LAPPEENRANNANT TEKNILLINEN YLIOPISTO
TIETOTEKNIIKAN OSASTO

25.9.2007

Ti5004000 Kandidaatintyö

Verkkonaapuruston valvontatehtävien siirto mobiililaitteelta kiinteälle laitteelle PeerHood -verkossa

Jani Wunsch
0222719 – TITE V

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto
Jani Wunsch

Verkkonaapuruston valvontatehtävien siirto mobiililaitteelta kiinteälle laitteelle PeerHood –verkossa.

Kandidaatintyö
2007

41 sivua, 5 kuvaa, 1 taulukkoa ja 1 liite.
Tarkasta: Professori Jari Porras

Hakusanat: tehtävä siirto, mobiililaite, peerhood, verkkonaapurusto, virransäästö, mobiili palvelut, henkilökohtainen mobiililaite
Keywords: task migration, mobile device, peerhood, network neighborhood, power management, mobile services, personal trusted device

PeerHood –verkon mobiililaitteiden akkutehon säästämiseksi siirretään mobiililaitteen verkkonaapuruston valvontatehtävät kiinteälle laitteelle. Valvontatehtävien siirto on tarkoitus tehdä silloin, kun laite pysyy paikallaan, esimerkiksi toimisto tiloissa. Laitteen pysyessä paikallaan voidaan verkkonaapurustoa seurata kiinteän laitteen resursseilla ja päivittää verkkomuutokset mobiililaitteelle tarvittaessa.

Mobiililaitteen ollessa vain kuuntelutilassa laite säästää akkutehoa, koska sen ei tarvitse aktiivisesti lähettää dataa verkkolaitteillaan. Verkkolaitteet pysyvät lepotilassa ja odottavat vain tulevaa dataa. Verkkonaapuruston valvontatehtävien siirto ei kuitenkaan vaikuta käyttäjän palveluiden hyödyntämiseen, joten verkkolaitteen akkutehon säästö riippuu suuresti käyttäjän toimista, käyttäjä voi edelleen käyttää muiden PeerHood laitteiden palveluita tai tarjota omiaan.

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology
Jani Wunsch

Migration of network neighborhood monitoring tasks from mobile device to fixed device in PeerHood network.

Bachelor's thesis.
2007

41 pages, 5 figures, 1 tables and 1 appendices.
Supervisor: Professor Jari Porras

Keywords: task migration, mobile device, peerhood, network neighborhood, power management, mobile services

To extend mobile devices battery lifetime in PeerHood network, migration of network neighborhood management tasks to fixed defies is implemented. Migration of management tasks is meant to be used when the mobile device is known to stay stationary, for example in work or office conditions. When the device is known to stay stationary it is possible to delegate the management tasks to a fixed device which will update the mobile devices neighborhood information when needed.

When the mobile device is only in listening state it will uses its networking devices power conservation methods to lower power drain. The networking devices will stay in idle mode and wait for incoming connections or data. Migration of the network neighborhood management tasks to another device does not affect usability of the mobile device. User is able to use all services found on other PeerHood enabled devices, or offer services himself/herself to other users.

ALKUSANAT

Kandidaatintyöni oli pääpiirteissään käytännön toteutukseen painottuva eikä niinkään tutkimukseen. Tästä johtuen kandidaatintyödokumentaationi tulee myös painottumaan enemmän käytännön toteutukseen kuin teoriaosuuteen.

Teoriaosuutta pienentää huomattavasti tehtävänanto, tehtävänä oli toteuttaa verkkonaapuruston ylläpitotehtävien siirto mobiililaitteelta kiinteälle laitteelle. Oletuksen siirron tekemiseen on käyttäjän halu ja perusteluna siihen on akkuvirran säästäminen.

Työssä tullaan käsittelemään lähinnä PeerHood:n Linux-versiota, koska myös työ toteutettiin Linux-versiota silmällä pitäen. Symbian OS-versio eroaa jonkin verran Linux-versiosta muun muassa sen vuoksi, että julkinen Symbian SDK (Software Development Kit) ei tue taustaprosessien luontia.

Kiitokset kandidaatintyötä tehdessäni saamasta Linux ja PeerHood -opastuksesta Arto Hämäläiselle. Oikoluvussa avustuksesta kiitokset Janne Piponiukselle. Kiitokset kandidaatintyöni tarkastajalle Jari Portaalle, saamastani mielenkiintoisesta ja haastavasta kandidaatintyön aiheesta.

SISÄLLYSLUETTELO

LYHENTEET

PH	PeerHood
MAC	Media Access Control (address)
BT	Bluetooth
WLAN	Wireless Local Area Network
GPRS	General Packet Radio Service
SDP	Service Discovery Protocol
PSM	Protocol Service Multiplexer
L2CAP	Logical Link Control and Adaptation Protocol
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
STL	Standard Template Library
PTD	Personal Trusted Device
x86	Generic term refering to multiple CISC instruction set architectures developed by Intel
x87	Generic term refering to 80386 instruction set architecture floating point co-processor compatible floating point unit
CISC	Complex Instruction Set Computer
RISC	Reduced Instruction Set Computer
ARM	Advanced RISC Machines / The Acorn RISC Machine (referring to ARM architecture RISC processors)
ALU	Arithmethic Logic Unit
FPU	Floating Point Unit
MMX	Multi-Media Extesions (Intel's additional instruction set to x86)
3dNow!	Advanced Microdevices Ltd. 's competitor to Intel's MMX
SIMD	Single Instruction Multiple Data
SSE	Streaming Simd Extensions (Intel)
CPU	Central Processing Unit
RAM	Random Access Memory
RD-RAM	Rambus DRAM
SD-RAM / SDR	Singe Data Rate RAM
DDR / DDRx	Double Data Rate (RAM (where x refers to version of DDR))
PDU	Usually refers to Protocol Data Unit (name not specified in PeerHood documentation)

1. JOHDANTO

Kandidaatintyön aiheena oli toteuttaa verkkonaapurustotiedon ylläpitotehtävien siirto mobiililaitteelta kiinteälle laitteelle PeerHood –verkossa. Kandidaatintyö tehtiin Lappeenrannan Teknillisen Yliopiston Tietoliikennetekniikan laitokselle. PeerHood on myös tietoliikennetekniikan laitoksen kehittämä alusta. PeerHood tarjoaa mahdollisuuden rakentaa nopeasti ad-hoc verkkoja erilaisten laitteiden välille ja mahdollistaa samalla palveluiden mainostamisen ja tarjoamisen muille verkon laitteille.

PeerHood –verkon laitteet voivat olla mobiililaitteita tai normaaleita tietokonelaitteita, PeerHood:ia voidaan ajaa pääasiassa Linuxiin pohjautuvissa käyttöjärjestelmissä, mutta siitä on myös testiluontoinen toteutus Symbian OS käyttöjärjestelmälle.

1.1 Työn aihepiiri ja johdatus aiheeseen

PeerHood tarjoaa työkalut ad-hoc –verkkojen rakentamiseen mobiililaitteille sekä normaaleille tietokoneille. PeerHood koostuu pääasiassa ohjelmointikirjastosta, daemon-prosessista sekä verkkoteknologiakohtaisista ”plugineista”.

Daemon on PeerHood:n taustaprosessi joka huolehtii verkkonaapuruston ylläpidosta, sen tehtäviin kuuluu eri verkkoteknologioiden kantoalueella olevien laitteiden löytäminen sekä niiden tietojen keruu ja ylläpito. Laitekohtaiset tiedot pitävät sisällään laitteiden tukemat verkkoteknologiat, laiteosoitteet (MAC) sekä verkko-osoitteet verkkolaitteisiin sekä laitteen tarjoamien palveluiden tiedot ja portit.

PeerHood -kirjasto toimii ohjelmointirajapintana PeerHood –ohjelmille (palveluille) sekä tarjoaa ohjelmoijalle erilaisia verkkoympäristöön liittyviä palveluita tai toimintoja joko suoraan plugineiden tai daemon-prosessin välityksellä. Kirjaston avulla ohjelmoija voi mm. ottaa yhteyttä toisiin laitteisiin, etsiä palveluita tai laitteita daemonin tietokannasta tai rekisteröidä omia palveluitaan verkkoon mainostettavaksi.

