Min Wang. 2011. “Fine-grained Power Modeling for Smartphones Using System Call Tracing”. Teoksessa *Proceedings of the Sixth Conference on Computer Systems,* 153–168. EuroSys ’11. Salzburg, Austria: ACM. ISBN: 978-1-4503-0634-8. doi:10.1145/1966445.1966460. http://doi.acm.org/10.1145/1966445.1966460.

Pathak, Abhinav, Abhilash Jindal, Y. Charlie Hu ja Samuel P. Midkiff. 2012. “What is Keeping My Phone Awake?: Characterizing and Detecting No-sleep Energy Bugs in Smartphone Apps”. Teoksessa *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services,* 267–280. MobiSys ’12. Low Wood Bay, Lake District, UK: ACM. ISBN: 978-1-4503-1301-8. doi:10.1145/2307636.2307661. http://doi.acm.org/10.1145/2307636.2307661.

Patrick, Dale R., ja Stephen W. Fardo. 2008. *Electricity and Electronics Fundamentals.* Fairmont Press. ISBN: 9780881736014. http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=236618&site=ehost-live.

“PC Market Finishes 2015 As Expected, Hopefully Setting the Stage for a More Stable Future, According to IDC”. 2016. Viitattu 14. marraskuuta 2016. https://www.idc.com/getdoc.jsp?containerId=prUS40909316.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger ja Samir Chatterjee. 2007. “A Design Science Research Methodology for Information Systems Research”. *Journal of Management Information Systems* 24 (3): 45–77. doi:10.2753/MIS0742-1222240302. eprint: http://www.tandfonline.com/doi/pdf/10.2753/MIS0742-1222240302.http://www.tandfonline.com/doi/abs/10.2753/MIS0742-1222240302.

Pereira, Rui, Marco Couto, Joao Saraiva, Jacome Cunha ja Joao Paulo Fernandes. 2016. “The Influence of the Java Collection Framework on Overall Energy Consumption”. Teoksessa *Proceedings of the 5th International Workshop on Green and Sustainable Software,* 15–21. GREENS ’16. Austin, Texas: ACM. ISBN: 978-1-4503-4161-5. doi:10.1145/2896967.2896968. http://doi.acm.org/10.1145/2896967.2896968.

“Performance Analysis Tool for Heterogeneous Multicore Virtual Platform”. 2011. Viitattu 20. helmikuuta 2016. http://www.openfoundry.org/of/download_path/hmavp/1.1/VPA-User-Manual.pdf.

“Performance Tips | Android Developers”. 2016. Viitattu 23. marraskuuta 2016. https://developer.android.com/training/articles/perf-tips.html.

"Performance Tips | App programming guide for iOS". 2016. Viitattu 23. marraskuuta 2016. https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/PerformanceTips/PerformanceTips.html.

"PICO Fuses - Littelfuse". 2016. Viitattu 3. joulukuuta 2016. http://www.littelfuse.com/products/fuses/surface-mount-fuses/pico-fuses.aspx.

Pierson, Jean-Marc. 2010. "Allocating Resources Greenly: Reducing Energy Consumption or Reducing Ecological Impact?" Teoksessa *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking,* 127–130. e-Energy '10. Passau, Germany: ACM. ISBN: 978-1-4503-0042-1. doi:10.1145/1791314.1791334. http://doi.acm/10.1145/1791314.1791334.

Pinto, Gustavo. 2013. "Do Language Constructs for Concurrent Execution Have Impact on Energy Efficiency?" Teoksessa *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity,* 121–122. SPLASH '13. Indianapolis, Indiana, USA: ACM. ISBN: 978-1-4503-1995-9. doi:10.1145/2508075.2514880. http://doi.acm.org/10.1145/2508075.2514880.

Pinto, Gustavo, Fernando Castor ja Yu David Liu. 2014. "Mining Questions About Software Energy Consumption". Teoksessa *Proceedings of the 11th Working Conference on Mining Software Repositories,* 22–31. MSR 2014. Hyderabad, India: ACM. ISBN: 978-1-4503-2863-0. doi:10.1145/2597073.2597110. http://doi.acm.org/10.1145/2597073.2597110.

"platform_frameworks_base/BatteryStatsImpl.java at master - android/platform_ frameworks_base GitHub". 2015. Viitattu 20. huhtikuuta 2015. https://github.com/android/platform_frameworks_base/blob/master/core/java/com/android/internal/os/BatteryStatsImpl.java.

"Power Profiles for Android". 2015. Viitattu 20. huhtikuuta 2015. https://source.android.com/devices/tech/power/index.html.

"PowerTutor". 2011. Viitattu 22. toukokuuta 2016. http://ziyang.eecs.umich.edu/projects/powertutor/.

"ProGuard". 2016. Viitattu 23. marraskuuta 2016. http://proguard.sourceforge.net/.

"Protocol Buffers | Google Developers". 2016. Viitattu 26. marraskuuta 2016. https://developers.google.com/protocol-buffers/.

"QEMU". 2016. Viitattu 20. helmikuuta 2016. http://wiki.qemu.org/Main_Page.

Qian, Feng, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen ja Oliver Spatscheck. 2011. "Profiling Resource Usage for Mobile Applications: A Cross-layer Approach". Teoksessa *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services,* 321–334. MobiSys '11. Bethesda, Maryland, USA: ACM. ISBN: 978-1-4503-0643-0. doi:10.1145/1999995.2000026. http://doi.acm.org/10.1145/1999995.2000026.

"Qualcomm Quick Charge FAQs". 2016. Viitattu 15. toukokuuta 2016. https://www.qualcomm.com/products/snapdragon/quick-charge/faq.

Ra, Moo-Ryong, Jeongyeup Paek, Abhishek B. Sharma, Ramesh Govindan, Martin H. Krieger ja Michael J. Neely. 2010. "Energy-delay Tradeoffs in Smartphone Applications". Teoksessa *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services,* 255–270. MobiSys '10. San Francisco, California, USA: ACM. ISBN: 978-1-60558-985-5. doi:10.1145/1814433.1814459. http://doi.acm.org/10.1145/1814433.1814459.

Rahman, Shah Faizur, Jichi Guo ja Qing Yi. 2011. "Automated Empirical Tuning of Scientific Codes for Performance and Power Consumption". Teoksessa *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers,* 107–116. HiPEAC '11. Heraklion, Greece: ACM. ISBN: 978-1-4503-0241-8. doi:10.1145/1944862.1944880. http://doi.acm/10.1145/1944862.1944880.

Ravindranath, Lenin, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller ja Shahin Shayandeh. 2012. “AppInsight: Mobile App Performance Monitoring in the Wild”. Teoksessa *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation,* 107–120. OSDI’12. Hollywood, CA, USA: USENIX Association. ISBN: 978-1-931971-96-6. http://dl.acm.org/citation.cfm?id=2387880.2387891.

“Renewable energy - Data Centers - Google”. 2016. Viitattu 4. marraskuuta 2016. https://www.google.com/about/datacenters/renewable/.

Rice, A., ja S. Hay. 2010. “Decomposing power measurements for mobile devices”. Teoksessa *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on,* 70–78. Maaliskuu. doi:10.1109/PERCOM.2010.5466991.

“Robotium Tech”. 2016. Viitattu 18. marraskuuta 2016. http://robotium.com/.

Rojas, Isabel K. Villanes, Silvia Meireles ja Arilo Claudio Dias-Neto. 2016. “Cloud-Based Mobile App Testing Framework: Architecture, Implementation and Execution”. Teoksessa *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing,* 10:1–10:10. SAST. Maringa, Parana, Brazil: ACM. ISBN: 978-1-4503-4766-2. doi:10.1145/2993288.2993301. http://doi.acm.org/10.1145/2993288.2993301.

Roy, Arjun, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières ja Nickolai Zeldovich. 2011. “Energy Management in Mobile Devices with the Cinder Operating System”. Teoksessa *Proceedings of the Sixth Conference on Computer Systems,* 139–152. EuroSys ’11. Salzburg, Austria: ACM. ISBN: 978-1-4503-0634-8. doi:10.1145/1966445.1966459. http://doi.acm.org/10.1145/1966445.1966459.

“Runkeeper Go”. 2016. Viitattu 5. joulukuuta 2016. https://runkeeper.com/go.

Sahin, Cagri, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock ja Kristina Winbladh. 2012. “Initial Explorations on Design Pattern Energy Usage”. Teoksessa *Proceedings of the First International Workshop on Green and Sustainable Software,* 55–61. GREENS ’12. Zurich, Switzerland: IEEE Press. ISBN: 978-1-4673-1832-7. http://dl.acm/citation.cfm?id=2663779.2663789.

Sahin, Cagri, Lori Pollock ja James Clause. 2014. “How Do Code Refactorings Affect Energy Usage?” Teoksessa *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement,* 36:1–36:10. ESEM ’14. Torino, Italy: ACM. ISBN: 978-1-4503-2774-9. doi:10.1145/2652524.2652538. http://doi.acm.org/10.1145/2652524.2652538.

“SailfishOS first application”. 2016. Viitattu 2. helmikuuta 2016. https://wiki.qt.io/SailfishOS_first_application.

Sakamoto, Takeshi, Takashi Yamada, Mamoru Mukuno, Yoshifumi Matsushita, Yasoo Harada ja Hiroto Yasuura. 2002. “Power Analysis Techniques for SoC with Improved Wiring Models”. Teoksessa *Proceedings of the 2002 International Symposium on Low Power Electronics and Design,* 259–262. ISLPED ’02. Monterey, California, USA: ACM. ISBN: 1-58113-475-4. doi:10.1145/566408.566476. http://doi.acm/10.1145/566408.566476.

Schulman, Aaron, Thomas Schmid, Prabal Dutta ja Neil Spring. 2011. “Demo: Phone Power Monitoring with BattOr”. Teoksessa *In the 17th ACM international conference on Mobile computing and networking, MobiCom,* nide 11.

“Sensors Overview | Android Developers”. 2016. Viitattu 18. marraskuuta 2016. https://developer.android.com/guide/topics/sensors/sensors_overview.html.

“Service | Android Developers”. 2016. Viitattu 9. joulukuuta 2016. https://developer.android.com/guide/components/services.html.

Shakshuki, Elhadi M., Qing Gu, Patricia Lago, Henry Muccini ja Simone Potenza. 2013. “The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013) A Categorization of Green Practices Used by Dutch Data Centers”. *Procedia Computer Science* 19:770–776. ISSN: 1877-0509. doi:http://dx.doi.org/10.1016/j.procs.2013.06.101. http://www.sciencedirect.com/science/article/pii/S1877050913007096.

Shye, Alex, Benjamin Scholbrock ja Gokhan Memik. 2009. “Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures”. Teoksessa *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture,* 168–178. MICRO 42. New York, New York: ACM. ISBN: 978-1-60558-798-1. doi:10.1145/1669112.1669135. http://doi.acm/10.1145/1669112.1669135.

Smith, Jim, ja Ravi Nair. 2005. *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design).* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558609105.

“Snapdragon 805 Processor Product Brief”. 2016. Viitattu 18. marraskuuta 2016. https://www.qualcomm.com/documents/snapdragon-805-processor-product-brief.

“Snapdragon 810 Mobile Processor (8 Core) | Qualcomm”. 2016. Viitattu 24. toukokuuta 2016. https://www.qualcomm.com/products/snapdragon/processors/810.

Souppaya, Murugiah, ja Karen A. Scarfone. 2013. *SP 800-124r1. Guidelines for Managing the Security of Mobile Devices in the Enterprise.* Tekninen raportti. Gaithersburg, MD, United States. http://dx.doi.org/10.6028/NIST.SP.800-124r1.

Sunwoo, Dam, W. Wang, M. Ghosh, C. Sudanthi, G. Blake, C.D. Emmons ja N.C. Paver. 2013. “A structured approach to the simulation, analysis and characterization of smartphone applications”. Teoksessa *Workload Characterization (IISWC), 2013 IEEE International Symposium on,* 113–122. Syyskuu. doi:10.1109/IISWC.2013.6704677.

Tan, Kiat Wee, Tadashi Okoshi, Archan Misra ja Rajesh Krishna Balan. 2013. “FOCUS: A Usable & Effective Approach to OLED Display Power Management”. Teoksessa *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing,* 573–582. UbiComp ’13. Zurich, Switzerland: ACM. ISBN: 978-1-4503-1770-2. doi:10.1145/2493432.2493445. http://doi.acm.org/10.1145/2493432.2493445.

Tan, Tat Kee, A. Raghunathan, G. Lakshminarayana ja N.K. Jha. 2002. “High-level energy macromodeling of embedded software”. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 21, numero 9 (syyskuu): 1037–1050. ISSN: 0278-0070. doi:10.1109/TCAD.2002.801094.

Tarkoma, Sasu, Matti Siekkinen, Eemil Lagerspetz ja Yu Xiao. 2014. *Smartphone Energy Consumption: Modeling and Optimization.* Cambridge University Press. ISBN: 9781107042339.

Teehan, Paul, ja Milind Kandlikar. 2013. “Comparing Embodied Greenhouse Gas Emissions of Modern Computing and Electronics Products”. PMID: 23458212, *Environmental Science & Technology* 47 (9): 3997–4003. doi:10.1021/es303012r. eprint: http://dx.doi.org/10.1021/es303012r. http://dx.doi.org/10.1021/es303012r.

“TelephonyManager.java - Git at Google”. 2016. Viitattu 30. marraskuuta 2016. https://android.googlesource.com/platform/frameworks/base.git/+/nougat-release/telephony/java/android/telephony/TelephonyManager.java.

“Test with the Microsoft Emulator for Windows 10 Mobile”. 2016. Viitattu 2. helmikuuta 2016. https://msdn.microsoft.com/en-us/library/windows/apps/mt188754.aspx.

“The History and Future of Internet Traffic”. 2015. Viitattu 4. marraskuuta 2016. http://blogs.cisco.com/sp/the-history-and-future-of-internet-traffic.

Thompson, Chris, Douglas C. Schmidt, Hamilton A. Turner ja Jules White. 2011. “Analyzing Mobile Application Software Power Consumption via Model-driven Engineering.” Teoksessa *PECCS,* toimittanut César Benavente-Peces ja Joaquim Filipe, 101–113. SciTePress. ISBN: 978-989-8425-48-5. http://dblp.uni-trier.de/db/conf/peccs/peccs2011.html#ThompsonSTW11.

Tiwari, Vivek, Sharad Malik ja Andrew Wolfe. 1994. “Power Analysis of Embedded Software: A First Step Towards Software Power Minimization”. *IEEE Trans. Very Large Scale Integr. Syst.* (Piscataway, NJ, USA) 2, numero 4 (joulukuu): 437–445. ISSN: 1063-8210. doi:10.1109/92.335012. http://dx.doi/10.1109/92.335012.

"Trepn Profiler". 2016. Viitattu 20. huhtikuuta 2016. https://developer.qualcomm.com/software/trepn-power-profiler.

Tu, Chia-Heng, Hui-Hsin Hsu, Jen-Hao Chen, Chun-Han Chen ja Shih-Hao Hung. 2014. "Performance and Power Profiling for Emulated Android Systems". *ACM Trans. Des. Autom. Electron. Syst.* (New York, NY, USA) 19, numero 2 (maaliskuu): 10:1–10:25. ISSN: 1084-4309. doi:10.1145/2566660. http://doi.acm/10.1145/2566660.

"UMFT234XF Datasheet - FTDI". 2016. Viitattu 1. joulukuuta 2016. http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_UMFT234XF.pdf.

"Using the Emulator | Android Developers". 2016. Viitattu 2. helmikuuta 2016. http://developer.android.com/tools/devices/emulator.html.

"Using the iOS Simulator to test and debug AIR applications". 2016. Viitattu 2. helmikuuta 2016. http://www.adobe.com/devnet/air/articles/ios-simulator.html.

Vallina-Rodriguez, Narseo, Pan Hui, Jon Crowcroft ja Andrew Rice. 2010. "Exhausting Battery Statistics: Understanding the Energy Demands on Mobile Handsets". Teoksessa *Proceedings of the Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds,* 9–14. MobiHeld '10. New Delhi, India: ACM. ISBN: 978-1-4503-0197-8. doi:10.1145/1851322.1851327. http://doi.acm.org/10.1145/1851322.1851327.

Varma, Ankush, Eric Debes, Igor Kozintsev, Paul Klein ja Bruce Jacob. 2008. "Accurate and Fast System-level Power Modeling: An XScale-based Case Study". *ACM Trans. Embed. Comput. Syst.* (New York, NY, USA) 7, numero 3 (toukokuu): 25:1–25:20. ISSN: 1539-9087. doi:10.1145/1347375.1347378. http://doi.acm/10.1145/1347375.1347378.