Pluginien tehtävä PeerHood:ssa on abstraktioida eri verkkoteknologioihin perustuvat toiminnot samojen funktioiden alle, jolloin ohjelmoijan ei tarvitse käyttää verkkoteknologiakohtaisia komentoja yhteyksien luomiseen, katkaisemiseen tai tiedonsiirtoon. Pluginien luomaa abstraktiota hyödynnetään myös daemon-prosessin kommunikaatiossa sekä laitetietokannan ylläpidossa.

1.2 Ongelman esittely ja rajaukset

Kandidaatintyön päätehtävänä oli toteuttaa verkkonaapuruston ylläpitotoimien suorituksen siirto mobiililaitteelta kiinteälle laitteelle, tämän lisäksi alitehtävän tuli toteuttaa laitteiden parametrisointi. Laitteiden parametrisoinnin tarkoitus on esittää laitteen teho-arvio parametreina sellaisella tarkkuudella, että käyttäjä tai ohjelma pystyy arvioimaan onko laite tietokoneresurssi vai mobiililaite.

Naapurustoinformaation ylläpitotehtävän siirrolla tarkoitetaan verkkonaapuruston laitteiden etsinnän ja tietojen poimimisen siirtoa mobiililaitteelta etäsuoritukseen, syynä etäsuoritukselle on akkutehon säästäminen. Oletuksena tehtävänsiirto tapahtuu käyttäjän halusta ja toimesta, toinen oletus on, että tällöin käyttäjä aikoo pysyä paikallaan pidemmän aikaa.

Esimerkiksi käyttäjä työskentelee toimistoympäristössä, jolloin mobiililaite pysyy paikallaan. Tällöin tehtävänsiirto on sekä järkevää, että mahdollista.

Ongelmat

- Verkkonaapuruston ylläpitotehtävien siirto laitteelta toiselle
- Laitteiden parametrisointi: tarkkuus, jolla laitetyypit voidaan erottaa toisistaan

Rajaukset

- Tehtävänsiirron järkevyydestä päättää käyttäjä
- Laitetyyppiä, laitteen muistinmäärää, tallennusmedian kokoa ja verkkoteknologioita ei tutkita automaattisesti parametrisoinnissa.

2. TEORIAOSUUS

Teoriaosuuden alussa perustellaan lyhyesti miksi verkkonaapurustotiedon ylläpitotehtävien siirto mobiililaitteelta kiinteälle laitteelle säästää mobiililaitteen akkua. PeerHood tukee Bluetooth, WLAN (Wireless Local Area Network) ja GPRS (General Packet Radio Service) verkkoteknologioita, joista työssä oleellisia ovat Bluetooth ja WLAN. Molemmat teknologiat sisältävät integroitua metodeja lähettimen käyttämän virran säästämiseksi.

Periaatteessa molempien verkkoteknologioiden virransäästö perustuu lähettimen sulkemiseen tai virtatilan madaltamiseen silloin kuin lähetintä ei käytetä. WLAN:in tapauksessa tekniikkaa nimitetään ”Power Save Polling” tekniikaksi, jolloin lähetin suljetaan tietyiksi aikaväleiksi. [2]

Bluetooth tekniikassa on linkkitasolla useampia virransäästötiloja, jotka toimivat keskenään hieman eri tavalla. Pääpiirteenä kuitenkin lähetin tiputetaan joko matalampaan virrankulutustasoon tai sammutetaan silloin kun sillä ei ole dataa lähetettävänä. [3]

Huom. Työssä suorittimia ja niiden tehoa käsittelevässä osuudessa kaikkiin Intelin x86 –arkkitehtuureihin liittyessä puhutaan x86 –arkkitehtuurista. Työn aihepiirin osilta ei ole perusteltua perehtyä x86 –arkkitehtuurin aikaisempiin 8-,16- ja 32 -bittisiin versioihin. Myöskään arkkitehtuureissa aikaisemmin irrallisena esiintynyttä liukulukusuoritinta ei käsitellä, vaan työssä viitataan joko FPU yksikköön, tai vanhempaan nimitykseen x87.

2.1 Työn käsitteiden esittely

Dokumentissa esiin tulevien käsitteiden esittely lyhyesti.

Suorittimet – parametrisaattori

- ALU : Arithmetic and Logic Unit : Suorittimen kokonaisluku yksikkö
- FPU : Floating Point Unit : Suorittimet liukuluku yksikkö
- SIMD – Single Instruction Multiple Data – Vektorioperaatioita
- NEON : ARM suorittimien SIMD –käskykannan nimi
- MMX/SSE : Intel:in kehittämät SIMD –käskykannat
- 3dNow! : AMD:n kehittämä MMX:n kanssa kilpaillut SIMD –käskykanta

Lyhenteet / käsitteet – PeerHood - yleistä

- PTD (Personal Trusted Device) : Mikä tahansa mobiililaite, joka kykenee ajamaan PeerHood:ia. Laitteen ajatellaan olevan persoonallinen ja täten aina vain yhden henkilön käytössä. Esim. matkapuhelin tai kämmentietokone.
- PDU: PeerHood kirjaston ja daemonin välinen enumeroitu 8 bittinen komentomuuttuja.
- BT : Bluetooth
- WLAN : Wireless Local Area Network
- GPRS : General Packet Radio Service
- PH : PeerHood
- Plugin : verkkoteknologia kohtainen luokka, toteuttaa verkkoteknologia kohtaiset Connect(), Disconnect(), Close(), Send() ja Recv() funktiot, sekä hoitaa Inquiry- ja Advert -toimenpiteet.
- STL (Standard Template Library)
- CallBack –rajapinta : Rajapinta jonka kautta PH –kirjasto välittää ilmoitusviestejä
- Notification Event : yllä mainittu ilmoitusviesti
- PSM (Protocol/Service Multiplexer) : Bluetooth spesifikaation portti
- Verkkoprototyyppi (prototype) : PeerHood:n tapa nimetä / kutsua eri verkkoteknologioita ja niiden plugineita.

PeerHood daemon:

- Laitetietokanta: Muunneltu linkitetty lista STL -kirjastosta joka pitää sisällän tiedot laitteista ja laitteiden tarjoamista palveluista, sekä laitteiden tukemista verkkoyhteyksistä.
- Pää-algoritmi: Daemonin toiminnasta vastaava algoritmi joka huolehtii kommunikaatiosta PeerHood -kirjaston kanssa sekä laitetietokannan ylläpidosta ja päivityksestä.

PeerHood –kirjasto:

- Monitoring, monitorointi, aktiivinen valvonta: PH –kirjaston menetelmä valvoa verkon laitetta
- SignalMonitor: Signaalitason laitteen valvonta, tarkkaillaan langattoman linkin kuuluvuutta.

”rMon” –palvelu; kandidaatintyön pää-aihe, verkkonaapurustoinformaation etäpäivitys

- Server : palvelun toteuttavan ohjelmiston osa, jota ajetaan palvelimella.
- Client: palvelun toteuttavan ohjelmiston asiakas osa, jolla otetaan yhteyttä palvelimeen.
- Customer / Asiakas: Server –ohjelman tietorakenne asiakaslaitteesta jolle verkkoinformaation päivityspalvelua tarjotaan.
- Ilmoitusviesti (CallBack): CallBack –rajapinnan kautta välitetyt ilmiotusviestit kirjastolta sovellukselle.

2.2 Taustatietoa tekniikasta

PeerHood koostuu siis käytännössä kolmesta osasta: daemonista, kirjastosta ja verkkoplugineista. Daemon hoitaa verkkonaapurustoinformaation ylläpitotehtävät, pitää tietokantaa naapuruston laitteista ja niiden tarjoamista palveluista, sekä tietenkin oman laitteensa tarjoamista palveluista ja niiden mainostamisesta laitteille jotka kyselevät tietoa.