Vergara, Ekhiotz Jon, Simin Nadjm-Tehrani ja Mihails Prihodko. 2014. “EnergyBox: Disclosing the wireless transmission energy cost for mobile devices”. Special Issue on Selected papers from EE-LSDS2013 Conference, *Sustainable Computing: Informatics and Systems* 4 (2): 118–135. ISSN: 2210-5379. doi:http://dx.doi.org/10.1016/j.suscom.2014.03.008. http://www.sciencedirect.com/science/article/pii/S2210537914000195.

Waag, Wladislaw, Christian Fleischer ja Dirk Uwe Sauer. 2014. “Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles”. *Journal of Power Sources* 258:321–339. ISSN: 0378-7753. doi:http://dx.doi.org/10.1016/j.jpowsour.2014.02.064. http://www.sciencedirect.com/science/article/pii/S0378775314002572.

Wang, Chengke, Fengrun Yan, Yao Guo ja Xiangqun Chen. 2013. “Power Estimation for Mobile Applications with Profile-driven Battery Traces”. Teoksessa *Proceedings of the 2013 International Symposium on Low Power Electronics and Design,* 120–125. ISLPED ’13. Beijing, China: IEEE Press. ISBN: 978-1-4799-1235-3. http://dl.acm/citation.cfm?id=2648668.2648697.

Wang, Jingtian, Guoquan wu, Xiaoquan Wu ja Jun Wei. 2012. “Detect and Optimize the Energy Consumption of Mobile App Through Static Analysis: An Initial Research”. Teoksessa *Proceedings of the Fourth Asia-Pacific Symposium on Internetware,* 22:1–22:5. Internetware ’12. Qingdao, China: ACM. ISBN: 978-1-4503-1888-4. doi:10.1145/2430475.2430497. http://doi.acm.org/10.1145/2430475.2430497.

Wilke, C., S. Richly, S. Gotz, C. Piechnick ja U. Assmann. 2013. “Energy Consumption and Efficiency in Mobile Applications: A User Feedback Study”. Teoksessa *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing,* 134–141. Elokuu. doi:10.1109/GreenCom-iThings-CPSCom.2013.45.

Wilke, Claas, Sebastian Götz ja Sebastian Richly. 2013. “JouleUnit: A Generic Framework for Software Energy Profiling and Testing”. Teoksessa *Proceedings of the 2013 Workshop on Green in/by Software Engineering,* 9–14. GIBSE ’13. Fukuoka, Japan: ACM. ISBN: 978-1-4503-1866-2. doi:10.1145/2451605.2451610. http://doi.acm.org/10.1145/2451605.2451610.

Wilke, Claas, Christian Piechnick, Sebastian Richly, Georg Püschel, Sebastian Götz ja Uwe Aßmann. 2013. “Comparing Mobile Applications’ Energy Consumption”. Teoksessa *Proceedings of the 28th Annual ACM Symposium on Applied Computing,* 1177–1179. SAC ’13. Coimbra, Portugal: ACM. ISBN: 978-1-4503-1656-9. doi:10.1145/2480362.2480583. http://doi.acm/10.1145/2480362.2480583.

Woodside, M., G. Franks ja D. C. Petriu. 2007. “The Future of Software Performance Engineering”. Teoksessa *Future of Software Engineering, 2007. FOSE ’07,* 171–187. Toukokuu. doi:10.1109/FOSE.2007.32.

“Worldwide Shipments of Slate Tablets Continue to Decline While Detachable Tablets Climb to New High, According to IDC”. 2016. Viitattu 14. marraskuuta 2016. https://www.idc.com/getdoc.jsp?containerId=prUS40990116.

“WSL...18 - Vishay”. 2016. Viitattu 3. joulukuuta 2016. http://www.vishay.com/docs/31057/wslhigh.pdf.

“Xamarin”. 2016. Viitattu 18. marraskuuta 2016. https://www.xamarin.com/.

“Xamarin Test Cloud”. 2016. Viitattu 13. joulukuuta 2016. https://www.xamarin.com/test-cloud.

Xiao, Yu, R. Bhaumik, Zhirong Yang, M. Siekkinen, P. Savolainen ja A. Yla-Jaaski. 2010. “A System-Level Model for Runtime Power Estimation on Mobile Devices”. Teoksessa *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int’l Conference on Int’l Conference on Cyber, Physical and Social Computing (CPSCom),* 27–34. Joulukuu. doi:10.1109/GreenCom-CPSCom.2010.114.

Xu, Fengyuan, Yunxin Liu, Qun Li ja Yongguang Zhang. 2013. “V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics”. Teoksessa *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation,* 43–56. nsdi’13. Lombard, IL: USENIX Association. http://dl.acm.org/citation.cfm?id=2482626.2482633.

Yoon, Chanmin, Dongwon Kim, Wonwoo Jung, Chulkoo Kang ja Hojung Cha. 2012. “AppScope: Application Energy Metering Framework for Android Smartphones Using Kernel Activity Monitoring”. Teoksessa *Proceedings of the 2012 USENIX Conference on Annual Technical Conference,* 36–36. USENIX ATC’12. Boston, MA: USENIX Association. http://dl.acm/citation.cfm?id=2342821.2342857.

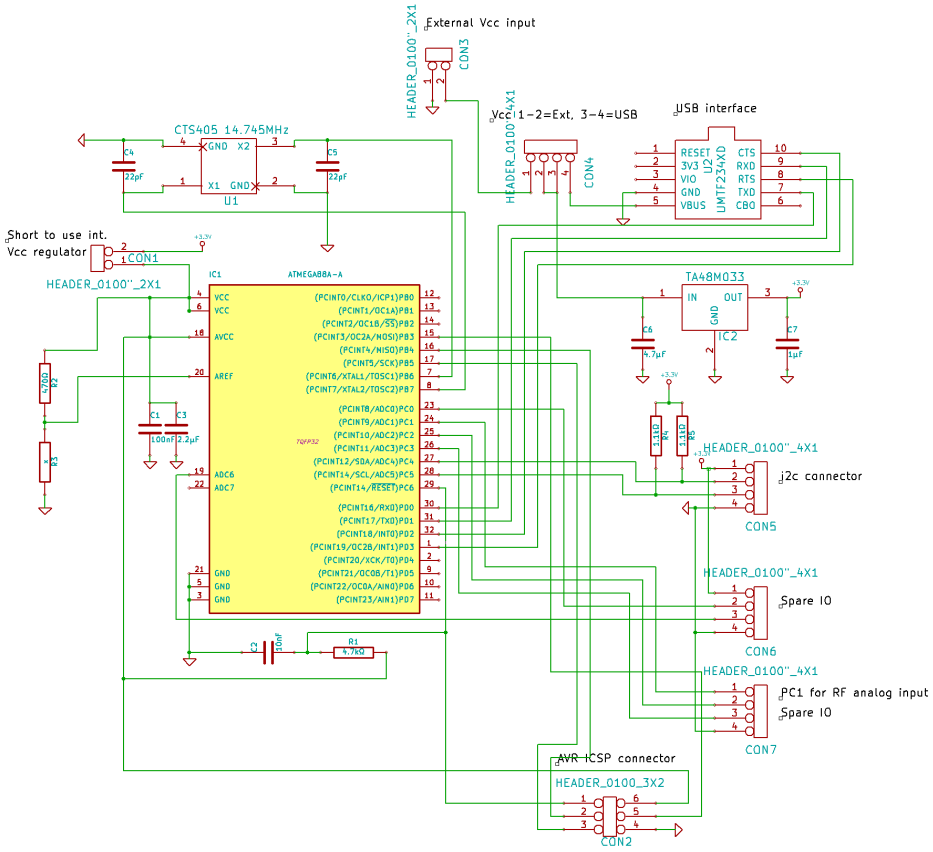
Zhang, Lide, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao ja Lei Yang. 2010. “Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones”. Teoksessa *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* 105–114. CODES/ISSS ’10. Scottsdale, Arizona, USA: ACM. ISBN: 978-1-60558-905-3. doi:10.1145/1878961.1878982. http://doi.acm.org/10.1145/1878961.1878982.

Zheng, Rui, J. Velamala, V. Reddy, V. Balakrishnan, E. Mintarno, S. Mitra, Srikanth Krishnan ja Yu Cao. 2009. “Circuit aging prediction for low-power operation”. Teoksessa *Custom Integrated Circuits Conference, 2009. CICC ’09. IEEE,* 427–430. Syyskuu. doi:10.1109/CICC.2009.5280814.

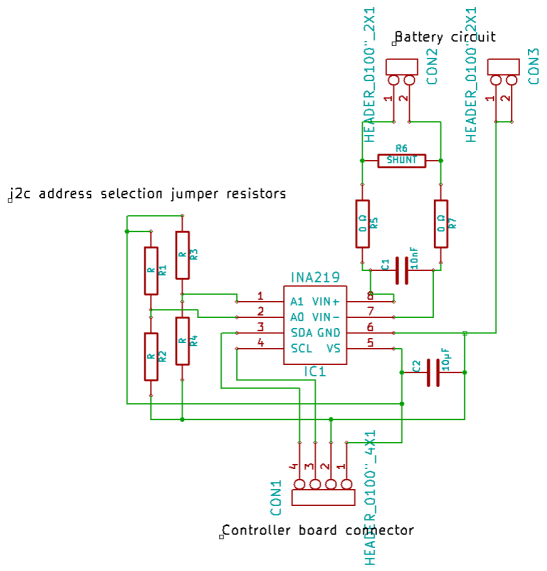
Liitteet

A Kehitetyn energiankulutuksen mittalaitteen kytkentäkaaviot

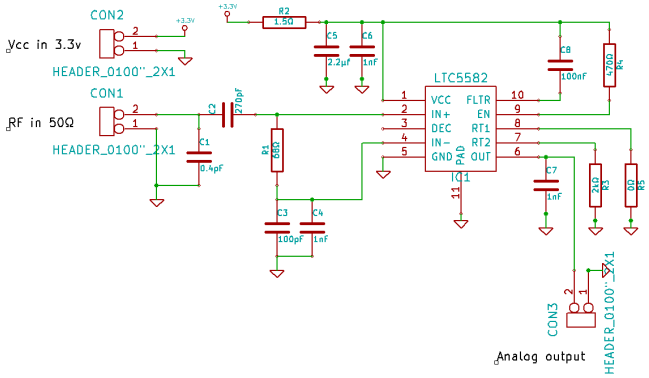
Energiankulutuksen mittaamiseen kehitetty laite toteutettiin kolmena erillisenä moduulina, joista ensimmäinen sisältää ATMega88 mikro-ohjaimen ja USB-TTL-sarjamuuntimen, toinen INA219 tehonmittausrajapinnan ja kolmas LTC5582 RF-tasomittapiirin (“UMFT234XF Datasheet - FTDI” 2016; “INA219 Datasheet - Texas Instruments” 2016; “LTC5582 Datasheet - Linear Rechnology” 2016). Kukin moduuli valmistettiin omalle piirilevylleen, jotka yhdistettiin kokonaisuudeksi kaapeleilla. Kytkentäkaavio on esitetty alla moduuleiksi jaoteltuna.



Kuvio 6: Mittalaitteen mikro-ohjainkortin kytkentäkaavio.



Kuvio 7: Mittalaitteen tehonmittausrajapinnan kytkentäkaavio.



Kuvio 8: Mittalaitteen radiotaajuisen tehon mittausrajapinnan kytkentäkaavio.

B Kehitetyn energiankulutuksen mittalaitteen lähdekoodi

Seuraavat lähdekoodit on toteutettu Atmel ATmega88-mikro-ohjaimelle (“ATMega88” 2016) C-ohjelmointikielellä käyttäen Atmelin AVR Libc -kirjastoa (“AVR Libc reference manual” 2016). Lähdekoodi käännettiin binääriksi ja ohjelmoitiin mikro-ohjaimelle Atmel Studio -kehitysympäristössä käyttäen AVR ISP -ohjelmointilaitetta, jolla piirin ohjelmamuisti voidaan päivittää sen ollessa osana valmista kytkentää (“Atmel Studio 7” 2016; “AVRISP mkII” 2016).

Lähdekooditiedostoista **AM88_current_monitor** sisältää ohjelman pääsilmukan ja funktiot,

jotka käsittelevät tulosten keräämistä mikro-ohjaimen A/D-muuntimelta (RF-taso) ja tehonmittausrajapinnoista. Lisäksi se vastaanottaa ja käsittelee mittalaitteen konfiguraatioviestit, kerää jännitettä ja virtaa kuvaavat arvot INA219-piireiltä ja lähettää mittaustulokset binäärimuodossa eteenpäin mikro-ohjaimen sarjaväylän välityksellä. Tiedosto **INA219_driver** sisältää INA219-piirien käsittelyä tukevat funktiot, jotka riippuvat tiedoston **TWI_driver** sisältämistä, I2C-väylän käsittelyn mahdollistavista funktiosta.

Ohjelma lukee jokaiseen lähetettävään mittausnäytteeseen mikro-ohjaimen A/D-muuntimen arvon ja valinnaisesti toisesta INA219-tehonmittausrajapinnasta virran ja/tai jännitteen. Molemmat lähteet tekevät muunnoksia vapaasti, eikä niiden ajoitusta ole synkronoitu toisiinsa, koska RF-tasoa ei käytetä kvantitatiivisesti. Kerätyt näytteet lähetetään sarjaväylän ja sarja-USB-muuntimen kautta tietokoneelle binääriprotokollalla, jossa näytteet ovat lähdemuodossaan ja niiden ympärille lisätään viestin rajat sekä kenttämäärän kuvaavat alku- ja lopputunnisteet.

AM88_current_monitor.h

tabsize

```
/*
 * AM88_current_monitor.h
 *
 * Author: Veli-Mikko Puupponen
 */


#ifndef AM88_CURRENT_MONITOR_H_
#define AM88_CURRENT_MONITOR_H_
void DEVICE_Init();
void USART_Init();
void USART_Transmit( unsigned char data );
unsigned char USART_Receive();
void ADC_Initialize_registers();
void ADC_Start_Freerun();
void ADC_Value_Transmit();
void Update_Chip_Configuration();
void Read_Active_Chip();
void Read_Active_Chip_Measurement_Register(uint8_t setpointer, uint8_t pointer, uint16_t *result);
void Send_Latest_Measurements();
void Send_Start_Header();
void Send_End_Header();


#endif /* AM88_CURRENT_MONITOR_H_ */
```

AM88_current_monitor.c

tabsize

```
/*
```

```
 * AM88_current_monitor.c
 * This file contains the main loop for sampling RF level ADC
 * and INA219 power measurement ICs. Results are sent over UART
 * in a binary format. Sampling settings are received over UART
 * before starting measurements.
 *
 *
 * Author: Veli-Mikko Puupponen
 *
 * NOTE: In the ina219 datasheet page 30 reads "2.) Reading the Power Register".
 * It should be "the voltage register".
 */

#define F_CPU 14745600UL
#define USART_BAUDRATE 921600
#define UBRR ((F_CPU / 16) / USART_BAUDRATE) - 1
/* i2c Addresses to the INA219 chips stored with R/W set to zero */
#define CHIP_0_ADDRESS 0b10000010
#define CHIP_1_ADDRESS 0b10001000
#define TWI_FREQ_DIVIDER 0x0B
#define TWI_FREQ_BASE 0x00
#define CONFIG_ID_BIT 0x80
#define SYNC_BYTE 0xFF

#include <avr/io.h>
#include <avr/interrupt.h>
#include "AM88_current_monitor.h"
#include "TWI_driver.h"
#include "INA219_driver.h"

volatile uint8_t adc_value_available;
volatile uint8_t adc_value_low;
volatile uint8_t adc_value_high;
uint16_t last_acd_value;
uint8_t chip_selection;
uint8_t shunt_active;
uint8_t voltage_active;
uint8_t active_chip_address;
uint8_t chip_status;
uint16_t last_voltage;
uint16_t last_shunt;
volatile uint8_t chip_configuration_available;
volatile uint8_t chip_configuration_high;
volatile uint8_t chip_configuration_low;
volatile uint8_t chip_active;
uint8_t last_start_header;
uint8_t last_chip_address;


/*
 * Data is sent in a variable length format:
 * No INA219 enabled: Start, ADC1, ADC0, End
 * Only bus or shunt voltage: Start, ADC1, ADC0, V1, V0, End
 * Both bus and shunt voltage: Start, ADC1, ADC0, SV1, SV0, BV1, BV0, End
 * Where start is 0x02, 0x04 or 0x06 based on the number of data fields
 * End is start^0xff
 * ADC0..1 is the raw ADC value from RF level measurement
 * SV0..1 is the raw shunt voltage value from the selected INA219
 * BV0..1 is the raw bus voltage value from the selected INA219
*/
int main(void)
{
        DEVICE_Init();
        while(1)
```

```
        {
                while ( adc_value_available != 1 )
                ;
                Send_Start_Header();
                ADC_Value_Transmit();
                Read_Active_Chip();
                Send_Latest_Measurements();
                Send_End_Header();
                Update_Chip_Configuration();
        }
}


void DEVICE_Init()
{
        last_chip_address = 0x00;
        DDRC = 0b00000001;
        USART_Init();
        TWI_Init(TWI_FREQ_DIVIDER, TWI_FREQ_BASE);
        ADC_Initialize_registers();
        sei();
        ADC_Start_Freerun();
}

void USART_Init()
{
        UBRR0H = (uint8_t)(UBRR>>8);
        UBRR0L = (uint8_t)UBRR;
        /* Enable send, receive and receive interrupt */
        UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0);
        /* 8data, 1stop bits */
        UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
        /* Double speed operation if enabled */
        //UCSR0A |= 0x02;
}

void USART_Transmit(uint8_t data)
{
        while ( !( UCSR0A & (1<<UDRE0)) )
        ;
        UDR0 = data;
}

unsigned char USART_Receive()
{
        while ( !( UCSR0A & (1<<RXC0)) )
        ;
        return UDR0;
}


/* ADC frequency prescaler 128 leads to 115.2kHz and in overall ~9ksps */
/* Uses free running conversion mode*/
/* ADC input is initialized to pin PC1*/
void ADC_Initialize_registers()
{
        // ADC1 digital disable
        DIDR0 = (1<<ADC1D);
        // Use external AREF, convert from ADC1 input PC1
        ADMUX = (0<<REFS1) | (0<<REFS0) | (1<<MUX0);
        // Enable ADC, Auto triggering, Interrupt mode, ADC clock prescaler 128
        ADCSRA = (1<<ADEN) | (1<<ADATE) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
        // Set auto trigger source to free running mode
        ADCSRB = (0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);
```

```
}

// Start free running ADC conversion
void ADC_Start_Freerun()
{
        ADCSRA |= (1<<ADSC);
}

void ADC_Value_Transmit()
{
        uint8_t data_low;
        uint8_t data_high;
        cli();
        data_low = adc_value_low;
        data_high = adc_value_high;
        last_acd_value = ((adc_value_high << 0x08) | adc_value_low);
        sei();
        adc_value_available = 0;
        USART_Transmit(data_high);
        USART_Transmit(data_low);
}

ISR(ADC_vect)
{
        adc_value_low = ADCL;
        adc_value_high = ADCH;
        adc_value_available = 1;
}

ISR(USART_RX_vect)
{
        cli();
        uint8_t buffer = UDR0;
        sei();
        if(chip_configuration_high != 0x00)
        {
                chip_configuration_low = buffer;
                chip_configuration_available = 0x01;
        }
        if(chip_configuration_high == 0x00)
        {

                if( (buffer & CONFIG_ID_BIT) != 0x00 )
                {
                        PORTC ^= _BV(PC0);
                        chip_configuration_high = buffer;
                }
        }
}

// Parses the device configuration from 2 octets. Format:
// HI: 0b 1   IS   BV   SV   SV   BD3  BD2  BD1
// LO: 0b BD0 SD3  SD2  SD1  SD0  CV2  CV1  CV0
// Where IS = input chip, 0 = 1, 1 = 2
// BV = bus voltage, 0 = 16V, 1 = 32V
// SV = shunt voltage, 0 = 40mV, 1 = 80mV, 2 = 160mV, 3 = 320mV
// BD0..3 = bus bit depth, 0 = 9bit ... 3 = 12bit
// SD0..3 = shunt bit depth, 0 = 9bit ... 3 = 12bit
// CV0..2 = configuration value, 0 = RF only, 1 = Bus, 2 = Shunt,bus
void Update_Chip_Configuration()
{
        if(chip_configuration_available == 0x00)
        {
                return;
```

```
        }
        chip_selection = ((chip_configuration_high & 0x40) >> 6);
        chip_configuration_high = (chip_configuration_high & 0x3f);
        if(chip_selection == 0)
        {
                active_chip_address = CHIP_0_ADDRESS;
        }
        else
        {
                active_chip_address = CHIP_1_ADDRESS;
        }
        uint8_t write_result = DEVICE_WRITE(active_chip_address, DEVICE_CONFIGURATION_POINTER, chip_configuration_high,
    chip_configuration_low);
        if(write_result == 0x00)
        {
                shunt_active = (0x01 & chip_configuration_low);
                voltage_active = ((0x02 & chip_configuration_low) >> 0x01);
                chip_active = 0x01;
                chip_configuration_high = 0x00;
                chip_configuration_low = 0x00;
                chip_configuration_available = 0x00;
                last_chip_address = 0x00;
        }
        else
        {
                chip_active = 0x00;
                shunt_active = 0x00;
                voltage_active = 0x00;
        }
}


void Read_Active_Chip()
{
        if(chip_active == 0x00)
        {
                return;
        }
        if(shunt_active != 0x00)
        {
                Read_Active_Chip_Measurement_Register(0x01, DEVICE_SHUNT_POINTER, &last_shunt);
        }
        if(voltage_active != 0x00)
        {
                Read_Active_Chip_Measurement_Register(0x01, DEVICE_BUS_POINTER, &last_voltage);
        }
}

void Read_Active_Chip_Measurement_Register(uint8_t setpointer, uint8_t pointer, uint16_t *result)
{
        uint8_t status;
        uint16_t register_value;
        if(chip_status == 0x00)
        {
                if(setpointer == 0x00)
                {
                        status = 0x00;
                }
                else
                {
                        status = DEVICE_SET_REGISTER_POINTER(active_chip_address, pointer);
                }
                if(status == 0x00)
                {
```

```
                        status = DEVICE_READ(active_chip_address, &register_value);
                        *result = register_value;
                        last_chip_address = pointer;
                }
                chip_status = status;
        }
}


void Send_Latest_Measurements()
{
        if(shunt_active != 0x00)
        {
                USART_Transmit(last_shunt >> 0x08);
                USART_Transmit(last_shunt);
        }
        if(voltage_active != 0x00)
        {
                USART_Transmit(last_voltage >> 0x08);
                USART_Transmit(last_voltage);
        }
}

void Send_Start_Header()
{
        last_start_header = 0x02;
        if(chip_active != 0x00){
                if(shunt_active != 0x00)
                {
                        last_start_header += 0x02;
                }
                if(voltage_active != 0x00)
                {
                        last_start_header += 0x02;
                }
        }
        USART_Transmit(last_start_header);
}

void Send_End_Header()
{
        USART_Transmit(last_start_header ^ 0xff);
}
```

INA219_driver.h

tabsize

```
/*
 * INA219_driver.h
 *
 * Author: Veli-Mikko Puupponen
 */


#ifndef INA219_DRIVER_H_
#define INA219_DRIVER_H_

extern const uint8_t DEVICE_CONFIGURATION_POINTER;
extern const uint8_t DEVICE_SHUNT_POINTER;
extern const uint8_t DEVICE_BUS_POINTER;
uint8_t DEVICE_WRITE(uint8_t target, uint8_t register_pointer, uint8_t data_high, uint8_t data_low);
uint8_t DEVICE_READ(uint8_t target, uint16_t *data);
```

```
uint8_t DEVICE_SET_REGISTER_POINTER(uint8_t target, uint8_t register_pointer);
uint8_t RETURN_RESET_STATUS_REGISTER();



#endif /* INA219_DRIVER_H_ */
```

INA219_driver.c

tabsize

```
/*
 * INA219_driver.c
 *
 * This file contains functions for interacting with the INA219
 * over the i2c/TWI bus using the TWI_driver.
 *
 * Author: Veli-Mikko Puupponen
 */

#include <avr/io.h>
#include "INA219_driver.h"
#include "TWI_driver.h"

const uint8_t DEVICE_CONFIGURATION_POINTER = 0x00;
const uint8_t DEVICE_SHUNT_POINTER = 0x01;
const uint8_t DEVICE_BUS_POINTER = 0x02;

uint8_t DEVICE_WRITE(uint8_t target, uint8_t register_pointer, uint8_t data_high, uint8_t data_low)
{
	TWI_SEND_START();
	if(ERROR_STATUS_REGISTER != 0x00)
		RETURN_RESET_STATUS_REGISTER();
	TWI_SEND_TARGET_ADDRESS(target, 0x00);
	TWI_SEND_REGISTER_POINTER(register_pointer);
	if(ERROR_STATUS_REGISTER != 0x00)
		RETURN_RESET_STATUS_REGISTER();
	TWI_SEND_DATA(data_high);
	TWI_SEND_DATA(data_low);
	TWI_SEND_STOP();
	if(ERROR_STATUS_REGISTER != 0x00)
		RETURN_RESET_STATUS_REGISTER();
	return 0x00;
}

uint8_t DEVICE_READ(uint8_t target, uint16_t *data)
{
	uint8_t high_octet, low_octet;
	uint16_t complete_data;
	TWI_SEND_START();
	if(ERROR_STATUS_REGISTER != 0x00)
		RETURN_RESET_STATUS_REGISTER();
	TWI_SEND_TARGET_ADDRESS(target, 0x01);
	high_octet = TWI_RECEIVE_DATA(0x01);
	low_octet = TWI_RECEIVE_DATA(0x01);
	TWI_SEND_STOP();
	if(ERROR_STATUS_REGISTER != 0x00)
		RETURN_RESET_STATUS_REGISTER();
	complete_data = (high_octet << 8) | low_octet;
	*data = complete_data;
	return 0x00;
}
```

```
uint8_t DEVICE_SET_REGISTER_POINTER(uint8_t target, uint8_t register_pointer)
{
        TWI_SEND_START();
        TWI_SEND_TARGET_ADDRESS(target, 0x00);
        TWI_SEND_REGISTER_POINTER(register_pointer);
        TWI_SEND_STOP();
        if(ERROR_STATUS_REGISTER != 0x00)
                RETURN_RESET_STATUS_REGISTER();
        return 0x00;
}

uint8_t RETURN_RESET_STATUS_REGISTER()
{
        uint8_t REGISTER_CLONE = ERROR_STATUS_REGISTER;
        ERROR_STATUS_REGISTER = 0x00;
        return REGISTER_CLONE;
}
```

TWI_driver.h

tabsize

```
/*
 * TWI_driver.h
 *
 * Author: Veli-Mikko Puupponen
 */


#ifndef TWI_DRIVER_H_
#define TWI_DRIVER_H_

uint8_t ERROR_STATUS_REGISTER;
void TWI_Init(uint8_t divider, uint8_t pre);
void TWI_SEND_START();
void TWI_SEND_TARGET_ADDRESS(uint8_t address, uint8_t rw);
uint8_t TWI_RECEIVE_DATA(uint8_t ack);
void TWI_SEND_DATA(uint8_t data);
void TWI_SEND_REGISTER_POINTER(uint8_t r_pointer);
void TWI_SEND_STOP();
void TWI_ERROR();
void RESET_ERROR();


#endif /* TWI_DRIVER_H_ */
```

TWI_driver.c

tabsize

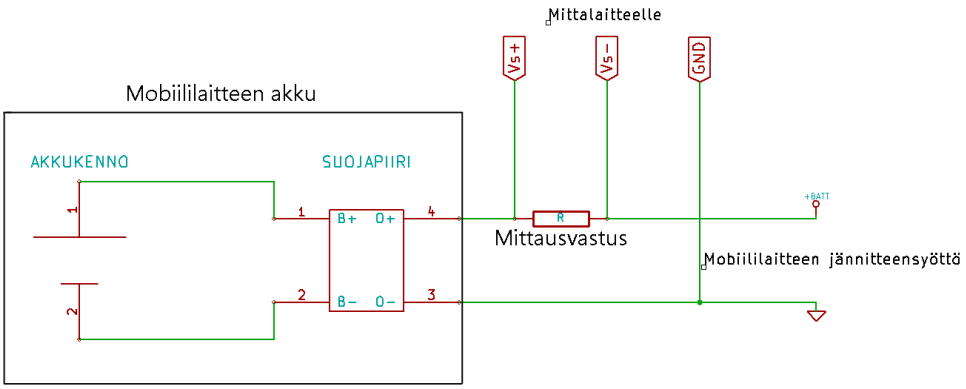
```
/*
 * TWI_driver.c
 *
 * This file contains functions for using the hardware
 * TWI interface on the Atmega88 chip
 *
 * Author: Veli-Mikko Puupponen
 */

#include <avr/io.h>
#include <util/twi.h>
#include "TWI_driver.h"
```

```
#define MT_ST 0x08
#define MT_MST 0x10
#define MT_SLA_ACK 0x18
#define MT_SLA_NACK 0x20
#define MT_DATA_ACK 0x28
#define MT_DATA_NACK 0x30
#define MT_ARB_LOST 0x38
#define MR_SLA_ACK 0x40
#define MR_SLA_NACK 0x48
#define MR_DATA_ACK 0x50
#define MR_DATA_NACK 0x58

uint8_t ERROR_STATUS_REGISTER = 0x00;

void TWI_Init(uint8_t divider, uint8_t pre)
{
        TWBR = divider;
        TWSR = TWSR | (0x03 & pre);
}

void TWI_SEND_START()
{
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        while (!(TWCR & (1<<TWINT)))
                ;
        if ((TWSR & 0xF8) != MT_ST)
                TWI_ERROR();
}

void TWI_SEND_TARGET_ADDRESS(uint8_t address, uint8_t rw)
{
        TWDR = address | (rw & 0x01);
        TWCR = (1<<TWINT) | (1<<TWEN);
        while (!(TWCR & (1<<TWINT)))
                ;
        if ((TWSR & 0xF8) != MT_SLA_ACK && rw == 0x00)
                TWI_ERROR();
        if ((TWSR & 0xF8) != MR_SLA_ACK && rw == 0x01)
                TWI_ERROR();
}

uint8_t TWI_RECEIVE_DATA(uint8_t ack)
{
        uint8_t received_octet;
        if(ack == 0x01)
        {
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
                while(!(TWCR & (1<<TWINT)))
                        ;
        }
        else
        {
                TWCR = (1<<TWINT) | (1<<TWEN);
                while(!(TWCR & (1<<TWINT)))
                        ;
        }
        received_octet = TWDR;
        return received_octet;
}

void TWI_SEND_DATA(uint8_t data)
{
        TWDR = data;
```

```
        TWCR = (1<<TWINT) | (1<<TWEN);
        while (!(TWCR & (1<<TWINT)))
                ;
        if ((TWSR & 0xF8) != MT_DATA_ACK)
                TWI_ERROR();
}

void TWI_SEND_REGISTER_POINTER(uint8_t r_pointer)
{
        TWDR = r_pointer;
        TWCR = (1<<TWINT) | (1<<TWEN);
        while (!(TWCR & (1<<TWINT)))
                ;
        if ((TWSR & 0xF8) != MT_DATA_ACK)
                TWI_ERROR();
}

void TWI_SEND_STOP()
{
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        while(TWCR & (1<<TWSTO))
                ;
}

void TWI_ERROR()
{
        ERROR_STATUS_REGISTER = (TWSR & 0xF8);
        TWI_SEND_STOP();
}

void RESET_ERROR()
{
        ERROR_STATUS_REGISTER = 0x00;
}
```

C Älypuhelimen modifioiminen energiankulutuksen mittauskäyttöön

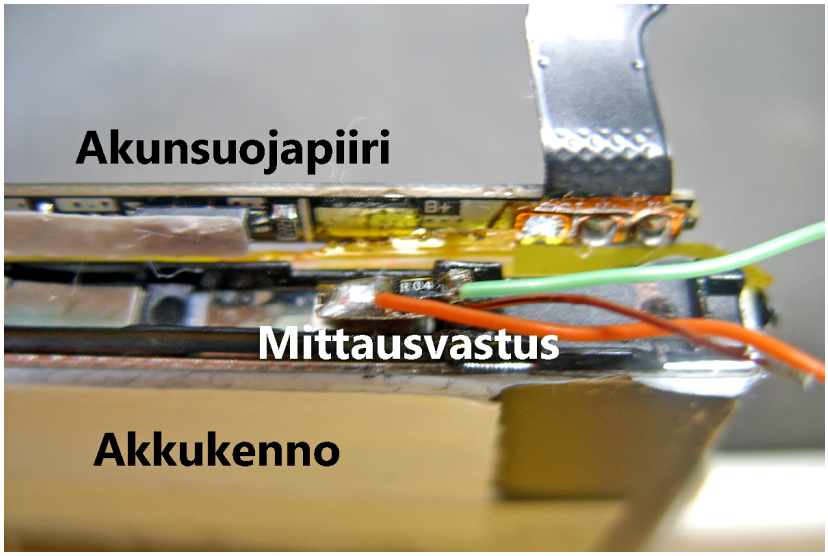
Mobiililaitteiden akut perustuvat käytännössä aina litium-akkukemiaan, jonka energiatiheys ja oikosulkuvirta on korkea, mutta yli- tai alijännitteensietokyky heikko. Näin akut sisältävät akkukennon tai -kennojen lisäksi niihin suoraan, mahdollisimman lyhyillä johtimilla, kytketyn suojapiirin. Tämän elementin tarkoitus on suojata akkua vaurioilta ja sen ympäristöä akun väärinkäyttöön liittyviltä mahdollisilta riskeiltä, erityisesti ylikuumenemiselta tai tulipalolta. Akunsuojapiiri toimii aktiivisena kytkimenä, joka estää lataamasta akkua yli sen maksimijännitteen, purkamasta sitä minimijännitettä tyhjemmäksi tai purkamasta sitä asetettua rajaa suuremmalla virralla (esim. “LM3641 Lithium-Ion Battery Pack Protection Circuit” 2016).