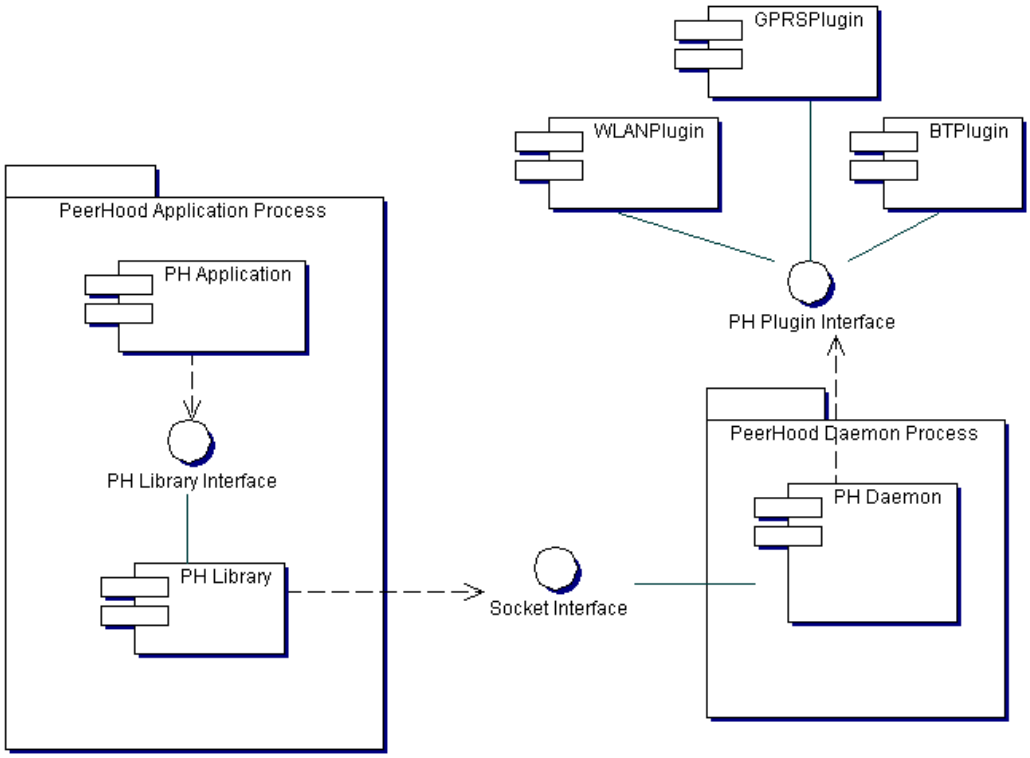
Kirjasto tarjoaa ohjelmoijalle mahdollisuuden etsiä laitteita ja palveluita daemonin tietokannasta, sekä lisätä ja poistaa omia palveluita verkkoon.

Kirjaston avulla ohjelmoija pystyy myös käyttämään eri verkkoteknologioita abstraktoivien virtuaalifunktioiden avulla välittämättä käytössä olevasta verkkoteknologiasta. Kirjaston avulla voidaan myös valvoa muita laitteita tai verkkolinkin kuuluvuustasoa muihin laitteisiin, sekä reagoida erilaisiin huomautusviesteihin CallBack –liittymän avulla.

Verkkoteknologiakohtaiset pluginit taas toteuttavat PeerHood:n virtualisoimat funktiot verkko-operaatioille, kuten Connect, Disconnect, Send ja Recv funktiot. Pluginit myös pitävät oman verkkotyyppinsä kaltaisten verkkolaitteiden tietokantaa. Lopullinen naapurustotietokanta kootaan yhdistämällä pluginien tiedot. Käytännössä siis yhdestä laitteesta on niin monta osatietokantaa kuin sillä on tuettuja verkkoteknologioita käytössään, näistä osatietokannoista rakennetaan lopullinen tietokanta daemonin toimesta, ja tätä lopullista tietokantaa myös ohjelmat käyttävät kirjaston kautta.

Mahdollisuus tietoliikenneyhteyden signaalin laadun tarkkailuun mahdollistaa ohjelman valvoa linkin tasoa esimerkiksi Bluetooth-yhteydellä ja mikäli linkin taso heikkenee liikaa, voidaan ohjelmassa reagoida ”PH_LINK_WEAK” huomautusviestiin, jolloin ohjelma voi pyytää kirjaston kautta verkkoteknologian vaihtoa esimerkiksi paremman kantomatkan WLAN yhteyteen. Näin PeerHood:ssa pystytään toteuttamaan ns. ”Seamless Connectivity”.

PeerHood:n komponentit ja niiden välinen kommunikaatio selviää kuvasta 1. PeerHood –ohjelmat kommunikoivat PeerHood kirjasto-interfacen avulla kirjastokomponentin kanssa. Kirjastokomponentti keskustelee käyttöjärjestelmätason lokaalisocketin kautta oman daemoninsa kanssa. Daemon ja kirjasto käyttävät plugineita hyväkseen PeerHood plugin-interfacen kautta. Huomautuksena, komponenttikaavio on Linux version mukainen, Symbian OS versio-eroaa hieman käyttöjärjestelmätason toiminnallisuuksien puutteiden vuoksi.



Kuva 1: PeerHood komponenttikaavio. [4]

2.2.1 PeerHood daemon

Kuten edellä mainittu, PeerHood daemon pitää yllä tietokantaa verkkonaapurustosta sekä toimii yhteistyössä kirjaston kanssa. Esittelen tässä kappaleessa daemon-prosessin toiminnan pääpiirteissään, vielä tarkemmat kuvaukset päivitystapahtumista seuraavat dokumentin loppuosassa, jossa esitellään käytännössä naapurustoinformaation päivitys.

Daemon-prosessia ajetaan taustalla jatkuvasti mobiili- tai kiinteän laitteen ollessa päällä,

prosessin käynnistyessä se lataa löytyvät tai käyttöönotetut verkkoteknologiakohtaiset pluginit. Pluginin käynnistyessä plugin käynnistää omat Advert- ja Inquiry-säikeet, joiden tehtävät ovat etsiä muita saman verkkotyypin laitteita kuuluvuusalueella sekä vastaanottaa verkkoinformaatiopyyntöjä. Kun daemon on saanut ladattua tarvittavat pluginit, se siirtyy ajamaan omaa pääalgoritmiaan.

Pluginien Inquiry-säikeen tehtävä on etsiä verkkonaapurustosta uusia laitteita. Bluetooth-plugin hoitaa laitteiden etsinnän omalla ajuritason menetelmällään lähettämällä Bluetoothin Inquiry-paketin, johon muut laitteet vastaavat.

WLAN –plugin hoitaa laitteiden etsinnän lähettämällä broadcast –UDP (User Datagram Protocol) paketin verkkolaitteen löytämään verkkoon. Paketti lähetetään tunnettuun porttiin, jota jokainen PeerHood –WLAN-laite kuuntelee. Jokainen WLAN-yhteyttä käyttävä PeerHood –laite vastaa tähän viestiin unicastina lähetettävällä UPD -paketilla.

GPRS –plugin eroaa toiminnallaan laitteiden etsinnän suhteen, koska GPRS –yhteyttä käytettäessä tarvitaan välityspalvelin. Lyhykäisyydessään GPRS:n kautta PeerHood laitteet löydetään välityspalvelimen tietokannasta mikä vaatii sen, että jokainen GPRS-yhteyttä käyttävä PeerHood –laite rekisteröityy välityspalvelimelle, jolloin kyselyt suunnataan välityspalveluun.

Koska GPRS –yhteys on PeerHood:ssa enemmänkin varakanava verkkoon, eikä pääasiallisesti käytettävä yhteys, kandidaatintyössäni toteutettu tehtäviensiirto ei ole käytössä GPRS –yhteydellä.

Laitteiden ja palveluiden etsintä

Bluetoothin tapauksessa laitteiden etsintään käytetään Bluetooth:n omaa SDP protokollaa (Service Discovery Protocol). Jotta PeerHood:ia tukevat laitteet voitaisiin erottaa muista BT-laitteista, tallentavat kaikki PeerHood:ia ajavat BT-laitteet niin sanotun PH-tagin BT:n laitetietoihin. SPD –kyselyn suorittava PeerHood-laite etsii tätä tagia laitteen

tiedoista, mikäli tagi löytyy merkitään laite PeerHood –yhteensopivaksi. [4]