Kuvio 9: Mobiililaitteen akkukennon, akunsuojapiirin ja energiankulutuksen virranmittausvastuksen kytkentä.

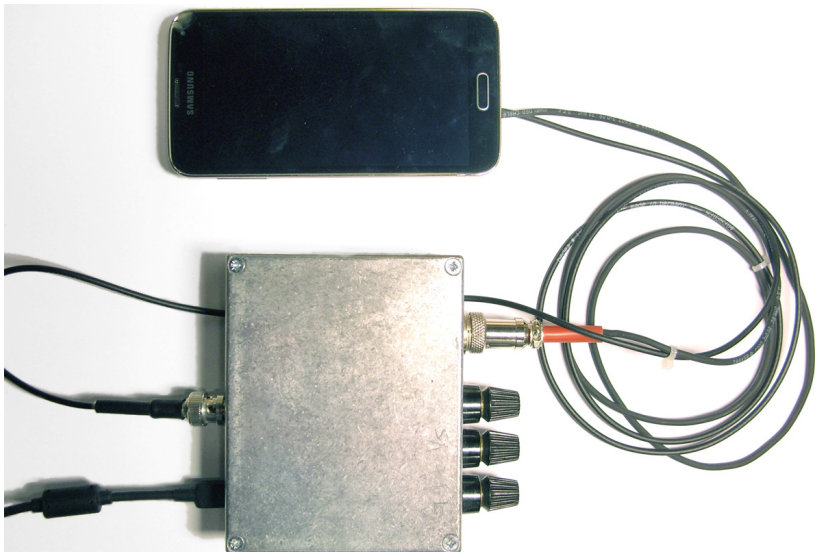
Mobiililaitteiden akut voi olla suunniteltu käyttäjän vaihdettaviksi, tai kuten enenevissä määrin moderneissa laitteissa, myös kiinteästi asennetuiksi ja vain hankalasti irrotettaviksi. Kummassakin tapauksessa niihin on kuitenkin yleensä mahdollista lisätä energiankulutuksen mittaukseen tarvittava mittausvastus, jonka käytännöllinen asennuspaikka on suojapiirin jälkeen, ennen mobiililaitetta, kaavion 9 mukaisesti. Tehon mittaaminen vaatii tietoa sekä jännitteestä että virrasta ja akun tapauksessa molemmat on mitattava, koska jännite ei säily varaustilan muuttuessa samalla tasolla. Virran määrittämiseen tarvitaan virtapiiriin kytkennät

ennen ja jälkeen mittausvastusta vallitsevan jännitteen mittaamiseen. Akkujännitte määritetään puolestaan mittausvastuksen jälkeisen jännitteen ja maapotentiaalin erona, joten kolmas kytkentä on lisättävä akun maapotentiaaliin (“INA219 Datasheet - Texas Instruments” 2016).

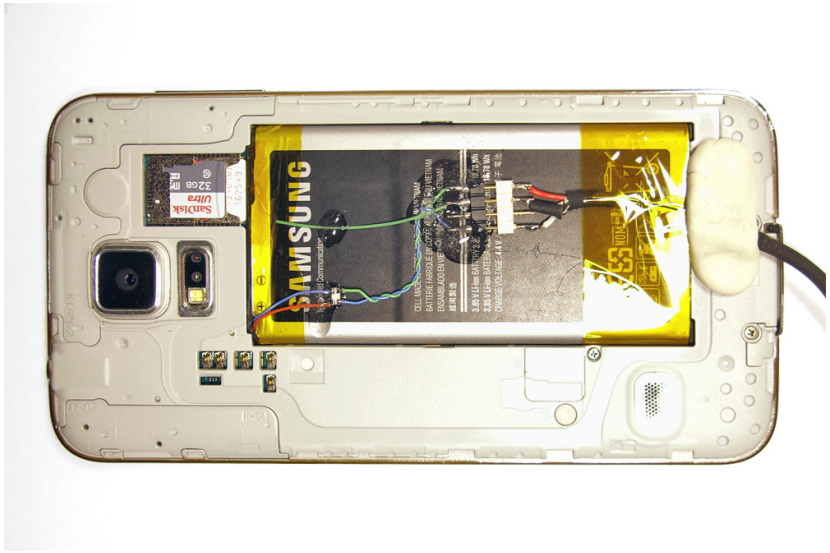


Kuvio 10: Virranmittausvastuksen asentaminen Lumia 925 -älypuhelimen akun sisään.

Tässä työssä mittausvastus ja siihen liittyvät kytkennät lisättiin sekä kiinteään akkuun perustuvaan Lumia 925 -puhelimeen, että vaihdettavaa akkua käyttävään Samsung Galaxy S5 -puhelimeen, joista vain S5-mallia käytettiin suoritetuissa kokeissa. Molemmissa muutos osoittautui mahdolliseksi, mutta Lumian akku oli liimattu kiinni puhelimen runkoon, josta sen irrottaminen ennen muutoksia vaati ylimääräistä työtä. Lumian akussa (mallinumero BL-4YW) oli vapaata tilaa akkukennon ja suojapiirin välissä, joten mittausvastus mahtui kokonaan akun sisään ja vain mittakaapelit tuotiin sen ulkopuolelle (kuva 10). Samsungin akussa (mallinumero EB-BG900BBE) vastaavaa tilaa ei ollut, joten vastus asennettiin akun ulkopintaan. Molemmissa asennuksissa kytkentä tehtiin ennen akunsuojapiiriä, joka on sellaisenaan turvallisuuden kannalta huonompi ratkaisu, koska suojapiiri ei voi estää akun liiallista purkamista mittauskytkennän kautta tai suojata oikosululta. Varsinkin tässä tapauksessa on tärkeää asentaa heti mittausvastuksen läheisyyteen, molempiin siitä lähteviin kytkentöihin nopeat sulakkeet, jotka estävät mahdolliset oikosulkuvirrat mittauspiirissä (esim. “PICO Fuses - Littelfuse” 2016).



Kuvio 11: Kehitetty mittalaite kytkettynä puhelimeen.



Kuvio 12: Mittauskytkentä asennettuna Samsung Galaxy S5 -älypuhelimen akkuun.

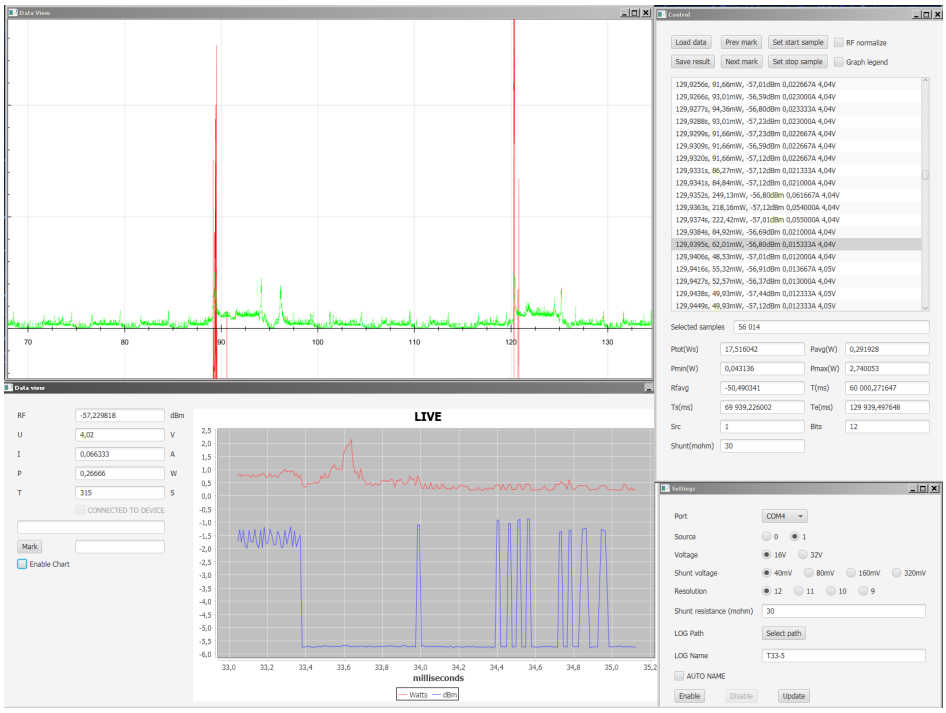
Tehdyissä asennuksissa käytettiin 30 ja 40 milliohmin 1206-pintaliitoskoteloisia virranmittausvastuksia, joiden toleranssit olivat 0,5 ja 1 % (“WSL...18 - Vishay” 2016; “LVK Series - Ohmite” 2016). Mittauskytkennässä virranmittausvastukselta saatavan differentiaalisignaalin jännite on matala (30–40 mV / 1 A), joten sitä kuljettava kaapelointi toteutettiin häiriöiden välttämiseksi kierretyllä parikaapelilla. Akkuun asennetun mittausliittimen ja mittalaitteen välillä käytettiin suojattua parikaapelia. Mittalaite ja muu mittausjärjestely on esitetty

kuvissa 11 ja 12.

Tässä työssä akkua muokkaamalla lisättyä mittauskytkentää yksinkertaisempi vaihtoehto, erityisesti käyttäjän vaihdettavan akun sisältäviin puhelimiin, voi olla valmistaa akun paikalle sopiva liitin ja kytkeä alkuperäinen tai vastaava akku mittauskytkennän kautta tähän liittimeen (esim. Brouwers, Zuniga ja Langendoen 2014). Yksinkertaisimmillaan toteutus voi koostua akun liitentöjen päälle asetettavista, kaksipuoleisista kontakteista, jotka mahdollistavat mittausvastuksen lisäämisen virtapiiriin (Schulman ym. 2011). Tällaisilla väliliittimillä voidaan välttää litium-akun purkaminen ja muokkaaminen, jota ei niiden suuren oikosulkuvirran ja muiden ominaisuuksien takia voida pitää täysin riskittömänä.

D Kehitetyn komponentin energiankulutuksen mittaustulokset

Komponentille suoritetuista testeistä kerättiin aikasarjoina akkujännite ja mobiililaitteen käyttämän virta 1 kHz:n näytenopeudella. Analyysivaiheessa näistä arvoista laskettiin näytekohtainen sähköteho ja lopulta aikasarjalta analysoitavaksi valitun jakson kokonaisenergia. Mittausten tallentamiseen, aikasarjojen visualisoimiseen ja valittujen aikajaksojen energiankulutuksen laskemiseen käytettiin tämän työn yhteydessä kehitettyä Java-ohjelmistoa (kuva 13). Lisäksi verkkoyhteyttä käyttävistä testeistä kerättiin palvelimella pakettitallenteet, joista laskettiin kunkin lähetysjakson aikana palvelimelle saapuneen datan kokonaismäärä.



Kuvio 13: Energiankulutuksen mittaamiseen ja aikasarjojen analysointiin kehitetty ohjelmisto.

Komponenttia testattiin 39:ssä eri konfiguraatiossa, joista kaikista kerättiin viisi rinnakkaista suoritusta, eli mittausjaksoja käsiteltiin yhteensä 195. Mittausjaksojen kuvaus ja rinnakkaisille suorituksille mitatut energiankulutukset on esitetty taulukossa 2. Testitasoinen, rinnakkaisten mittausten energiankulutusten keskiarvo ja keskivirhe esitetty taulukossa 1.

Testi	Kuvaus	Keskikulutus (J/min)	Keskivaihtelu	Keskivirhe
1	GPS 1s Tied. JSON 10s	12,2873	0,8070	0,3609
2	Taustakulutus	1,6363	0,0325	0,0145
3	GPS 1s	11,8771	0,3267	0,1461
4	GPS 2s	11,4863	0,4960	0,2218
5	GPS 5s	9,7931	0,7157	0,3201
6	GPS 10s	8,3306	0,4014	0,1795
7	Acce 500ms rajoitettu	4,4381	0,1699	0,0760
8	Acce 1s rajoitettu	4,2996	0,3950	0,1766
9	Acce 2s rajoitettu	3,6729	0,2633	0,1177
10	Acce 5s rajoitettu	2,3675	0,1930	0,0863
11	Baro 1s rajoitettu	3,9673	0,1332	0,0596
12	Baro 2s rajoitettu	3,5262	0,2764	0,1236
13	Baro 5s rajoitettu	2,4733	0,2105	0,0941
14	Baro 10s rajoitettu	1,9869	0,0496	0,0222
15	Baro 1s rajoittamaton	3,8340	0,0410	0,0183
16	Baro 10s rajoittamaton	3,9273	0,1294	0,0579
17	Acce 100ms	4,1468	0,0982	0,0439
18	Baro 100ms	3,8450	0,0949	0,0424
19	Acce 1s Baro 2s GPS 5s	11,8141	0,5779	0,2584
20	Acce 1s Baro 2s GPS 5s Tied. JSON 5s	11,1736	0,9390	0,4199
21	Acce 1s Baro 2s GPS 5s Tied. JSON 10s	11,2799	0,7192	0,3216
22	Acce 1s Baro 2s GPS 5s Tied. JSON 20s	11,3448	0,4813	0,2152
23	Acce 1s Baro 2s GPS 5s Tied. RAW 10s	10,9938	0,4094	0,1831
24	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 5s JSON HTTP	39,5809	1,1364	0,5082
25	Acce 1s Baro 2s GPS 5s Tied. XML 10s	11,2337	0,4040	0,1807
26	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 10s JSON HTTP	31,0042	2,0924	0,9358
27	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTP	19,1659	0,9690	0,4334
28	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 60s JSON HTTP	15,4547	0,6901	0,3086
29	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 120s JSON HTTP	14,8316	1,3163	0,5887
30	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML HTTP	18,9998	0,9702	0,4339
31	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s RAW HTTP	17,9787	0,5428	0,2428

32	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON Gzip HTTP	17,7737	0,9643	0,4313
33	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML Gzip HTTP	19,1598	1,0846	0,4851
34	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTPS	18,8398	0,9259	0,4141
35	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCP	19,1002	1,5933	0,7125
36	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCPP	20,1821	0,6605	0,2954
37	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON UDP	19,0601	1,0350	0,4629
38	Acce 1s Baro 2s Tied. JSON 10s	4,2185	0,0698	0,0312
39	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,1412	0,0595	0,0266

Taulukko 1: Komponentille suoritettujen energiankulutusmittausten tulosten keskiarvot, keskivaihtelut ja keskivirheet.

Testi	Toisto	Kuvaus	Energia (J/min)
1	1	GPS 1s Tied. JSON 10s	13,7111
1	2	GPS 1s Tied. JSON 10s	12,1509
1	3	GPS 1s Tied. JSON 10s	11,7944
1	4	GPS 1s Tied. JSON 10s	11,9109
1	5	GPS 1s Tied. JSON 10s	11,8692
2	1	Taustakulutus	1,6585
2	2	Taustakulutus	1,6254
2	3	Taustakulutus	1,5946
2	4	Taustakulutus	1,678
2	5	Taustakulutus	1,6248
3	1	GPS 1s	11,6987
3	2	GPS 1s	12,0102
3	3	GPS 1s	11,6097
3	4	GPS 1s	11,675
3	5	GPS 1s	12,392
4	1	GPS 2s	12,298
4	2	GPS 2s	11,0729
4	3	GPS 2s	11,4777
4	4	GPS 2s	11,4896

4	5	GPS 2s	11,0934
5	1	GPS 5s	10,1541
5	2	GPS 5s	9,6619
5	3	GPS 5s	8,6793
5	4	GPS 5s	10,6025
5	5	GPS 5s	9,8679
6	1	GPS 10s	8,382
6	2	GPS 10s	8,4245
6	3	GPS 10s	7,8184
6	4	GPS 10s	8,1248
6	5	GPS 10s	8,9035
7	1	Acce 500ms rajoitettu	4,3618
7	2	Acce 500ms rajoitettu	4,4793
7	3	Acce 500ms rajoitettu	4,4006
7	4	Acce 500ms rajoitettu	4,2464
7	5	Acce 500ms rajoitettu	4,7024
8	1	Acce 1s rajoitettu	4,6342
8	2	Acce 1s rajoitettu	4,7599
8	3	Acce 1s rajoitettu	4,2791
8	4	Acce 1s rajoitettu	3,9149
8	5	Acce 1s rajoitettu	3,91
9	1	Acce 2s rajoitettu	3,41815
9	2	Acce 2s rajoitettu	3,8895
9	3	Acce 2s rajoitettu	3,9192
9	4	Acce 2s rajoitettu	3,7726
9	5	Acce 2s rajoitettu	3,365
10	1	Acce 5s rajoitettu	2,3878
10	2	Acce 5s rajoitettu	2,423
10	3	Acce 5s rajoitettu	2,635
10	4	Acce 5s rajoitettu	2,109
10	5	Acce 5s rajoitettu	2,2829
11	1	Baro 1s rajoitettu	4,1378
11	2	Baro 1s rajoitettu	4,0564
11	3	Baro 1s rajoitettu	3,8962
11	4	Baro 1s rajoitettu	3,7985
11	5	Baro 1s rajoitettu	3,9476
12	1	Baro 2s rajoitettu	3,7417
12	2	Baro 2s rajoitettu	3,5599
12	3	Baro 2s rajoitettu	3,2264

12	4	Baro 2s rajoitettu	3,8404
12	5	Baro 2s rajoitettu	3,2626
13	1	Baro 5s rajoitettu	2,3615
13	2	Baro 5s rajoitettu	2,6345
13	3	Baro 5s rajoitettu	2,7599
13	4	Baro 5s rajoitettu	2,2949
13	5	Baro 5s rajoitettu	2,3158
14	1	Baro 10s rajoitettu	1,9789
14	2	Baro 10s rajoitettu	1,9855
14	3	Baro 10s rajoitettu	1,9232
14	4	Baro 10s rajoitettu	2,0625
14	5	Baro 10s rajoitettu	1,9844
15	1	Baro 1s rajoittamaton	3,847
15	2	Baro 1s rajoittamaton	3,8528
15	3	Baro 1s rajoittamaton	3,853
15	4	Baro 1s rajoittamaton	3,7609
15	5	Baro 1s rajoittamaton	3,8564
16	1	Baro 10s rajoittamaton	4,0372
16	2	Baro 10s rajoittamaton	3,8208
16	3	Baro 10s rajoittamaton	3,8797
16	4	Baro 10s rajoittamaton	4,0914
16	5	Baro 10s rajoittamaton	3,8073
17	1	Acce 100ms	4,3043
17	2	Acce 100ms	4,108
17	3	Acce 100ms	4,0778
17	4	Acce 100ms	4,1779
17	5	Acce 100ms	4,0658
18	1	Baro 100ms	3,771
18	2	Baro 100ms	3,8325
18	3	Baro 100ms	4,0095
18	4	Baro 100ms	3,7942
18	5	Baro 100ms	3,818
19	1	Acce 1s Baro 2s GPS 5s	12,325
19	2	Acce 1s Baro 2s GPS 5s	11,3489
19	3	Acce 1s Baro 2s GPS 5s	11,1312
19	4	Acce 1s Baro 2s GPS 5s	11,8255
19	5	Acce 1s Baro 2s GPS 5s	12,44
20	1	Acce 1s Baro 2s GPS 5s Tied. JSON 5s	10,34522
20	2	Acce 1s Baro 2s GPS 5s Tied. JSON 5s	12,5756

20	3	Acce 1s Baro 2s GPS 5s Tied. JSON 5s	10,5826
20	4	Acce 1s Baro 2s GPS 5s Tied. JSON 5s	11,695
20	5	Acce 1s Baro 2s GPS 5s Tied. JSON 5s	10,6695
21	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s	11,1107
21	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s	10,2953
21	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s	12,3102
21	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s	11,3106
21	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s	11,3725
22	1	Acce 1s Baro 2s GPS 5s Tied. JSON 20s	10,9556
22	2	Acce 1s Baro 2s GPS 5s Tied. JSON 20s	11,936
22	3	Acce 1s Baro 2s GPS 5s Tied. JSON 20s	10,9636
22	4	Acce 1s Baro 2s GPS 5s Tied. JSON 20s	11,7978
22	5	Acce 1s Baro 2s GPS 5s Tied. JSON 20s	11,0709
23	1	Acce 1s Baro 2s GPS 5s Tied. RAW 10s	11,1923
23	2	Acce 1s Baro 2s GPS 5s Tied. RAW 10s	10,5014
23	3	Acce 1s Baro 2s GPS 5s Tied. RAW 10s	11,2347
23	4	Acce 1s Baro 2s GPS 5s Tied. RAW 10s	11,4258
23	5	Acce 1s Baro 2s GPS 5s Tied. RAW 10s	10,6146
24	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 5s JSON HTTP	40,796
24	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 5s JSON HTTP	38,8815
24	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 5s JSON HTTP	39,8949
24	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 5s JSON HTTP	40,3403
24	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 5s JSON HTTP	37,992
25	1	Acce 1s Baro 2s GPS 5s Tied. XML 10s	11,7323
25	2	Acce 1s Baro 2s GPS 5s Tied. XML 10s	10,9658
25	3	Acce 1s Baro 2s GPS 5s Tied. XML 10s	11,1283
25	4	Acce 1s Baro 2s GPS 5s Tied. XML 10s	11,567
25	5	Acce 1s Baro 2s GPS 5s Tied. XML 10s	10,7751
26	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 10s JSON HTTP	34,3892
26	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 10s JSON HTTP	29,2738
26	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 10s JSON HTTP	31,6486
26	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 10s JSON HTTP	29,8836
26	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 10s JSON HTTP	29,8256
27	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTP	19,0642
27	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTP	19,3673
27	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTP	17,9352
27	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTP	18,8506
27	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTP	20,612
28	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 60s JSON HTTP	14,8716

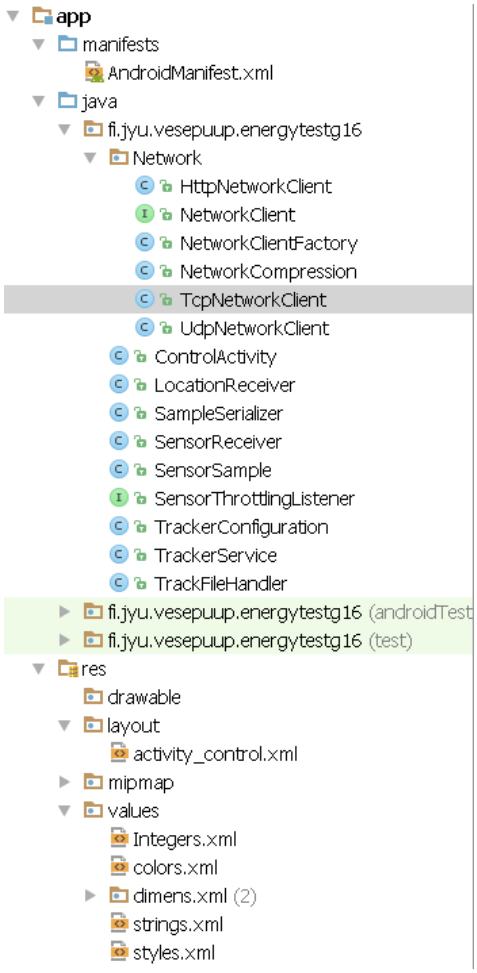
28	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 60s JSON HTTP	14,8814
28	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 60s JSON HTTP	16,5291
28	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 60s JSON HTTP	15,2973
28	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 60s JSON HTTP	15,6939
29	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 120s JSON HTTP	14,6152
29	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 120s JSON HTTP	13,6725
29	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 120s JSON HTTP	14,5521
29	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 120s JSON HTTP	14,2286
29	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 120s JSON HTTP	17,0895
30	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML HTTP	19,503
30	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML HTTP	18,2503
30	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML HTTP	20,2382
30	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML HTTP	19,1836
30	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML HTTP	17,8238
31	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s RAW HTTP	18,9246
31	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s RAW HTTP	17,6858
31	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s RAW HTTP	17,588
31	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s RAW HTTP	17,7731
31	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s RAW HTTP	17,9222
32	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON Gzip HTTP	18,8329
32	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON Gzip HTTP	18,4992
32	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON Gzip HTTP	16,4062
32	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON Gzip HTTP	17,3258
32	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON Gzip HTTP	17,8045
33	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML Gzip HTTP	20,7549
33	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML Gzip HTTP	19,2446
33	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML Gzip HTTP	18,3442
33	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML Gzip HTTP	17,9809
33	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s XML Gzip HTTP	19,4746
34	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTPS	20,0891
34	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTPS	18,367
34	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTPS	18,5904
34	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTPS	19,4261
34	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON HTTPS	17,7266
35	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCP	21,545
35	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCP	17,7493
35	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCP	19,7435
35	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCP	18,6952
35	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCP	17,7682

36	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCPP	20,9422
36	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCPP	19,3794
36	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCPP	20,3501
36	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCPP	20,6128
36	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON TCPP	19,6259
37	1	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON UDP	18,5428
37	2	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON UDP	18,1676
37	3	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON UDP	20,8311
37	4	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON UDP	18,7819
37	5	Acce 1s Baro 2s GPS 5s Tied. JSON 10s Verk. 30s JSON UDP	18,9772
38	1	Acce 1s Baro 2s Tied. JSON 10s	4,220959
38	2	Acce 1s Baro 2s Tied. JSON 10s	4,3223945
38	3	Acce 1s Baro 2s Tied. JSON 10s	4,1544
38	4	Acce 1s Baro 2s Tied. JSON 10s	4,154495
38	5	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,240344
39	1	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,13855
39	2	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,117084
39	3	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,072734
39	4	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,1422695
39	5	Acce 1s Baro 2s Tied. JSON 10s, XML generointi jonoon lähettämättä	4,235424

Taulukko 2: Komponentille suoritettujen testien toistojen energiankulutusmittausten tulokset.

E Kehitetyn komponentin lähdekoodi

Kehitetyn sovelluksen ja siihen sisältyvän komponentin lähdekoodit on listattu tässä luvussa. Osa projektin tiedostoista on tuotettu automaattisesti eikä niihin ole tehty muutoksia, joten ne on jätetty listauksista pois. Tiedostojen sijoittuminen projektin kansioihin on esitetty kuvassa 14. Lähdekooditiedostoista **ControlActivity** sisältää logiikan sovelluksen käyttöliittymälle ja vastaa laitteistosensoreiden olemassaolon tarkistamisesta sekä komponentin käyttämien oikeuksien hallinnasta.



Kuvio 14: Testatun sovelluksen lähdekooditiedostojen sijainti projektin kansioissa.

Palvelukomponentin lähdekoodi on tiedostossa **TrackerService** ja loput lähdekooditiedostot sisältävät sen käyttämien toimintojen toteutuksia. Näistä GPS-paikkatietojen ja sensoritapahtumien vastaanottaminen on toteutettu tiedostoissa **LocationReceiver** ja **SensorRecei-**

ver. Eri protokollia käyttävät verkkoasiakkaat ovat tiedostoissa **HttpNetworkClient**, **TcpNetworkClient** ja **UdpNetworkClient**. Lisäksi xml-tiedostoissa on määritelty sovelluksen käyttämiä vakioita.

ControlActivity.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.databinding.DataBindingUtil;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

import fi.jyu.vesepuup.energytestg16.databinding.ActivityControlBinding;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Code for the TrackerConfiguration activity. Contains the application logic to check the
 * availability of all relevant hardware sensors, handles permission requests to use the required
 * features and controls the configuration and execution of background service.
 */
public class ControlActivity extends AppCompatActivity {
    public static final String tag = "ControlActivity";
    public static final String REFERENCE_KEY = "Configurationstr";
    public static final int PERMISSION_REQUEST_ID_CODE = 448;
    protected static final String[] sAllPermissions =
            {"android.permission.ACCESS_FINE_LOCATION", "android.permission.ACCESS_COARSE_LOCATION",
                    "android.permission.READ_EXTERNAL_STORAGE", "android.permission.WRITE_EXTERNAL_STORAGE"};
    private static TrackerService sService;
    private TrackerConfiguration mConfig;
    private boolean mAllSensorHwExists = false;
    private boolean mAllPermissionsOk = false;
    private boolean mStarted = false;
    private AppCompatActivity mContext;
    private Button mStartButton = null;
    private SharedPreferences mSharedPref;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_control);
```

```
    ActivityControlBinding binding = DataBindingUtil.setContentView(this, R.layout.activity_control);
    mConfig = new TrackerConfiguration();
    mSharedPref = this.getPreferences(Context.MODE_PRIVATE);
    loadSettings();
    binding.setConfig(mConfig);
    mContext = this;
    mStartButton = (Button) findViewById(R.id.button_start);
}

public void starButtonHandler(View view) {
    if (!mStarted) {
        saveSettings();
        handleStartRequest();
    } else
        stopService();
}

protected void loadSettings() {
    String prefString = mSharedPref.getString(REFERENCE_KEY, "");
    if (prefString.length() > 1) {
        mConfig.fromString(prefString);
    }
}

protected void saveSettings() {
    SharedPreferences preferences = this.getPreferences(Context.MODE_PRIVATE);
    SharedPreferences.Editor prefEditor = preferences.edit();
    prefEditor.putString(REFERENCE_KEY, mConfig.toString());
    prefEditor.commit();
}

protected void handleStartRequest() {
    ((TextView) findViewById(R.id.textViewConfigLine)).setText(mConfig.toString());
    checkSensorsExist();
    if (!mAllSensorHwExists) {
        showOkCancelAlertDialog(getResources().getString(R.string.dialog_text_sensors),
                null, null);
        return;
    }
    checkRequestAllPermissions();
    if (mAllPermissionsOk) {
        startService();
    }
}

protected void startService() {
    Log.d(tag, Integer.toString(android.os.Process.myTid()));
    mStarted = true;
    mStartButton.setText(getResources().getString(R.string.button_text_stop));
    Intent i = new Intent(mContext, TrackerService.class);
    i.putExtra("trackerconfiguration", mConfig);
    startService(i);
}

protected void stopService() {
    mStartButton.setText(getResources().getString(R.string.button_text_start));
    stopService(new Intent(mContext, TrackerService.class));
    mStarted = false;
    mAllSensorHwExists = false;
    mAllPermissionsOk = false;
}

protected void checkRequestAllPermissions() {
    final List<String> descriptionList = new ArrayList<String>();
```

```
    final List<String> permissionList = new ArrayList<String>();
    String[] permissionNames = getResources().getStringArray(R.array.permission_names);
    for (int i = 0; i < sAllPermissions.length; i++)
        if (needToRequestPermission(sAllPermissions[i], permissionList))
            descriptionList.add(permissionNames[i]);
    if (permissionList.size() > 0) {
        if (descriptionList.size() > 0) {
            StringBuilder builder = new StringBuilder(getResources().getString(R.string.dialog_text_permissions));
            for (String description : descriptionList)
                builder.append(description + ", ");
            showOkCancelAlertDialog(builder.substring(0, builder.length() - 2),
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            ActivityCompat.requestPermissions(mContext,
                                    permissionList.toArray(new String[permissionList.size()]),
                                    PERMISSION_REQUEST_ID_CODE);
                        }
                    }, null);
        } else {
            ActivityCompat.requestPermissions(mContext,
                    permissionList.toArray(new String[permissionList.size()]),
                    PERMISSION_REQUEST_ID_CODE);
        }
    } else
        mAllPermissionsOk = true;
}

protected void checkSensorsExist() {
    SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    if (sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) == null
            || sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) == null)
        return;
    mAllSensorHwExists = true;
}

private boolean needToRequestPermission(String permission, List<String> requestList) {
    if (ContextCompat.checkSelfPermission(mContext, permission) != PackageManager.PERMISSION_GRANTED) {
        requestList.add(permission);
        if (ActivityCompat.shouldShowRequestPermissionRationale(mContext, permission))
            return true;
    }
    return false;
}

protected void showOkCancelAlertDialog(String message, DialogInterface.OnClickListener okListener,
                                       DialogInterface.OnClickListener CancelListener) {
    AlertDialog.Builder builder = new AlertDialog.Builder(mContext);
    builder.setMessage(message);
    builder.setPositiveButton(getResources().getString(R.string.button_text_ok), okListener);
    if (CancelListener != null)
        builder.setNegativeButton(getResources().getString(R.string.button_text_cancel), CancelListener);
    AlertDialog dialog = builder.create();
    dialog.show();
}

@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
    if (requestCode == PERMISSION_REQUEST_ID_CODE) {
        for (int i = 0; i < permissions.length; i++) {
            if (grantResults[i] != PackageManager.PERMISSION_GRANTED) {
                showOkCancelAlertDialog(getResources().getString
                        (R.string.dialog_text_missing_permissions), null, null);
                return;
```