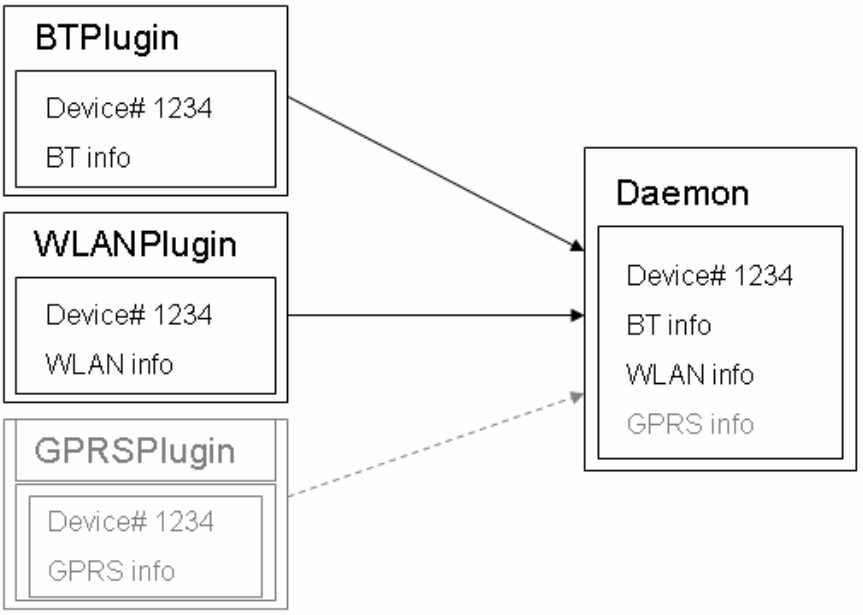
WLAN –yhteyden tapauksessa Peerhood –laitteet tunnistetaan yksinkertaisesti siitä, että ne vastaavat lähetettyyn ”broadcast” -viestiin ”unicast” -viestillä. Viestin sisältö on ennalta määrätty, joten mikäli jokin muu ohjelma sattuisi vastaamaan tähän viestiin, pystytään se erottamaan PeerHood-laitteesta. [4]

Bluetooth:n ollessa kyseessä, uuden laitteen löytyessä sen käyttämät portit selvitetään, jotta otettaessa yhteyttä laitteeseen uudestaan tiedetään minne yhteys tulisi ottaa. BT-yhteydellä käytetään laitteen tiedoista löytynyttä PSM –porttia. WLAN –yhteys poikkeaa tästä käyttäen ennalta määriteltyjä portteja.

Laitetietokantojen päivitys

Pluginin Inquiry -metodin löytäessä uusia laitteita, Inquiry -metodi lähettää löydetyn laitteen verkkolaitteelle pyynnön laitteen tiedoista. PeerHood:ssa on muutamia eritasoisia pyyntöjä. Pyynnöstä riippuen vastaanottavan laitteen Advert –säie vastaa joko vain omia tietojaan tai koko sen näkemän naapuruston tiedot. Mikäli pyydetään tietoja vain laitteesta itsestään, voidaan kyselyä tarkentaa ja kysyä vain osa laitteen tiedoista. Jos laitteelta kysytään koko verkkonaapuruston tietoa, lähetetään vastaajalle automaattisesti myös jokaisen laitteen tarjoamien palveluiden tiedot.

Kun kyselyn suorittanut laite saa vastauksena joko kohdelaitteen tiedot tai kohteen koko naapuruston tiedot, päivittää kyselyn suorittanut plugin omaa laitetietokantaansa. Mikäli pluginin laitetietokannassa on tapahtunut muutoksia, päivitetään daemonin laitetietokantaa. Daemon-prosessin laitetietokanta on käytännössä kokoelma pluginien tietokannoista.



Kuva 2. PeerHood -laitetietojen yhdistäminen.

Mikäli PeerHoodia ajetaan vakioasetuksilla laitehakuihin tulevat vastaukset sisältävät tiedot laitteen verkkoyhteyksistä sekä sen tarjoamista palveluista. Yksittäisiltä laitteilta voidaan haussa jättää pois muun muassa palvelutietojen kysely, mutta tällaisten asetusten käyttäminen muuhun kuin testaustarkoitukseen ei ole perusteltua.

Laitekohtainen siirrettävä data rakennetaan PeerHood:n omalla serialisointimenetelmällä, joka muuntaa laitetietokannan laiteoliot merkkijonoksi. Datan vastaanottava laite rakentaa laiteolion kopion tästä merkkijonosta ja lisää omaan laitetietokantaansa.

2.2.2 PeerHood -kirjasto

PeerHood -kirjasto toimii rajapintana, jonka avulla PeerHood -ohjelmat voivat käyttää laitetietokantaa, verkkoyhteyksiä ja PeerHood:n toimintoja. Jokaisen PeerHood –ohjelman tulee käynnistyessään rekisteröidä CallBack –rajapinta sekä initialisoida PeerHood –kirjasto. CallBack –rajapinnan kautta kirjasto ilmoittaa ohjelmalle muutoksista ja muista tapahtumista ”Notification Event” –viestillä. Kirjaston initialisointi mahdollistaa kirjaston käytön sekä varmistaa, että kirjastosta on käynnissä vain yksi instanssi (Singleton Design).

Osa kirjaston tarjoamista palveluista toimii suoraan verkkopluginien avulla, osa taas kommunikoi paikallisen daemon-prosessin kanssa. Tietoliikenne lokaaliin daemoniin tapahtuu käyttöjärjestelmätason Socket –rajapinnan kautta.

Daemonia kirjasto hyödyntää muun muassa palveluiden etsimisen, rekisteröinnin ja poiston yhteydessä. Laitteita tai palveluita haettaessa kirjasto pyytää daemonilta laitetietokannan, jonka daemon lähettää kirjastolle. Laitetietokannasta voidaan suoraan kirjaston funktioilla joko selata laitteita tai etsiä tietyn nimistä palvelua. Etsittäessä tiettyä palvelua kirjasto palauttaa listan kaikista laitteista joissa kyseinen palvelu pyörii.

Tietoliikenteen osilta kirjasto tarjoaa ohjelmalle virtuaaliset funktiot yhteyden luomiseen, katkaisemiseen ja tietopakettien lähettämiseen tai vastaanottamiseen. Näin ollen ohjelmoijan ei tarvitse perehtyä verkkoteknologian omiin funktioihin tai protokolliin vaan kaikki toiminta tapahtuu automaattisesti PeerHood:n sisällä. Esimerkiksi BT-yhteys hyödyntää L2CAP (Logical Link Control and Adaption Protocol) protokollaa, kun WLAN yhteyksillä käytetään TCP/IP (Transmission Control Protocol / Internet Protocol) -yhteyksiä.

PeerHood –kirjasto ei rajoita käytettävien yhteyksien määrää millään tavalla, vaan aina uusi yhteys luotaessa luodaan uusi yhteysobjekti jota käytetään tiedon välittämiseen. Näin ollen PeerHood –ohjelmat voivat ottaa useita yhteyksiä joko yhteen tai useaan laitteeseen ja yhteydet on helppo sisällyttää STL –kirjaston mukaisiin automaattisiin tietorakenteisiin kuten jonot, pinot ja listat. Yhteysobjektit tarjoavat myös muutamia toiminnallisia funktioita yhteyden tilan tarkkailuun.

Näiden metodien lisäksi PeerHood –kirjasto tarjoaa mahdollisuuden aktiiviseen laitteiden valvontaan, laitteita voidaan seurata joko signaalitasolla tai ilman. Signaalitason seurannassa PeerHood tarkkailee kyseisen verkkoteknologian ajuritason tietoja signaalin voimakkuudesta ja palauttaa ilmoitusviestimenettelyllä (CallBack) tiedon mikäli signaali on heikko. Signaaliton seuranta on periaatteessa kyseisen laitteen ping –testausta, jolloin

voidaan vain huomata jos laite katoaa verkosta. Kirjasto luo automaattisesti omat säikeensä valvontaa varten, joten valvonnan käyttäminen on huomattavan helppoa.

2.3 Tutkimusongelman esittely

Kandidaatintyöni on melkoisen toteutuspainotteinen, joten suoranaisesti työssäni ei ollut tutkimuksellista ongelmaa. Ainoa lähellekään teoreettinen ongelma työssä oli keksiä, kuinka verkkoinformaation etävalvontatehtävä pystytään toteuttamaan mahdollisimman vähän mobiililaitetta kuormittavaksi.

Valvontapalvelun tulisi pystyä palvelemaan mahdollisimman laajaa asiakaskuntaa kerrallaan ja toimia luotettavasti ottaen huomioon langattomien verkkoyhteyksien mahdollinen katkeaminen. Mahdollisia ongelmia jotka ohjelman kannalta näkyvät verkko-ongelmina ovat langattoman yhteyden huono kuuluvuus, käyttäjän virheestä tapahtuva kuuluvuusalueelta poistuminen sekä katkeilevasta yhteydestä johtuvat satunnaiset viestien katoamiset.

Valvontaohjelmiston tulisi pitää verkkonaapurusto tiedot mahdollisimman ajan tasalla, sekä myös tunnistaa oikeasti katkenneet tai hävinneet yhteydet satunnaisista virheistä. Käytännössä tämä tarkoittaa sitä, että ensimmäisen virheen ilmetessä ei tule olettaa, että asiakas on hävinnyt kuuluvuusalueelta. Toisaalta toistuvien virheiden tapahtuessa palvelun tulee jossain vaiheessa poistaa asiakas palvelusta, jotta palvelu ei täyty niin sanotuista ”zombieista”.