```
                }
            }
            mAllPermissionsOk = true;
            startService();
        } else
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

TrackerService.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import android.app.Notification;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.location.LocationManager;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.support.annotation.Nullable;
import android.support.v4.app.NotificationCompat;
import android.util.Log;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

import fi.jyu.vesepuup.energytestg16.Network.NetworkClient;
import fi.jyu.vesepuup.energytestg16.Network.NetworkClientFactory;


/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Implementation of the background service. Collects accelerometer, barometer and GPS data,
 * serializes it to JSON, XML or symbol separated format, writes the data to flash and sends it
 * over the network.
 */
public class TrackerService extends Service implements SensorThrottlingListener {
    public static final String tag = "TrackerService";
    public static final int PROCESS_NOTIFICATION_ID = 497;
    public static final String PROCESS_WAKELOCK_TAG = "trackerwl";
    private Service mContext;
    private TrackerConfiguration mConfig;
    private WakeLock mWakeLock = null;
    private Handler mHandler = null;

    private BlockingQueue<SensorSample> mSensorInputQueue;
    private BlockingQueue<String> mNetworkQueue;
    private BlockingQueue<String> mFileQueue;
```

```
private BlockingQueue<ListenerResetter> mPendingSensorRestarts;
private SensorReceiver mPressureReceiver;
private SensorReceiver mAccelerationReceiver;
private LocationReceiver mLocationReceiver;
private boolean mLocationActive;

private int mQueueLenth = 4096;
private String mTrackFileName;

private SensorManager mSensorManager;
private LocationManager mLocationManager;
private NetworkWriterRunnable mNetworkWriterRunnable = null;
private FileWriterRunnable mFileWriterRunnable = null;
private SampleSerializer mSampleSerializer;
private TrackFileHandler mTrackFileHandler;
private HandlerThread mParserThread;
private Handler mParserHandler;
private HandlerThread mNetworkThread;
private Handler mNetworkHandler;
private SampleParser mParserRunnable;
private NetworkClient mNetworkClient;


@Nullable
@Override
public IBinder onBind(Intent intent) {
    return null;
}

@Override
public void onCreate() {
    mContext = this;
    mHandler = new Handler();
    getWakeLock();
}


@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    mConfig = (TrackerConfiguration) intent.getParcelableExtra("trackerconfiguration");
    String fileTimeStamp = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss").format(new Date());
    mTrackFileName = "trk-" + fileTimeStamp + "---" + mConfig.toString() + ".txt";
    generateRunningNotification();
    initializeQueues();
    openTrackFile();
    getSensorLocationManager();
    initializeSensorListeners();
    initializeLocationListeners();
    initializeFileNetworkActivities();
    startFileNetworkActivities();
    startSensorListener(mAccelerationReceiver);
    startSensorListener(mPressureReceiver);
    startLocationListener(mLocationReceiver);
    initializeNetwork();
    initializeNetworkThread();
    initializeParserThread();
    return START_REDELIVER_INTENT;
}

@Override
public void onDestroy() {
    if (mWakeLock != null)
        mWakeLock.release();
    mHandler.removeCallbacks(mFileWriterRunnable);
```

```
        closeTrackFile();
        removePendingRestartRequests();
        stopSensorListener(mAccelerationReceiver);
        stopSensorListener(mPressureReceiver);
        stopLocationListener(mLocationReceiver);
        closeParserThread();
        closeNetworkThread();
        closeNetwork();
        stopForeground(true);
    }

    @Override
    public void throttlingRequest(LocationReceiver receiver, int waitPeriod) {
        stopLocationListener(receiver);
        ListenerResetter reset = new ListenerResetter(receiver);
        mPendingSensorRestarts.add(reset);
        mHandler.postDelayed(reset, waitPeriod);
    }

    // There is often a chance to throttle the excessive sensor data rate by
    // periodically removing the sensor listener subscriptions. In the tested
    // device (SM-G990F) both barometer and accelerometer are reported at their
    // slowest every 180ms
    @Override
    public void throttlingRequest(SensorReceiver receiver, int waitPeriod) {
        stopSensorListener(receiver);
        ListenerResetter reset = new ListenerResetter(receiver);
        mPendingSensorRestarts.add(reset);
        mHandler.postDelayed(reset, waitPeriod);
    }

    private void closeNetwork() {
        if (mConfig.getNetworkInterval() < 1)
            return;
        mNetworkClient.close();
    }

    private void initializeNetwork() {
        if (mConfig.getNetworkInterval() < 1)
            return;
        mNetworkClient = NetworkClientFactory.getClientWithConfiguration(
                mConfig.getmNetworkProtocolEnum(),
                mContext.getResources().getInteger(R.integer.server_port_number),
                mContext.getResources().getString(R.string.server_ip_address),
                mConfig.getmNetworkCompression());
    }

    private void initializeNetworkThread() {
        if (mConfig.getNetworkInterval() < 1) {
            mNetworkThread = null;
            return;
        }
        mNetworkThread = new HandlerThread("NetworkThread", Thread.MAX_PRIORITY);
        mNetworkThread.start();
        mNetworkHandler = new Handler(mNetworkThread.getLooper());
        mNetworkWriterRunnable = new NetworkWriterRunnable(mConfig.getNetworkInterval(), mNetworkHandler);
        mNetworkHandler.post(mNetworkWriterRunnable);
    }

    private void closeNetworkThread() {
        if (mNetworkThread == null)
            return;
        mNetworkHandler.removeCallbacks(mNetworkWriterRunnable);
        mNetworkThread.interrupt();
```

```
        mNetworkThread.quit();
    }

    private void initializeParserThread() {
        if (mConfig.getFileInterval() < 1 && mConfig.getNetworkInterval() < 1) {
            mParserThread = null;
            return;
        }
        mParserThread = new HandlerThread("ParserThread", Thread.MAX_PRIORITY);
        mParserThread.start();
        mParserHandler = new Handler(mParserThread.getLooper());
        mParserRunnable = new SampleParser(mConfig.getmFileFormatEnum(),
                mConfig.getmNetworkFormatEnum(), mConfig.fileNetworkMatchingFormats());
        mParserHandler.post(mParserRunnable);
    }

    private void closeParserThread() {
        if (mParserThread == null)
            return;
        mParserHandler.removeCallbacks(mParserRunnable);
        mParserThread.interrupt();
        mParserThread.quit();
    }

    private void closeTrackFile() {
        if (mConfig.getFileInterval() > 1 && mTrackFileHandler != null) {
            mTrackFileHandler.closeFile();
        }
    }

    private void openTrackFile() {
        if (mConfig.getFileInterval() > 1) {
            mTrackFileHandler = new TrackFileHandler();
            mTrackFileHandler.openFile(mTrackFileName, mContext);
        }
    }

    private void stopLocationListener(LocationReceiver receiver) {
        if (!receiver.isActive())
            return;
        try {
            mLocationManager.removeUpdates(receiver);
        } catch (SecurityException e) {
            return;
        }
        receiver.setActive(false);
    }

    private boolean startLocationListener(LocationReceiver receiver) {
        if (mConfig.getGpsRate() < 1)
            return false;
        try {
            mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                    mConfig.getGpsRate(), 0, receiver);
        } catch (SecurityException se) {
            return false;
        } catch (IllegalArgumentException ae) {
            return false;
        }
        receiver.setActive(true);
        return true;
    }

    private void removePendingRestartRequests() {
```

```
    for (ListenerResetter restart : mPendingSensorRestarts)
        mHandler.removeCallbacks(restart);
    mPendingSensorRestarts.clear();
}

private void initializeFileNetworkActivities() {
    mSampleSerializer = new SampleSerializer();
    mFileWriterRunnable = new FileWriterRunnable(mConfig.getFileInterval(), mTrackFileHandler);
}

private void startFileNetworkActivities() {
    if (mConfig.getFileInterval() > 1)
        mFileWriterRunnable.run();
}

private boolean startSensorListener(SensorReceiver receiver) {
    if (receiver.getIntervalMillis() < 1 || receiver.isActive())
        return false;
    boolean success = mSensorManager.registerListener(receiver, receiver.getSourceSensor(),
            receiver.getIntervalMillis() * 1000);
    if (success)
        receiver.setActive(true);
    return success;
}

private void stopSensorListener(SensorReceiver receiver) {
    if (receiver.isActive())
        mSensorManager.unregisterListener(receiver, receiver.getSourceSensor());
    receiver.setActive(false);
}

private void initializeLocationListeners() {
    mLocationReceiver = new LocationReceiver(mSensorInputQueue,
            mConfig.getGpsRate(), this);
    mLocationReceiver.setActive(false);
}

private void initializeSensorListeners() {
    Sensor pressureSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
    Sensor accelerationSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mPressureReceiver = new SensorReceiver(mSensorInputQueue,
            mConfig.getBaroRate(), this, pressureSensor);
    mAccelerationReceiver = new SensorReceiver(mSensorInputQueue,
            mConfig.getAcceRate(), this, accelerationSensor);
    mPressureReceiver.setActive(false);
    mAccelerationReceiver.setActive(false);
}

private void getSensorLocationManager() {
    mSensorManager = (SensorManager) mContext.getSystemService(Service.SENSOR_SERVICE);
    mLocationManager = (LocationManager) mContext.getSystemService(LOCATION_SERVICE);
}

private void initializeQueues() {
    mSensorInputQueue = new ArrayBlockingQueue<SensorSample>(mQueueLenth);
    mNetworkQueue = new ArrayBlockingQueue<String>(mQueueLenth);
    mFileQueue = new ArrayBlockingQueue<String>(mQueueLenth);
    mPendingSensorRestarts = new ArrayBlockingQueue<ListenerResetter>(5);
}

private void getWakeLock() {
    PowerManager powerManager = (PowerManager) getSystemService(POWER_SERVICE);
    mWakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
            PROCESS_WAKELOCK_TAG);
```

```
        mWakeLock.acquire();
    }

    private void generateRunningNotification() {
        Intent notificationIntent = new Intent(this, TrackerService.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
        NotificationCompat.Builder notificationBuilder =
                new NotificationCompat.Builder(this)
                        .setContentTitle(getString(R.string.process_name))
                        .setContentText(getString(R.string.process_description))
                        .setSmallIcon(R.mipmap.ic_launcher)
                        .setContentIntent(pendingIntent);
        Notification notification = notificationBuilder.build();
        startForeground(PROCESS_NOTIFICATION_ID, notification);
    }


    private class ListenerResetter implements Runnable {
        private SensorReceiver mSensorReceiver = null;
        private LocationReceiver mLocationReceiver = null;

        public ListenerResetter(SensorReceiver s) {
            mSensorReceiver = s;
        }

        public ListenerResetter(LocationReceiver l) {
            mLocationReceiver = l;
        }

        public void run() {
            if (mSensorReceiver != null)
                startSensorListener(mSensorReceiver);
            else if (mLocationReceiver != null)
                startLocationListener(mLocationReceiver);
            mPendingSensorRestarts.remove(this);
        }
    }


    private class NetworkWriterRunnable implements Runnable {
        private int mDelay;
        private Handler mHandler;
        private ArrayList<String> mSampleData;
        private boolean mInitRun = false;

        public NetworkWriterRunnable(int delay, Handler handler) {
            mDelay = delay;
            mHandler = handler;
            mSampleData = new ArrayList<String>(256);
        }

        public void run() {
            if (!mInitRun) {
                mNetworkClient.open();
                mInitRun = true;
            }
            mSampleData.clear();
            mNetworkQueue.drainTo(mSampleData);
            mNetworkClient.sendSamples(mSampleData);
            mNetworkHandler.postDelayed(mNetworkWriterRunnable, mDelay);
        }
    }

    private class FileWriterRunnable implements Runnable {
```

```
        private int mDelay;
        private TrackFileHandler mFileHandler;
        private StringBuilder builder;
        private String current;

        public FileWriterRunnable(int delay, TrackFileHandler file) {
            mDelay = delay;
            mFileHandler = file;
        }

        public void run() {
            builder = new StringBuilder();
            while ((current = mFileQueue.poll()) != null)
                builder.append(current);
            if (builder.length() > 0)
                mFileHandler.writeFile(builder.toString());
            mHandler.postDelayed(mFileWriterRunnable, mDelay);
        }
    }

    private class SampleParser implements Runnable {
        private TrackerConfiguration.DataFormat[] mFormats;
        private boolean mMatchingFormats;

        public SampleParser(TrackerConfiguration.DataFormat fileFormat,
                            TrackerConfiguration.DataFormat netFormat, boolean matchingFormats) {
            mFormats = new TrackerConfiguration.DataFormat[]{fileFormat, netFormat};
            mMatchingFormats = matchingFormats;
        }

        @Override
        public void run() {
            while (true) {
                try {
                    SensorSample sample = mSensorInputQueue.take();
                    String[] parsedStr = new String[2];
                    for (int i = 0; i < 2; i++) {
                        switch (mFormats[i]) {
                            default:
                            case Raw:
                                parsedStr[i] = mSampleSerializer.toSeparatorString(sample);
                                break;
                            case JSON:
                                parsedStr[i] = mSampleSerializer.getJsonRecord(sample);
                                break;
                            case XML:
                                parsedStr[i] = mSampleSerializer.getXmlRecord(sample);
                                break;
                        }
                        if (mMatchingFormats) {
                            parsedStr[1] = parsedStr[0];
                            break;
                        }
                    }
                    mFileQueue.offer(parsedStr[0]);
                    mNetworkQueue.offer(parsedStr[1]);
                } catch (InterruptedException e) {
                    return;
                }
            }
        }
    }

}
```

SensorThrottlingListener.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Interface enabling to receive sensor data throttling requests in
 * the case of excessive sensor data update speed.
 */
public interface SensorThrottlingListener {

    public void throttlingRequest(LocationReceiver receiver, int waitPeriod);

    public void throttlingRequest(SensorReceiver receiver, int waitPeriod);
}
```

LocationReceiver.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import android.location.Location;
import android.location.LocationListener;
import android.os.Bundle;

import java.util.concurrent.BlockingQueue;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Class implementing LocationListener. Receives location updates and pushes
 * them to the mSampleQueue as SensorSample objects.
 */
public class LocationReceiver implements LocationListener {
    private int mIntervalMillis;
    private long mLastTimeStamp;
    private BlockingQueue<SensorSample> mSampleQueue;
    private boolean mActive;
    private boolean mDiscardExcess = false;
    private SensorThrottlingListener mThrottlingController;

    public LocationReceiver(BlockingQueue<SensorSample> queue, int intervalMillis,
                            SensorThrottlingListener throttler) {
        mSampleQueue = queue;
        mIntervalMillis = intervalMillis;
        mThrottlingController = throttler;
    }

    @Override
    public void onLocationChanged(Location location) {
        long time = location.getTime();
        long sinceLast = time - mLastTimeStamp;
        if (sinceLast < (mIntervalMillis - 50)) {
            if (!isDiscardExcess())
                mThrottlingController.throttlingRequest(this, (int) (((mIntervalMillis - 50) - sinceLast)));
        } else {
```

```
            SensorSample sample = new SensorSample(SensorSample.SampleType.GPS,
                    time, locationDataArray(location));
            mSampleQueue.offer(sample);
            mLastTimeStamp = time;
        }
    }

    @Override
    public void onStatusChanged(String s, int i, Bundle bundle) {
    }

    @Override
    public void onProviderEnabled(String s) {
    }

    @Override
    public void onProviderDisabled(String s) {
    }

    private double[] locationDataArray(Location location) {
        double[] locationData = new double[5];
        locationData[0] = location.getLatitude();
        locationData[1] = location.getLongitude();
        locationData[2] = location.getSpeed();
        locationData[3] = location.getAltitude();
        locationData[4] = location.getAccuracy();
        return locationData;
    }

    public boolean isActive() {
        return mActive;
    }

    public void setActive(boolean mActive) {
        this.mActive = mActive;
    }

    public boolean isDiscardExcess() {
        return mDiscardExcess;
    }

    public void setDiscardExcess(boolean discardExcess) {
        this.mDiscardExcess = discardExcess;
    }
}
```

SensorReceiver.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;

import java.util.concurrent.BlockingQueue;


/**
 * Veli-Mikko Puupponen 2016
 */
```

```
/**
 * Class implementing SensorEventListener. Receives sensor samples and pushes
 * them to the mSampleQueue as SensorSample objects.
 */
public class SensorReceiver implements SensorEventListener {
    private BlockingQueue<SensorSample> mSampleQueue;
    private int mIntervalMillis;
    private long mLastTimeStamp;
    private Sensor mSourceSensor;
    private boolean mActive;
    private boolean mDiscardExcess = false;
    private SensorThrottlingListener mThrottlingController;

    public SensorReceiver(BlockingQueue<SensorSample> queue, int intervalMillis,
                          SensorThrottlingListener throttler, Sensor sensor) {
        this(queue, intervalMillis, throttler);
        mSourceSensor = sensor;
    }

    public SensorReceiver(BlockingQueue<SensorSample> queue, int intervalMillis,
                          SensorThrottlingListener throttler) {
        mSampleQueue = queue;
        mIntervalMillis = intervalMillis;
        mThrottlingController = throttler;
    }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        long time = sensorEvent.timestamp;
        long sinceLast = time - mLastTimeStamp;
        if (sinceLast < (900000 * mIntervalMillis)) {
            if (!isDiscardExcess()) {
                if (sinceLast < 0)
                    sinceLast = 1;
                mThrottlingController.throttlingRequest(this, (int) (mIntervalMillis - (sinceLast / 950000)));
            }
        } else {
            SensorSample sample = new SensorSample
                    (SensorSample.SampleType.fromAndroidType(sensorEvent.sensor.getType()),
                            System.currentTimeMillis(), doublesFromFloats(sensorEvent.values));
            mSampleQueue.offer(sample);
            mLastTimeStamp = time;
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {

    }

    public Sensor getSourceSensor() {
        return mSourceSensor;
    }

    public void setSourceSensor(Sensor mSourceSensor) {
        this.mSourceSensor = mSourceSensor;
    }

    public int getIntervalMillis() {
        return mIntervalMillis;
    }

    public void setIntervalMillis(int mIntervalMillis) {
        this.mIntervalMillis = mIntervalMillis;
```

```
    }

    private double[] doublesFromFloats(float[] source) {
        int length = source.length;
        double[] doubles = new double[length];
        for (int i = 0; i < length; i++)
            doubles[i] = source[i];
        return doubles;
    }

    public boolean isActive() {
        return mActive;
    }

    public void setActive(boolean mActive) {
        this.mActive = mActive;
    }

    public boolean isDiscardExcess() {
        return mDiscardExcess;
    }

    public void setDiscardExcess(boolean discardExcess) {
        this.mDiscardExcess = discardExcess;
    }

}
```

SensorSample.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import java.util.HashMap;
import java.util.Map;

import static android.hardware.Sensor.TYPE_ACCELEROMETER;
import static android.hardware.Sensor.TYPE_PRESSURE;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Class for sensor and location samples.
 */
public class SensorSample {
    public static final String TAG_RECORD = "Sample";
    public static final String TAG_TYPE = "SampleType";
    public static final String TAG_TIMESTAMP = "TimeStamp";
    public static final String TAG_CONTENT = "Content";

    public enum SampleType {
        GPS(1000), ACCELEROMETER(TYPE_ACCELEROMETER), BAROMETER(TYPE_PRESSURE);
        private int identifier;
        private static Map<Integer, SampleType> idMap = new HashMap<Integer, SampleType>();

        private SampleType(final int id) {
            identifier = id;
        }

        static {
```

```
            for (SampleType st : SampleType.values())
                idMap.put(st.identifier, st);
        }

        public static SampleType fromAndroidType(int at) {
            return idMap.get(at);
        }
    }

    // left public to minimize the getter/setter usage
    public long timeStamp;
    public double[] content;
    public SampleType contentType;
    public String currentSerialization = null;

    public SensorSample(SampleType contentType, long time, double[] value) {
        this.contentType = contentType;
        timeStamp = time;
        content = value;
    }

    public long getTimeStamp() {
        return timeStamp;
    }

    public void setTimeStamp(long timeStamp) {
        this.timeStamp = timeStamp;
    }

    public double[] getContent() {
        return content;
    }

    public void setContent(double[] content) {
        this.content = content;
    }

    public SampleType getContentType() {
        return contentType;
    }

    public void setContentType(SampleType contentType) {
        this.contentType = contentType;
    }
}
```

NetworkClient.java

tabsize

```
package fi.jyu.vesepuup.energytestg16.Network;

import java.util.List;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Interface for all network client implementations
 */
public interface NetworkClient {
    public boolean open();
```

```
    public void close();

    public boolean sendSamples(List<String> samples);

    public void setHost(int port, String hostAddress);

    public void setCompression(boolean c);

}
```

NetworkClientFactory.java

tabsize

```
package fi.jyu.vesepuup.energytestg16.Network;


import fi.jyu.vesepuup.energytestg16.TrackerConfiguration;

/**
 * Veli-Mikko Puupponen 2016
 */

public class NetworkClientFactory {

    public static NetworkClient getClientWithConfiguration(TrackerConfiguration.NetworkProtocol protocol,
                                                            int port, String host, boolean compression) {
        NetworkClient client;
        switch (protocol) {
            case TCPP:
            case TCP:
                TcpNetworkClient tcl = new TcpNetworkClient();
                if (protocol == TrackerConfiguration.NetworkProtocol.TCPP)
                    tcl.setPersistentConnection(true);
                client = tcl;
                break;
            case UDP:
                client = new UdpNetworkClient();
                break;
            case HTTP:
            case HTTPS:
                HttpNetworkClient htc = new HttpNetworkClient();
                if (protocol == TrackerConfiguration.NetworkProtocol.HTTPS)
                    htc.setHTTPS(true);
                client = htc;
                break;
            default:
                client = null;
                break;
        }
        if (client != null) {
            client.setCompression(compression);
            client.setHost(port, host);
        }
        return client;
    }
}
```

HttpNetworkClient.java

tabsize

```java
package fi.jyu.vesepuup.energytestg16.Network;

import android.util.Log;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.X509Certificate;
import java.util.List;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * HTTP(S) client for making POST requests to a remote server.
 */
public class HttpNetworkClient implements NetworkClient {
    public static final int CONNECTION_TIMEOUT = 4000;
    private String tag = "HttpNetworkConnection";
    private int mHostPort;
    private String mHostAddress;
    private boolean mCompression = false;
    private boolean mUsesHTTPS = false;
    private URL mConnectionURL;
    private HttpURLConnection mHttpConnection;
    private OutputStream mOutputStream;
    private InputStream mInputStream;
    private DataOutputStream mDataStream;


    @Override
    public boolean open() {
        try {
            if (!mUsesHTTPS)
                mConnectionURL = new URL("http://" + mHostAddress + ":" + Integer.toString(mHostPort) + "/");
            else
                mConnectionURL = new URL("https://" + mHostAddress + ":" + Integer.toString(mHostPort) + "/");
        } catch (MalformedURLException e) {
            return false;
        }
        return true;
    }

    @Override
    public void close() {
    }

    @Override
    public boolean sendSamples(List<String> samples) {
```

```
    boolean resultOk = false;
    try {
        mHttpConnection = (HttpURLConnection) mConnectionURL.openConnection();
    } catch (IOException e) {
        return false;
    }
    try {
        mHttpConnection.setConnectTimeout(CONNECTION_TIMEOUT);
        mHttpConnection.setReadTimeout(CONNECTION_TIMEOUT);
        mHttpConnection.setRequestMethod("POST");
        mHttpConnection.setDoOutput(true);
        mOutputStream = mHttpConnection.getOutputStream();
        if (!mCompression) {
            mDataStream = new DataOutputStream(mOutputStream);
            for (String s : samples)
                mDataStream.writeBytes(s);
        } else {
            NetworkCompression.getGZipCompressed(samples, mOutputStream);
        }
        mOutputStream.close();
        if (mHttpConnection.getResponseCode() == HttpsURLConnection.HTTP_OK)
            resultOk = true;
        mInputStream = mHttpConnection.getInputStream();
        InputStreamReader r = new InputStreamReader(mInputStream);
        while (r.ready())
            r.read();
        r.close();
    } catch (IOException e) {
        return false;
    }
    return resultOk;
}


/**
 * Setting URLConnection to use a trustmanager that performs no checking and ignores all
 * errors. Never to be used for other than testing purposes!
 */
private void setHTTPSIgnoreCertificateErrors() {
    try {
        SSLContext context = SSLContext.getInstance("TLS");
        context.init(null, new TrustManager[]{
                new X509TrustManager() {
                    public void checkClientTrusted(X509Certificate[] chain, String authType) {
                    }

                    public void checkServerTrusted(X509Certificate[] chain, String authType) {
                    }

                    public X509Certificate[] getAcceptedIssuers() {
                        return new X509Certificate[]{};
                    }
                }
        }, null);
        HttpsURLConnection.setDefaultHostnameVerifier(new HostnameVerifier() {
            public boolean verify(String hostname, SSLSession session) {
                return true;
            }
        });
        HttpsURLConnection.setDefaultSSLSocketFactory(context.getSocketFactory());
    } catch (NoSuchAlgorithmException e) {
    } catch (KeyManagementException e) {
    }
}
```

```
    @Override
    public void setHost(int port, String hostAddress) {
        mHostAddress = hostAddress;
        mHostPort = port;
    }

    @Override
    public void setCompression(boolean c) {
        mCompression = c;
    }

    public boolean usesHTTPS() {
        return mUsesHTTPS;
    }

    public void setHTTPS(boolean mUsesHTTPS) {
        this.mUsesHTTPS = mUsesHTTPS;
        setHTTPSIgnoreCertificateErrors();
    }
}
```

TcpNetworkClient.java

tabsize

```
package fi.jyu.vesepuup.energytestg16.Network;

import android.util.Log;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.util.List;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Network client using TCP transport with no high-level application protocols
 */
public class TcpNetworkClient implements NetworkClient {
    private String tag = "TcpNetworkClient";
    private int mHostPort;
    private String mHostAddress;
    private boolean mCompression = false;
    private boolean mPersistentConnection = false;
    private Socket mTCPClientSocket;
    private OutputStream mTCPOutStream;
    private DataOutputStream mTCPDataStream;
    private boolean mOpen = false;

    @Override
    public boolean open() {
        if (!mPersistentConnection)
            return true;
        mOpen = reopenTcpSocket();
        return mOpen;
    }

    @Override
```

```
public void close() {
    closeTcpSocket();
    mOpen = false;
}

@Override
public boolean sendSamples(List<String> samples) {
    if (!mOpen)
        mOpen = reopenTcpSocket();
    if (!mCompression) {
        try {
            for (String s : samples)
                mTCPDataStream.writeBytes(s);
        } catch (IOException e) {
            closeTcpSocket();
            return false;
        }
    } else
        try {
            NetworkCompression.getGZipCompressed(samples, mTCPOutStream);
        } catch (IOException e) {
            closeTcpSocket();
            return false;
        }
    if (!mPersistentConnection)
        closeTcpSocket();
    return true;
}

private boolean reopenTcpSocket() {
    try {
        mTCPClientSocket = new Socket(mHostAddress, mHostPort);
        mTCPOutStream = mTCPClientSocket.getOutputStream();
        mTCPDataStream = new DataOutputStream(mTCPOutStream);
    } catch (IOException e) {
        mTCPClientSocket = null;
        mTCPOutStream = null;
        mTCPDataStream = null;
        return false;
    }
    return true;
}

private void closeTcpSocket() {
    mOpen = false;
    if (mTCPClientSocket == null)
        return;
    try {
        if (mTCPDataStream != null && mTCPOutStream != null)
            mTCPDataStream.close();
        mTCPClientSocket.shutdownInput();
        mTCPClientSocket.shutdownOutput();
    } catch (IOException e) {
    } finally {
        try {
            mTCPClientSocket.close();
        } catch (IOException e) {
        }
    }
}

@Override
public void setHost(int port, String hostAddress) {
    mHostAddress = hostAddress;
```

```
        mHostPort = port;
    }

    @Override
    public void setCompression(boolean c) {
        mCompression = c;
    }

    public boolean isPersistentConnection() {
        return mPersistentConnection;
    }

    public void setPersistentConnection(boolean mPersistentConnection) {
        this.mPersistentConnection = mPersistentConnection;
    }
}
```

UdpNetworkClient.java

tabsize

```
package fi.jyu.vesepuup.energytestg16.Network;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.List;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Network client using UDP transport with no high-level application protocols.
 * This implementation does not control for packet loss
 */
public class UdpNetworkClient implements NetworkClient {
    private String tag = "UdpNetworkClient";
    private int mHostPort;
    private String mHostAddress;
    private boolean mCompression = false;
    private DatagramSocket mSocket;
    private InetAddress mHostIAddress;
    private DatagramPacket mLastPacket;
    private boolean mInitialized = false;

    @Override
    public boolean open() {
        try {
            mSocket = new DatagramSocket();
        } catch (SocketException e) {
            mInitialized = false;
            return mInitialized;
        }
        try {
            mHostIAddress = InetAddress.getByName(mHostAddress);
        } catch (UnknownHostException e) {
            mInitialized = false;
            close();
            return mInitialized;
```

```
        }
        mInitialized = true;
        return mInitialized;
    }

    @Override
    public void close() {
        if (mInitialized && mSocket != null)
            mSocket.close();
    }

    @Override
    public boolean sendSamples(List<String> samples) {
        StringBuilder builder = new StringBuilder();
        byte[] packetContent;
        if (!mInitialized)
            open();
        for (String s : samples)
            builder.append(s);
        if (!mCompression)
            packetContent = builder.toString().getBytes();
        else
            packetContent = NetworkCompression.getGZipCompressed(builder.toString());
        mLastPacket = new DatagramPacket(packetContent,
                packetContent.length, mHostIAddress, mHostPort);
        try {
            mSocket.send(mLastPacket);
        } catch (IOException e) {
            close();
            return false;
        }
        return true;
    }

    @Override
    public void setHost(int port, String hostAddress) {
        mHostAddress = hostAddress;
        mHostPort = port;
    }

    @Override
    public void setCompression(boolean c) {
        mCompression = c;
    }
}
```

NetworkCompression.java

tabsize

```
package fi.jyu.vesepuup.energytestg16.Network;

import android.support.annotation.Nullable;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.List;
import java.util.zip.GZIPOutputStream;

/**
 * Veli-Mikko Puupponen 2016
 */
```

```java
/**
 * Static utility methods providing GZip compression
 */
public class NetworkCompression {

    @Nullable
    public static byte[] getGZipCompressed(String data) {
        ByteArrayOutputStream bos = new ByteArrayOutputStream(data.length());
        byte[] gzip;
        try {
            GZIPOutputStream gzs = new GZIPOutputStream(bos);
            gzs.write(data.getBytes());
            gzs.close();
            bos.close();
            gzip = bos.toByteArray();
        } catch (IOException e) {
            return null;
        }
        return gzip;
    }

    @Nullable
    public static byte[] getGZipCompressed(List<String> data) {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        byte[] gzip;
        try {
            GZIPOutputStream gzs = new GZIPOutputStream(bos);
            for (String d : data)
                gzs.write(d.getBytes());
            gzs.close();
            bos.close();
            gzip = bos.toByteArray();
        } catch (IOException e) {
            return null;
        }
        return gzip;
    }

    public static void getGZipCompressed(List<String> data, OutputStream os) throws IOException {
        int length = data.size();
        GZIPOutputStream gzs = new GZIPOutputStream(os);
        for (int i = 0; i < length; i++) {
            gzs.write(data.get(i).getBytes());
        }
        gzs.flush();
        gzs.finish();
    }
}
```

SampleSerializer.java

tabsize

```java
package fi.jyu.vesepuup.energytestg16;

import android.util.Xml;

import com.google.gson.FieldNamingPolicy;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import org.xmlpull.v1.XmlSerializer;
```

```
import java.io.IOException;
import java.io.StringWriter;
import java.util.Arrays;


/**
 * Veli-Mikko Puupponen 2016
 */

public class SampleSerializer {
    private Gson mGsonSerializer;
    private XmlSerializer mXMLSerializer;

    public SampleSerializer() {
        mGsonSerializer = new GsonBuilder()
                .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
                .disableHtmlEscaping()
                .setPrettyPrinting()
                .create();
        mXMLSerializer = Xml.newSerializer();
    }

    public String getJsonRecord(SensorSample sample) {
        return mGsonSerializer.toJson(sample);
    }

    public String getXmlRecord(SensorSample sample) {
        StringWriter writer = new StringWriter();
        try {
            mXMLSerializer.setOutput(writer);
            mXMLSerializer.startDocument("UTF-8", true);
            mXMLSerializer.setFeature("http://xmlpull.org/v1/doc/features.html#indent-output", true);
            mXMLSerializer.startTag("", SensorSample.TAG_RECORD);
            mXMLSerializer.startTag("", SensorSample.TAG_TYPE);
            mXMLSerializer.text(sample.contentType.name());
            mXMLSerializer.endTag("", SensorSample.TAG_TYPE);
            mXMLSerializer.startTag("", SensorSample.TAG_TIMESTAMP);
            mXMLSerializer.text(Long.toString(sample.timeStamp));
            mXMLSerializer.endTag("", SensorSample.TAG_TIMESTAMP);
            mXMLSerializer.startTag("", SensorSample.TAG_CONTENT);
            mXMLSerializer.text(Arrays.toString(sample.content));
            mXMLSerializer.endTag("", SensorSample.TAG_CONTENT);
            mXMLSerializer.endTag("", SensorSample.TAG_RECORD);
            mXMLSerializer.endDocument();
            return writer.toString();
        } catch (IOException e) {
            return null;
        }
    }

    public String toSeparatorString(SensorSample sample) {
        StringBuilder builder = new StringBuilder(Long.toString(sample.timeStamp));
        builder.append("|");
        builder.append(sample.contentType.name());
        builder.append("|");
        builder.append(Arrays.toString(sample.content));
        builder.append("||");
        return builder.toString();
    }