Parametrisoinnin osilta tutkimusongelmana oli lähinnä toteuttaa sellainen parametrisaattori, joka kykenee erottamaan PTD –laitteet mobiileista- tai kiinteistä tietokonelaitteista suoritustehon perusteella. Parametrisaattorin ei tarvinnut automaattisesti selvittää laitteessa olevan tallennusmedian kokoa tai muistinmäärää.

2.3.1 Tutkimusongelmaan liittyvien aikaisempien tutkimusten esittely

Tutkimusongelman teoriaosuuteen löydetyt lähteet eivät käsittele asiaa samalta kantilta kuin tässä kandidaatintyössä. Monissa lähteissä otetaan kantaa laitteen virransäästöön, mutta virransäästöä haetaan siirtämällä raskaita tehtäviä mobiililaitteen suorituksesta tehokkaampien laskentaresurssien hoidettavaksi.

Eroavaisuus kandidaatintyöhöni onkin siinä, että näissä lähteissä virransäästö saavutetaan laskentatehtävän siirrolla, eikä verkkoliikenne ole suuri vaikuttaja. Kandidaatintyössäni siirrettävä tehtävä ei ole laskentaintensiivinen, vaan se on jatkuvasti taustalla suoritettava tehtävä, jonka suurin akkuvirtaa kuluttava osuus on verkkoliikenne.

Parametrisointi

Parametrisointiohjelman osilta en löytänyt tähän aiheeseen liittyen kovin järkeviä tieteellisiä julkaisuja, julkaisuiden parametrisointi yleensä liittyi joko ajettavan prosessin parametrisointiin siten, että parametreista ymmärretään prosessin vaatimat resurssitarpeet. Toisissa artikkeleissa parametreilla mitattiin ”cluster” ja supertietokoneiden ominaisuuksia, joiden avulla koneita yritettiin kuvata riittävällä tarkkuudella Grid – ympäristöön liittyvässä suoritusresurssien huutokaupassa.

Omaan parametrisaattoriini liittyen suorittimien teoriaosuudesta vastasi jo paljon kandidaatintyötäni aikaisemmin lukemani Ars Technica -sivuston ”CPU Theory & Praxis”-artikkeleiden sisältö. Kyseinen sivusto on IT-alan ammattilaisten ylläpitämä sivusto joka käsittelee tietoteknisiä asioita ohjelmistojen, raudan ja valmiiden laitteiden osilta. [5]

Tärkeimpänä yksittäisenä lähteenä mainittakoon lähde 6, joka keskittyy x86 – CISC-arkkitehtuuria tukevien suorittimien muutoksiin Pentium suorittimesta uudempia kohti mentäessä. Suurin muutos Pentium-suorittimella edeltäjiinsä on, että Pentium ei suorita x86 –natiivikoodia sellaisenaan, vaan tulkkaa koodin oman ytimensä ymmärtämiksi

mikrokäskyiksi. [6]

Nykyisistä x86 –suorittimista kaikkien valmistajien suorittimet toimivat samalla periaatteella kuin Pentium, jokainen suoritin dekoodaa x86 –käskykannan mukaiset käskyt oman ytimensä ymmärtämäksi mikrokoodiksi. Dekoodaus vaiheessa suoritin muun muassa purkaa käskyjä useiksi mikro-operaatioiksi ja koittaa ajaa mahdollisimman monia käskyjä rinnakkain samaan aikaan. [6]

RISC –arkkitehtuuriin pohjautuvat ARM –suorittimet yleisesti ottaen suorittavat vain yhden laskutoimituksen kellojaksoa kohden, poikkeuksena suorittimien SIMD –käskykannan mukaiset vektorioperaatiot. [7] [8]

Johtopäätös: Jotta suorittimien sisäinen rinnakkaisuus vaikuttaisi nopeusindeksiin, täytyy parametrisaattorin pystyä kuormittamaan useita laskentayksiköitä jokaisella laskentakierroksella. Toisin sanoen jokaisella laskentakierroksella täytyy olla useampia laskutoimituksia joissa laskutoimitusten lähtöarvojen ja tulosten tulee olla toisistaan riippumattomia laskennan aikana, jotta suoritin pystyy ajamaan käskyjä läpi rinnakkain.

Tehtävänsiirto

Perehtyessäni tehtävänsiirto-ongelmaan tutustuin aluksi PeerHood:iin ja sen yhteydessä esiteltyyn PTD (Personal Trusted Device) –ideologiaan, PeerHoodia ja PTD –konseptia esitellään lähteissä [1], [16] ja [17]. Kaikki kolme julkaisua liittyvät itse PeerHood:iin, näiden lähteiden sekä PeerHood:n API-spesifikaation perusteella muodostin käsityksen mitä kandidaatintyön tehtävänsiirto-ongelmassa tullaan tekemään.

Lähde [1] käsittelee hajautettujen resurssien hyödyntämistä PeerHood –verkossa sekä esittelee neljännessä kappaleessa esimerkkisovelluksen viivakoodin analysointiin etäresurssissa. Esimerkkisovellus lähettää PTD –laitteelta viivakoodin kuvatiedostona etäresurssille, joka palauttaa mobiililaittelle viivakoodin sisällön.

Lähde [16] esittelee PeerHood:in ideaa, sekä PTD -ideologiaa. Julkaisu esittelee

PeerHood:in verkkoteknologiat ja antaa muutaman esimerkin viestiliikenteestä sekä verkkonaapurustotiedon ylläpidosta. Lähteessä käydään läpi PeerHood:n toimintaa sekä pari sovellusesimerkkiä.

Lähde [17] esittelee kattavammin PTD –konseptia ja esittelee muutamia käyttöskenaarioita sille. Julkaisussa esitellään käyttöskenaariot PTD –laitteelle muutamiin eri tilanteisiin, kuten kulunvalvonta ja opastusjärjestelmiin. Lähde [1] esittelee käytännön sovellusta PeerHood –ympäristössä, sovellus lähettää PTD –laitteella kuvatun viivakoodin kiinteän tietokoneresurssin käsiteltäväksi, tietokone lukee viivakoodista numerosarjan ja lähettää vastauksen takaisin puhelimelle.

Itse tehtävänsiirto-ongelmaan löysin huonosti teorialähteitä, joissa olisi perustana tehtävän siirrolle laitteen pysyminen paikallaan tai käyttäjän halu tehdä niin. Useimmiten lähteissä tarkastellaan tehtävän tai prosessin siirto-ongelmaa, mutta eri kuvakulmasta. Useimmissa lähteissä tehtävä siirretään suoritusresurssien tai suoritusajan säästämiseksi, jolla tavoitellaan samalla myös akkutehon säästöä.

PeerHood:n tapauksessa akkutehon säästöön pyritään vähentämällä verkkoliikenteen määrää, sillä verkkotiedon ylläpitotehtävän ajo on välttämätöntä PeerHood:n toiminnan kannalta. Tehtävä on hyvin kevyt suorittaa, mutta vaatii kohtuullisen usein verkkolaitteiden käyttöä.

Lähteissä [11] ja [12] tehtävänsiirtoa pohditaan suoritustehokkuuden kannalta, ja mittalaitteistoina käytetään tietokoneita eikä mobiililaitteita. Lähde [11] vertailee prosessin ja osatehtävän siirron tehokkuutta Unix-pohjaisissa Cluster- ja MMP (Massively multiprocessing) –koneissa. Lähde [12] vertailee erilaisia dynaamisia tehtävänsiirto skeemoja ja tekee tehomittauksia niiden välillä.

Mobiililaitteilla suoritettavaan tehtävänsiirtoon liittyen materiaalia löytyy lähteistä [13], [14] ja [15]. Lähteessä [13] esitellään automaattista tehtävien paloittelua ja siirtoa etäsuoritukseen ohjelmiston objektitasolla, tehtävänsiirtopäätöksiin vaikuttaa laitteessa

ajettavan ohjelman muistintarve. Lähteessä tehtävänsiirto suoritetaan laitteen resurssien säätämiseksi, sekä pohditaan menetelmiä suorittaa tehtäviä hajautetusti, joiden suorittamiseksi yksin mobiililaitteen resurssit eivät riitä. Dokumentissa tehdään tehovertailua Java –ohjelmilla, jotka kuormittavat laitetta hieman eri tavoin. Dokumentti pohtii tehtävänsiirtoa suoritustehon parantamisen ja akkutehonsäästämisen kannalta, siinä otetaan myös kantaa erityyppisten ohjelmien osittaiseen etäsuoritettavuuteen.

Lähde [14] tutkii kuinka useiden mobiililaitteiden yhteistä toiminta-aikaa voitaisiin pidentää siirtämällä tehtäviä laitteilta toisille. Dokumentissa pohditaan onko tehtävänsiirto laitteen itsensä, sekä muiden laitteiden kannalta kannattavaa, tarkoituksena on tasapainottaa sekä pidentää laitteiden yhteistä jäljellä olevaa akunkestoa. Tehtävänsiirron mielekkyydestä esitetään muutamia arviointilaskelmia.

Lähde myös esittelee lyhyesti ARM-suorittimen ja Orinoco WLAN-laitteen virrankulutukset eri tiloissa, joiden pohjalta myös omassa tehtävänsiirrossani pyritään minimoimaan verkkoliikennettä. Laitteiden virrankulutus on karkeasti noin 5-10% virransäästötilassa, verrattuna toimintatilaan, laitteet menevät näihin virransäästötiloihin kuormituksen tippuessa matalaksi.

Lähteessä [15] tehtävänsiirtoa tutkitaan myös mobiililaitteen akkutehon säästämiseksi, lähteessä arvioidaan Markovianin mallilla tehtävänsiirtoa. Tarkoitus on tehdä arvio tehtävästä ja laskea, onko tehtävän suorittaminen kannattavaa etäresurssissa vai lokaaleilla resursseilla. Lähteen migraatiomalli mobiililaitteen ja kiinteän laitteen välillä on tuttu client-server –malli.

Johtopäätös: Teoriaosuudessa käsitellyistä materiaaleista saadaan tukea alkuperäiselle hypoteesille, että mobiililaitteen akkutehoa pystytään säästämään vähentämällä kuormitusta suorittimelta ja verkkolaitteilta. Tämän työn aihepiirissä keskitytään verkkolaitteiden kuormituksen minimoimiseen.

2.4 Ratkaisutavan esittely

Ennen ratkaisutavan pohtimista ratkaisuksi ehdotettiin muun muassa joko pääalgoritmin vaiheistamista tai jonkinlaisen palvelutyyppisen toteutuksen tekemistä. Parametrisoinnin osilta ei sinänsä ollut mitään rajoituksia tai ohjeistusta suunnitteluun liittyen. Suunnitellessani etämonitorointia mielessä kävi kolme erilaista ratkaisutapaa. Kaksi tavoista liittyy pääalgoritmin vaiheistukseen ja yksi palvelupohjaiseen toteutukseen.
Vaiheistusmalliin perustuvista ratkaisuista ensimmäinen oli daemonin tilakoneistaminen lepo- ja työtiloihin perustuva ratkaisutapa. Hylkäsin menetelmän hyvin nopeasti sen toteuttamisen vaikeuden vuoksi. Daemonin tilakoneistamisessa ongelmallista on se, että daemon ja pluginit toimivat yhdessä ja ajavat monia säikeitä samanaikaisesti, tämän kaiken tilakoneistaminen olisi ollut hyvin työlästä sekä myös erittäin helposti ongelmatilanteisiin (bugit) johtava menetelmä.

Toinen vaiheistusmalli perustui muunneltuun MVC (Model-View-Controller) malliin, jossa palvelun palvelin olisi ohjannut keskitetysti objektitasolla asiakkaiden ohjelmiston objekteja verkkoyhteyden ylitse. Toteutus muistuttaisi etäisesti lähteen 13 tapaa hajauttaa Java -sovelluksien etäsuoritusta. Tämän mukainen toteutus olisi ollut toteutettavissa huomattavasti helpommin kuin ensimmäinen ratkaisuehdotus.

Ratkaisu ei kuitenkaan ole tässä tapauksessa järkevästi sovellettavissa, koska objektien ohjaaminen etäältä olisi vaatinut huomattavan paljon viestiliikennettä ohjauskäskyjen vuoksi, jolloin asiakaslaitteen verkkolaitetta olisi käytetty suhteellisen usein eikä tehokkaaseen virransäästöön päästäisi.

Viimeisenä ratkaisutapana oli perinteisen Client-Server –ajattelun mukainen ratkaisu, jossa asiakkaan mobiililaite ottaa asiakasohjelmistolla yhteyttä palvelun palvelinohjelmistoon. Toteuttaminen tällä tavalla näytti suhteellisen helpolta, joskin työläältä.

Lähtiessäni toteuttamaan ratkaisua Client-Server –ajattelun mukaan päätin toteuttaa

palvelun siten, että viestiliikenne on yhteyden muodostamisen jälkeen mahdollisimman paljon palvelulta asiakkaalle päin. Käytännössä tavoitteeseen viestiliikenteen osilta päästiinkin, kun asiakas on saanut yhteyden palveluun ja kättely (handshake) on valmis, kaikki tapahtuva verkkoliikenne aloitetaan aina palvelimelta asiakkaalle päin.

Liikenne on toteutettu niin, että se toimii myös varmennuksena sille, että asiakas on yhä kuuluvuusalueella. Asiakassovelluksen odotetaan kuittaavan jokainen viesti. Mikäli viesti ei mene perille, tai kuittauksen vastaanotto epäonnistuu, lisätään viestivirhelaskuria. Laskurin saavuttaessa tarpeeksi korkea arvo katsotaan asiakaslaitteen kadonneen verkon kuuluvuusalueelta.

3 KÄYTÄNNÖN OSUUS

Käytännön toteutus vaiheistui karkeasti noin viiteen eri vaiheeseen, mikäli vaiheistusta katsotaan esiintyneiden ongelmien perusteella. Työn ensimmäinen vaihe oli lähinnä PeerHood:iin tutustumista sekä C++ -kielen uudelleen lämmittelyä parametrisaattorin parissa. Suurin osa ensimmäisestä toteutusvaiheesta oli PeerHood:iin tutustumista ja sen tarjoamien metodien testausta muunnelluilla esimerkkisovelluksilla.

Ensimmäisen vaiheen jälkeen pääsin käytännössä toteuttamaan kandityön pääaihetta, eli verkkonaapuruston valvontatehtävien siirtoa. Toisen vaiheen aikana tutustuin lisää PeerHood –kirjaston toiminnallisuuksiin sekä toteutin ensimmäiset testisovellukset, jotka toimivat vain lokaaliyhteyksillä. Tässä vaiheessa työtä ohjelmistot hallitsivat yhteyspyynnön hyväksymisen, asiakkaan poistumisen palvelusta, sekä palvelun alasajoon tarvittavat toimet.

Toisen vaiheen aikana suunnittelin myös palvelimen algoritmisesta toiminnasta vedoksen, palvelinsovelluksen pääalgoritmi jakautui viiteen eri lohkoon, joista jokaisella on oma tehtävänsä. Toisen vaiheen aikana suunnittelemani palvelimen runko pysyi koko työn ajan melko samanlaisena.

Kolmas toteutusvaihe sisälsi lokaaliyhteyksillä toimivien ohjelmien muuntamisen toimimaan PeerHood:n verkkoyhteyksien kautta. Kolmannen toteutusvaiheen kulmakivi on palvelun kyky vastaanottaa seurantaa varten asiakkaalta laite- tai verkko-osoitteita, joiden tilanmuutoksiin ohjelmiston tulisi reagoida. Tässä vaiheessa palvelin- ja asiakasohjelmat sisälsivät tarvittavat metodit tilanmuutoksiin reagoimiseksi, mutta toiminnallisuus jäi puuttumaan.

Neljännessä toteutusvaiheessa ongelmien ratkomiseen meni suhteessa muuhun työhön pisin aika. Tämän vaiheen aikana toteutin asiakas- ja palveluohjelmistojen välille verkkonaapurustotiedon päivitysmenettelyn. Tämän vaiheen aikana kompuroin useampia kertoja epäyhteensopivien tietorakenteiden kanssa. Useamman viikon tutkimisen jälkeen

pystyin toteuttamaan luotettavan menetelmän naapurustotiedon päivittämiseksi. Tämän vaiheen aikana törmäsin myös Bluetoothin ensimmäisen version aiheuttamiin ongelmiin.