}
```

TrackerConfiguration.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import android.databinding.BindingAdapter;
import android.databinding.InverseBindingAdapter;
import android.os.Parcel;
import android.os.Parcelable;
import android.widget.EditText;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Class for storing and serializing TrackerService configuration values.
 * This class is used to configure the TrackerService using the values supplied
 * by the UI in the ControlActivity class.
 */
public class TrackerConfiguration implements Parcelable {
    public enum DataFormat {Raw, JSON, XML}

    ;

    public enum NetworkProtocol {TCP, UDP, HTTP, HTTPS, TCPP}

    ;
    private static final int sDefaultGpsRate = 10000;
    private static final int sDefaultAcceRate = 2000;
    private static final int sDefaultBaroRate = 2000;
    private static final int sDefaultFileInterval = 5000;
    private static final int sDefaultfileFormat = 1;
    private static final int sDefaultnetworkInterval = 30000;
    private static final int sDefaultnetworkFormat = 0;
    private static final int sDefaultnetworkProto = 1;

    // update rates in milliseconds, -1 to disable
    // for GPS, Accelerometer, Barometer
    private int mGpsRate;
    private int mAcceRate;
    private int mBaroRate;
    private int mFileInterval;
    // file format; 0 for raw string, 1 for JSON, 2 for XML
    private int mFileFormat;
    // network transmission interval, -1 to disable
    private int mNetworkInterval;
    // network serialization, 0 for raw string, 1 for JSON, 2 for XML
    // and with enabled GZip compression  3, 4, 5 respectively
    private int mNetworkFormat;
    // network transmission format, 0 for TCP, 1 for UDP, 2 for HTTP, 3 HTTPS, 4 for persisting TCP socket
    private int mNetworkProto;
    private boolean mNetworkCompression;

    private DataFormat mFileFormatEnum;
    private DataFormat mNetworkFormatEnum;
    private NetworkProtocol mNetworkProtocolEnum;

    public TrackerConfiguration() {
        setDefaults();
    }

    private void setDefaults() {
```

```
        mGpsRate = sDefaultGpsRate;
        mAcceRate = sDefaultAcceRate;
        mBaroRate = sDefaultBaroRate;
        mFileInterval = sDefaultFileInterval;
        setFileFormat(sDefaultfileFormat);
        mNetworkInterval = sDefaultnetworkInterval;
        setNetworkFormat(sDefaultfileFormat);
        setNetworkProto(sDefaultnetworkProto);
    }

    protected TrackerConfiguration(Parcel in) {
        setGpsRate(in.readInt());
        setAcceRate(in.readInt());
        setBaroRate(in.readInt());
        setFileInterval(in.readInt());
        setFileFormat(in.readInt());
        setNetworkInterval(in.readInt());
        setNetworkFormat(in.readInt());
        setNetworkProto(in.readInt());
    }

    public boolean isPersistentTcp() {
        if (mNetworkProtocolEnum == NetworkProtocol.TCPP)
            return true;
        return false;
    }

    public boolean fileNetworkMatchingFormats() {
        if (mNetworkFormatEnum == mFileFormatEnum)
            return true;
        return false;
    }

    public static final Creator<TrackerConfiguration> CREATOR = new Creator<TrackerConfiguration>() {
        @Override
        public TrackerConfiguration createFromParcel(Parcel in) {
            return new TrackerConfiguration(in);
        }

        @Override
        public TrackerConfiguration[] newArray(int size) {
            return new TrackerConfiguration[size];
        }
    };

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeInt(getGpsRate());
        parcel.writeInt(getAcceRate());
        parcel.writeInt(getBaroRate());
        parcel.writeInt(getFileInterval());
        parcel.writeInt(getFileFormat());
        parcel.writeInt(getNetworkInterval());
        parcel.writeInt(getNetworkFormat());
        parcel.writeInt(getNetworkProto());
    }

    @BindingAdapter("android:text")
    public static void adapterIntToText(EditText et, int value) {
```

```
    et.setText(Integer.toString(value));
    et.setSelection(et.getText().length());
}

@InverseBindingAdapter(attribute = "android:text")
public static int adapterTextToInt(EditText et) {
    String string = et.getText().toString();
    try {
        return string.isEmpty() ? 0 : Integer.parseInt(string);
    } catch (NumberFormatException e) {
        return 0;
    }
}

public int getFileInterval() {
    return mFileInterval;
}

public void setFileInterval(int fileInterval) {
    this.mFileInterval = fileInterval;
}

public int getFileFormat() {
    return mFileFormat;
}

public void setFileFormat(int fileFormat) {
    this.mFileFormat = fileFormat;
    mFileFormatEnum = DataFormat.values()[fileFormat];
}

public int getNetworkInterval() {
    return mNetworkInterval;
}

public void setNetworkInterval(int networkInterval) {
    this.mNetworkInterval = networkInterval;
}

public int getNetworkFormat() {
    return mNetworkFormat;
}

public void setNetworkFormat(int networkFormat) {
    this.mNetworkFormat = networkFormat;
    mNetworkFormatEnum = DataFormat.values()[(networkFormat % 3)];
    if ((networkFormat % 3) != networkFormat)
        setmNetworkCompression(true);
    else
        setmNetworkCompression(false);
}

public int getNetworkProto() {
    return mNetworkProto;
}

public void setNetworkProto(int networkProto) {
    this.mNetworkProto = networkProto;
    mNetworkProtocolEnum = NetworkProtocol.values()[networkProto];
}

public int getGpsRate() {
    return mGpsRate;
}
```

```
public void setGpsRate(int gpsRate) {
    this.mGpsRate = gpsRate;
}

public int getAcceRate() {
    return mAcceRate;
}

public void setAcceRate(int acceRate) {
    this.mAcceRate = acceRate;
}

public int getBaroRate() {
    return mBaroRate;
}

public void setBaroRate(int baroRate) {
    this.mBaroRate = baroRate;
}

public DataFormat getmFileFormatEnum() {
    return mFileFormatEnum;
}

public void setmFileFormatEnum(DataFormat mFileFormatEnum) {
    this.mFileFormatEnum = mFileFormatEnum;
}

public DataFormat getmNetworkFormatEnum() {
    return mNetworkFormatEnum;
}

public void setmNetworkFormatEnum(DataFormat mNetworkFormatEnum) {
    this.mNetworkFormatEnum = mNetworkFormatEnum;
}

public NetworkProtocol getmNetworkProtocolEnum() {
    return mNetworkProtocolEnum;
}

public void setmNetworkProtocolEnum(NetworkProtocol mNetworkProtocolEnum) {
    this.mNetworkProtocolEnum = mNetworkProtocolEnum;
}

public void setmNetworkCompression(boolean mNetworkCompression) {
    this.mNetworkCompression = mNetworkCompression;
}

public boolean getmNetworkCompression() {
    return mNetworkCompression;
}

// For viewing purposes
@Override
public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append(mGpsRate);
    builder.append(",");
    builder.append(mAcceRate);
    builder.append(",");
    builder.append(mBaroRate);
    builder.append(",");
    builder.append(mFileInterval);
```

```
        builder.append(",");
        builder.append(mFileFormat);
        builder.append(",");
        builder.append(mNetworkInterval);
        builder.append(",");
        builder.append(mNetworkFormat);
        builder.append(",");
        builder.append(mNetworkProto);
        return builder.toString();
    }

    public void fromString(String s) {
        String[] tokens = s.split(",");
        mGpsRate = Integer.parseInt(tokens[0]);
        mAcceRate = Integer.parseInt(tokens[1]);
        mBaroRate = Integer.parseInt(tokens[2]);
        mFileInterval = Integer.parseInt(tokens[3]);
        mFileFormat = Integer.parseInt(tokens[4]);
        mNetworkInterval = Integer.parseInt(tokens[5]);
        mNetworkFormat = Integer.parseInt(tokens[6]);
        mNetworkProto = Integer.parseInt(tokens[7]);
    }

}
```

TrackFileHandler.java

tabsize

```
package fi.jyu.vesepuup.energytestg16;

import android.app.Service;
import android.os.Environment;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * Veli-Mikko Puupponen 2016
 */

/**
 * Class for writing the sample data to th external storage location.
 */
public class TrackFileHandler {
    private Service mHostService;
    private String mFileName = null;
    private FileOutputStream mFileStream = null;

    public boolean openFile(String name, Service host) {
        File external;
        mHostService = host;
        mFileName = name;
        if (externalStorageWritable()) {
            external = Environment.getExternalStorageDirectory();
            File externalDirectory = new File(external.getAbsolutePath() + "/tracks");
            if (!externalDirectory.exists())
                externalDirectory.mkdirs();
            File externalFile = new File(externalDirectory, mFileName);
            try {
                mFileStream = new FileOutputStream(externalFile);
```

```
            } catch (FileNotFoundException e) {
                mFileStream = null;
            }
        }
        try {
            if (mFileStream == null)
                mFileStream = mHostService.openFileOutput(mFileName, mHostService.MODE_PRIVATE);
        } catch (FileNotFoundException e) {
            mFileStream = null;
            return false;
        }
        return true;
    }

    public boolean writeFile(String payload) {
        if (mFileStream == null)
            return false;
        try {
            mFileStream.write(payload.getBytes());
            mFileStream.write(Integer.toString(payload.length()).getBytes());
            mFileStream.flush();
        } catch (IOException e) {
            closeFile();
            return false;
        }
        return true;
    }

    public void closeFile() {
        if (mFileStream == null)
            return;
        try {
            mFileStream.close();
        } catch (IOException e) {
        }
        mFileStream = null;
    }

    public boolean externalStorageWritable() {
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            return true;
        }
        return false;
    }

}
```

activity_control.xml

tabsize

```
<?xml version="1.0" encoding="utf-8"?>

<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="config" type="fi.jyu.vesepuup.energytestg16.TrackerConfiguration" contentType="fi.jyu.vesepuup.
      energytestg16.TrackerConfiguration"/>
    </data>
<RelativeLayout
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_control"
    android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="fi.jyu.vesepuup.energytestg16.ControlActivity">


<Button
    android:text="@string/button_text_start"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button_start"
    android:onClick="starButtonHandler"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />

<EditText
    android:text="@={config.gpsRate}"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberSigned"
    android:ems="10"
    android:id="@+id/edGpsRate"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

<EditText
    android:text="@={config.acceRate}"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberSigned"
    android:ems="10"
    android:id="@+id/edAccRate"
    android:layout_below="@+id/edGpsRate"
    android:layout_alignLeft="@+id/edGpsRate"
    android:layout_alignStart="@+id/edGpsRate"
    android:layout_marginTop="10dp" />

<EditText
    android:text="@={config.baroRate}"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberSigned"
    android:ems="10"
    android:id="@+id/edBaroRate"
    android:layout_below="@+id/edAccRate"
    android:layout_alignLeft="@+id/edAccRate"
    android:layout_alignStart="@+id/edAccRate"
    android:layout_marginTop="10dp" />

<EditText
    android:text="@={config.fileInterval}"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberSigned"
    android:ems="10"
    android:id="@+id/edFileInt"
    android:layout_below="@+id/edBaroRate"
    android:layout_alignLeft="@+id/edBaroRate"
    android:layout_alignStart="@+id/edBaroRate"
    android:layout_marginTop="10dp"/>

<EditText
```

```
        android:text="@={config.fileFormat}"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="numberSigned"
        android:ems="10"
        android:id="@+id/edFileForm"
        android:layout_centerVertical="true"
        android:layout_below="@+id/edFileInt"
        android:layout_alignLeft="@+id/edFileInt"
        android:layout_alignStart="@+id/edFileInt"
        android:layout_marginTop="10dp"/>

    <EditText
        android:text="@={config.networkInterval}"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="numberSigned"
        android:ems="10"
        android:id="@+id/edNetInt"
        android:layout_below="@+id/edFileForm"
        android:layout_alignLeft="@+id/edFileForm"
        android:layout_alignStart="@+id/edFileForm"
        android:layout_marginTop="10dp"/>

    <EditText
        android:text="@={config.networkFormat}"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="numberSigned"
        android:ems="10"
        android:id="@+id/edNetFor"
        android:layout_below="@+id/edNetInt"
        android:layout_alignLeft="@+id/edNetInt"
        android:layout_alignStart="@+id/edNetInt"
        android:layout_marginTop="15dp" />

    <EditText
        android:text="@={config.networkProto}"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="numberSigned"
        android:ems="10"
        android:id="@+id/edNetPro"
        android:layout_below="@+id/edNetFor"
        android:layout_alignLeft="@+id/edNetFor"
        android:layout_alignStart="@+id/edNetFor"
        android:layout_marginBottom="10dp" />

    <TextView
        android:text="@string/gps_rate_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/edGpsRate"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/textViewGpsRate" />

    <TextView
        android:text="@string/acce_rate_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/edAccRate"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
```

```
        android:id="@+id/textViewAcceRate" />

    <TextView
        android:text="@string/baro_rate_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textViewBaroRate"
        android:layout_alignTop="@+id/edBaroRate"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:text="@string/file_rate_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textViewFileInterval"
        android:layout_alignTop="@+id/edFileInt"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:text="@string/file_format_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textViewFileFormat"
        android:layout_alignTop="@+id/edFileForm"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:text="@string/net_rate_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/edNetInt"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/textViewNetInterval" />

    <TextView
        android:text="@string/net_format_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/edNetFor"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/textViewNetFormat" />

    <TextView
        android:text="@string/net_protocol_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/edNetPro"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/textViewNetProto" />

    <TextView
        android:text=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/edNetPro"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/textViewConfigLine" />
```

```
</RelativeLayout>
</layout>
```

AndroidManifest.xml

tabsize

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fi.jyu.vesepuup.energytestg16">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <uses-feature android:name="android.hardware.location.gps"
        android:required="true"/>
    <uses-feature android:name="android.hardware.location"
        android:required="true"/>
    <uses-feature android:name="android.hardware.location.network"
        android:required="true"/>
    <uses-feature android:name="android.hardware.sensor.accelerometer"
        android:required="true"/>
    <uses-feature android:name="android.hardware.sensor.barometer"
        android:required="true"/>
    <uses-feature android:name="android.hardware.sensor.gyroscope"
        android:required="true"/>
    <uses-feature android:name="android.hardware.sensor.hifi_sensors"
        android:required="true"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ControlActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:description="@string/process_description"
            android:enabled="true"
            android:exported="false"
            android:isolatedProcess="false"
            android:label="@string/process_name"
            android:name=".TrackerService">
        </service>
    </application>

</manifest>
```

Integers.xml

tabsize

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="server_port_number">10000</integer>

</resources>
```

strings.xml

tabsize

```
<resources>
    <string name="app_name">Energy Test G16</string>
    <string name="process_name">Tracker service</string>
    <string name="process_description">Location tracker background service</string>

    <string name="button_text_start">Start</string>
    <string name="button_text_stop">Stop</string>

    <string name="gps_rate_label">GPS rate</string>
    <string name="acce_rate_label">Accelerometer rate</string>
    <string name="baro_rate_label">Barometer rate</string>
    <string name="file_rate_label">File write int.</string>
    <string name="file_format_label">File format</string>
    <string name="net_rate_label">Network int.</string>
    <string name="net_format_label">Network format</string>
    <string name="net_protocol_label">Network proto.</string>

    <string name="button_text_ok">Ok</string>
    <string name="button_text_cancel">Cancel</string>
    <string name="dialog_title_permissions">Missing permissions</string>
    <string name="dialog_title_sensors">Missing Sensors</string>
    <string name="dialog_text_sensors">Some sensors unavailable. Will not continue.</string>
    <string name="dialog_text_permissions">We need these functions: </string>
    <string name="dialog_text_missing_permissions">Permissions missing. Please go to settings to enable them.</string>

    <string name="perm_loc_c">Coarse location</string>
    <string name="perm_loc_f">Fine location</string>
    <string name="perm_stor_r">Storate read</string>
    <string name="perm_stor_w">Storage write</string>

    <string name="config_id_string">Configurationstr</string>



    <string-array name="permission_names">
        <item>@string/perm_loc_c</item>
        <item>@string/perm_loc_f</item>
        <item>@string/perm_stor_r</item>
        <item>@string/perm_stor_w</item>
    </string-array>

    <string name="server_ip_address">127.0.0.1</string>

</resources>
```