Työn viides vaihe toteutettiin lähdekoodin katselmoinnin jälkeen. Tässä vaiheessa toteutin ohjelmiston viimeiset puuttuvat ominaisuudet sekä kevensin ohjelmiston algoritmista toimintaa. Samalla muutin ohjelmiston toimintaa Bluetooth-ongelmien vuoksi. Muun muassa asiakkaiden antamien osoitteiden aktiivinen seuranta piti poistaa, koska tämä häiritsi Bluetooth:n omia laitteiden etsintämetodeja.

3.1 I –vaihe: Parametrisointi ja tutustuminen

Tutustumisvaiheessa luin läpi PeerHood:n API-spesifikaation sekä projektin Wiki –sivuston esimerkit ja funktiokuvaukset, kuitenkin tehtävänsiirron toteuttamiseksi tämä tieto ei ollut riittävää vaan jouduin tutustumaan PeerHood:n sisäiseen toimintaan lukemalla lähdekoodia. Samalla työstin työn sekundaarista osa-aluetta, eli parametrisointia.

Parametrisointiohjelman toteutin siten, että ohjelman käynnistyessä ohjelma testaa ensin laitteen suorituskykyä, jonka mukaan arvioidaan kuinka monta laskentakierrosta ohjelma ajaa. Laskentakierrosten määrän tulee olla tarpeeksi suuri, jotta käyttöjärjestelmän omilla ajanmittausfunktioilla saatavilla aikaeroilla voidaan laskea laitteen suhteellinen suorituskyky. Toisaalta laskentakierroksia tulee olla sen verran vähän, ettei suoritusaika kasva liian suureksi.

Tämän lisäksi jokaisella laskentakierroksella tulee laskea useampia laskutoimituksia kuin yksi. Laskentakierroksien aritmeettisen osuuden toteutin siten, että jokainen laskentakierros on ohjelmointikielen (C++) toistorakenteen mukainen kierros. Jokaisen kierroksen sisällä yritetään laskea 4 laskuoperaatiota: lisäys-, vähennys-, jako- ja kertolaskut. Ilman kääntäjän optimointeja tuloksena pitäisi olla ohjelmakoodia, jota nykyiset x86 –yhteensopivat suorittimet osaavat ajaa rinnakkain, mikäli muistiosoitteiden

välillä ei ole riippuvuuksia.

Koodin rinnakkaisuus aiheuttaa x86- ja ARM –suorittimien välille suuremman eron tehoindeksissä, mutta myös pidentää suoritusaikaa. Kuten teoriaosuudessa tuli viitattua lähteeseen 7 ja 8, useimmat ARM –suorittimet eivät pysty ajamaan useita laskuoperaatioita rinnakkaisesti samalla kellojaksolla. Olettaen, että keskiarvoisesti x86 –arkkitehtuurin prosessorit voivat suorittaa 2-3 operaatiota kierroksella, yksi laskentakierros valmistuu 2 kierroksella, kun ARM –suorittimilla tässä operaatiossa kestäisi noin 4 kellojaksoa.

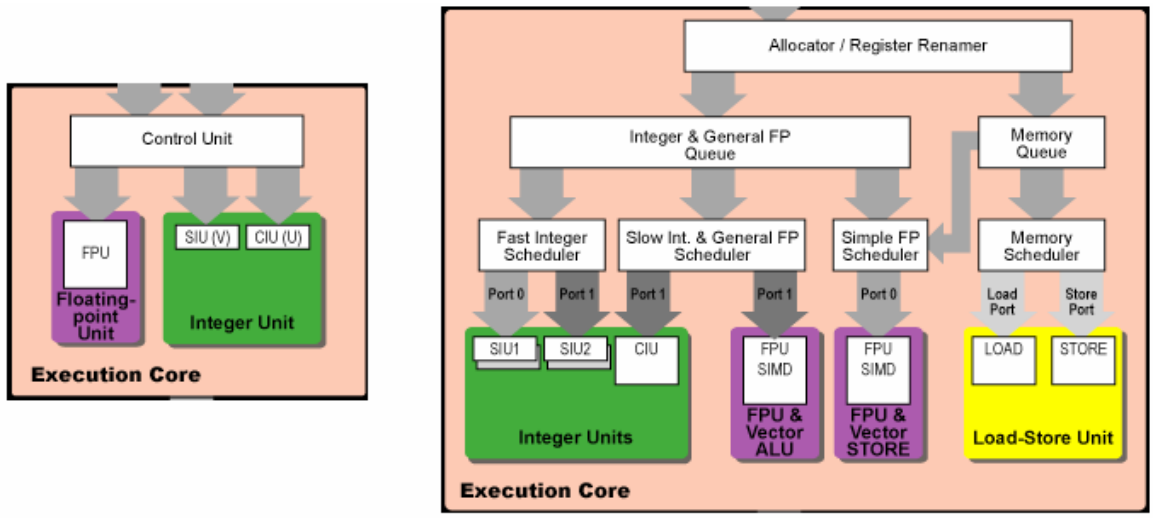
Jotta parametrisoinnin tuottamat arviot eivät olisi liian yksipuolisia, mittaa parametrisaattori laskentanopeutta 32 bitin kokonaisluvuilla, sekä yksinkertaisen että tuplatarkkuuden liukuluvuilla. Molemman tyyppisten liukulukujen käyttämisen perusteena on se, että osa x86 –arkkitehtuurin suorittimet poikkeavat malli- sekä valmistajakohtaisesti tavaltaan laskea näitä. Osa suorittimista pystyvät laskemaan täyden tarkkuuden liukulukuoperaatioita jokaisella kellojaksolla, kun toisilla suorittimilla tämä vaatii enemmän aikaa.

Koska mobiililaitteiden muisti on yleisesti ottaen huomattavasti hitaampaa kuin kiinteiden- tai kannettavien tietokoneiden muisti, määrittelin parametrisaattorissa laskentaan käytetyt muuttujat **volatile-**tyyppisiksi, jolloin kääntäjä pakottaa suorittimen lataamaan muuttujan arvon uudestaan muistista. Tällöin tehoindeksi on myös riippuvainen laitteen muistiväylän nopeudesta sekä latenssista.

Kaikkien kolmen muuttujatyypin laskentatehosta mitataan indeksi, oikea nopeusindeksi lasketaan näiden keskiarvosta, mutta muuttujatyyppikohtaiset indeksit säilytetään myös mahdollista myöhempää tarvetta varten.

Kuva 3 selventää x86 –arkkitehtuurin suorittimien sisäisiä eroja suoritusyksiköiden rinnakkaisuuden osilta. Kuva on toteutettu yhdistämällä kahdesta erillisestä kuvasta, kuvassa vasemmalla Pentium –ytimen suoritusyksiköt, oikealla Pentium 4 –ytimen

suoritusyksiköt.



Kuva 3. Pentium ja Pentium 4 -ytimien suoritusyksiköt. [9][10]

Perehtymättä enempää arkkitehtuurillisiin ratkaisuihin tai laskentayksiköiden keskinäisiin eroihin, nähdään kuvasta, että rinnakkaisten suoritusyksiköiden määrä voi vaihdella huomattavasti, vaikka suorittimet suorittavatkin saman käskykannan mukaista koodia.

3.2 II –vaihe: Palvelun suunnittelu ja esivedos

Palvelun esivedosta tehdessä yritin suunnitella ohjelmien käyttämät tietorakenteet siten, että niitä ei tarvitsisi enää myöhemmässä vaiheessa muutella. Muut tavoitteet esivedokselle oli onnistua suunnittelemaan verkkoprotokollat yhteydenottoa (Connect), yhteyden katkaisua (Disconnect) ja palvelun alasajoa (Abort) varten. Tämän lisäksi tavoitteena oli myös saada jonkin tasoinen esivedos palvelimen algoritmisesta toiminnasta sekä jaotella koodi omiin osioihinsa vastuualueiden mukaan.

3.2.1 Palvelinohjelman tietorakenteet

Palvelinohjelman asiakastietorakenne pitää sisällään asiakkaan yhteysobjektin (MAbstractconnection), STL –kirjaston mukaisen linkitetynlistan laiteosoitteille joita

halutaan valvoa, sekä totuusarvomuuttujan (Boolean) jonka arvosta voidaan päätellä tarvitseeko asiakas verkkonaapurustotiedon päivityksen. Lisäksi jokaisen asiakkaan tietorakenteessa on laskuri, johon lasketaan tietoliikenneyhteyksissä havaitut virheet.

Asiakkaan tietorakenne näyttää seuraavanlaiselta:

```
struct RmCustomer {
  MAbstractConnection* connection;  // osoitin yhteysobjektiin
  list<string> myDeviceList;        // seurattavien laitteiden osoitteet
  bool needUpdate;                  // päivitystarpeen flagi
  int CommunicationFailure;         // virhelaskuri
};
```

Lisäksi server –ohjelma tallentaa CallBack –rajapinnan kautta tulevat ilmoitusviestit seuraavanlaisiin rakenteisiin:

```
// Saapuvien asiakkaiden jono (osoittin yhteysobjektiin)
queue<MAbstractConnection*> incoming;

// Verkkoon ilmestyneiden laiteosoitteiden jono (osoite merkkijonona)
queue<string> joinedDevices;

// Verkosta poistuneiden laiteosoitteiden jono
queue<string> leftDevices;
```

Kaikki ohjelmiston jonot ovat loogisesti globaaleja muuttujia, jotta sekä CallBack –rajapinta että server –ohjelman pääalgoritmi pystyvät käsittelemään niitä suoraan.

3.2.2 Palvelinohjelman algoritminen toiminta

Algoritmisesti palvelinohjelma pyörii yhdessä pääsilmukassa, joka on jaettu viiteen lohkoon. Jokaisella lohkolla on omat tehtävänsä, jotkin tehtävistä muuttavat palvelimen tietojen tietorakenteita siten, että niiden täytyy keskeyttää suorituksensa ensimmäisen tehtävän suoritettua.
Esimerkkinä mainittakoon pääsilmukan toinen lohko, joka reagoi palvelusta poistuviin asiakkaihin. Lohko poistaa kaikki poistuvan asiakkaan datat sekä vapauttaa niille varatut muistialueet. Koska listasta poistamisen jälkeen STL –listan tarjoamat iteraattorit korruptoituvat, täytyy tehtävä keskeyttää ensimmäisen poiston jälkeen. Tämän vuoksi

myös osa lohkoista käynnistää pääsilmukan uudelleen.

Palvelimen pääsilmukan lohkot ja niiden tehtävät suoritusjärjestyksessä pseudoalgoritmina.

PÄÄSILMUKKA (toista, kunnes palvelin suljetaan)

1) WHILE(jono ”incoming” ei ole tyhjä): → toista kunnes jono on tyhjä
   - vastaanota uusi asiakas
2) JOS jokin asiakkaista lähettänyt viestin, tarkista onko viesti ”DISCONNECT” →
   - TOSI: Kuittaa ”OK” ja poista asiakas
   - EPÄTOSI: Poista viesti. (asiakkaan ainoa käsky palvelimelle päin)
3) WHILE(jono ”leftDevices” ei ole tyhjä): → toista kunnes jono on tyhjä
   - Käy läpi asiakkaat, vertaa jokaisen asiakkaan seurattavien osoitteiden listaa saatuun osoitteeseen. JOS osoite täsmää: → merkitse asiakkaan ”needUpdate” = TOSI
   - Kun osoitetta verrattu kaikkiin asiakkaisiin, poista osoite jonon kärjestä. Toistetaan kunnes jono tyhjä.
4) WHILE (jono ”joinedDevices” ei ole tyhjä): → toista kunnes jono on tyhjä
5) JOS verkossa riittävästi muutoksia, tai ajastin ylittynyt:
   → lähetä kaikille päivitys
   - JOS OK: muuta needUpdate epätodeksi, nollaa viestintävirhelaskuri.
   - JOS VIRHE: Lisää viestintävirhelaskuria

   MUUTEN: → Lähetä päivitys kaikille joilla ”needUpdate” on tosi.
   - JOS OK: muuta needUpdate epätodeksi, nollaa viestintävirhelaskuri.
   - JOS VIRHE: Lisää viestintävirhelaskuria

Huomautuksena mainittakoon, että viestintävirhelaskurien tarkastusta en suunnitellut vielä tässä vaiheessa. Toisen toteutusvaiheen aikana toteutin palvelinohjelmien lohkot 1 ja 2, sekä niihin liittyvän viestiliikenteen protokollat. Lohkojen 3 – 5 toiminnallisuuksia en toteuttanut luonnostelua pidemmälle tässä vaiheessa.

3.2.3 Palvelinohjelman viestityypit

Päätin lähteä tekemään esivedoksen tietoliikennettä lokaaliyhteyksien avulla, koska en vielä ymmärtänyt kunnolla PeerHood:n toimintaa ja halusin minimoida mahdollisia virhetilanteita tai ongelmia. ”rMon” –palvelun esivedokseen toteutin tarvittavat toiminnot ohjelmiston client- ja server –osioihin yhteyden muodostamiselle, sekä seurattavien laitteiden osoitteiden vastaanotolle ja käsittelylle yhteys muodostettaessa.

Yhteydenottoprotokollan lisäksi kehitin molempiin ohjelmiin viestiprotokollan palvelusta

eroamiseen (Disconnect) sekä palvelun alasajoa varten (Abort). Varsinkin abort –viestityyppi on tärkeä, koska jos palvelu ajetaan alas jostain syystä, täytyy client-sovelluksen pystyä käynnistämään oman daemoninsa verkkoinformaationylläpito.

Viestikohtaisten MSC (Message Sequence Chart) ollessa suhteellisen suuria ja vaikeita keskittää tekstiin, löytyvät Connect, Disconect ja Abort viestien MSC –kuvat työn lopusta liitteinä. Huomautus Connect –tapahtuman MSC –kaaviossa yhteydenmuodostuksen jälkeen lähetettävä lukumääräparametri voi olla 0, jolloin palvelin kuittaa suoraan ”OK” –viestillä. Mikäli lukumääräparametri on suurempi kuin 0, odottaa palvelin saavansa lukumäärän osoittaman määrän ylimääräisiä datapaketteja.

3.3 III –vaihe: Verkkoyhteyksien käyttöönotto.

Toisen vaiheen ohjelmoinnin aikana tutustuin mielestäni riittävästi PeerHood:n tarjoamiin toiminnallisuuksiin, mutta en vielä tiennyt miten saisin kaikki vaaditut ominaisuudet toteutettua. Päätin kuitenkin lopettaa turhauttavan pohtimisen ja ruveta toteuttamaan toisen vaiheen aikana tehdystä ohjelmistosta versiota joka osaisi tehdä samat asiat verkkolaitteiden kautta.

Client- ja Server –ohjelmien muuttaminen toimimaan verkkolaitteiden avulla oli loppujen lopuksi hyvin triviaalia, suurimmat muutoksen ohjelmissa tapahtui client –ohjelman puolella. Lokaaliclientin käyttämä funktio, jolla etsitään laitetietokannasta oman laitteen paikallisia palveluita, ei ollut yhteensopiva funktion kanssa jolla etsitään verkon laitteita.

Pienien muutostöiden jälkeen pystyin kokeilemaan Connect –protokollaa Bluetooth-yhteyden avulla, en voinut kuitenkaan alkuun testata seurattavien laitteiden listan lähetystä yhteydenotossa, koska käytössäni ei ollut tarpeeksi Bluetooth-laitteita. Suoritin kuitenkin havainnollistavan testin rakentamalla kuvitteellisen osoitelistan käsin, listan lähetys ja vastaanotto onnistui, joten oletin sen toimivan myös oikeilla laiteosoitteilla.

Server –sovellukseen tein tarvittavat toiminnallisuudet verkon muutoksiin reagoimiseksi pääsilmukan lohkoihin 3 ja 4 (verkkoon liittyneet ja verkosta kadonneet laitteet). Lohkojen sisäisten silmukoiden lisäksi tein apufunktiot joiden avulla pystyttiin vertaamaan 2. lohkon disconnect-tapahtumassa asiakkaan seurattavien laitteiden listaa muihin asiakkaihin. Sellaiset osoitteet joita ei löytynyt kuin poistuvalta asiakkaalta oli tarkoitus poistaa myös valvottavien osoitteiden valvontasilmukasta. (Valvontasilmukka on PeerHood –kirjaston tarjoama metodi ”MonitorDevice()”.)

Koska tuolla hetkellä en saanut mistään lisää Bluetooth-laitteita käyttööni, päätin ruveta suunnittelemaan kuinka verkkonaapuruston tiedot pystyttäisiin siirtämään laitteelta toiselle. Tässä vaiheessa suunnittelin tekeväni päivityksen siten, että palvelin käyttäisi PeerHood:n omia sisäisiä funktioita laitelistan serialisointiin ja lähettäisi listan asiakkaille, jotka pystyivät kokoamaan tietokannan uudestaan.

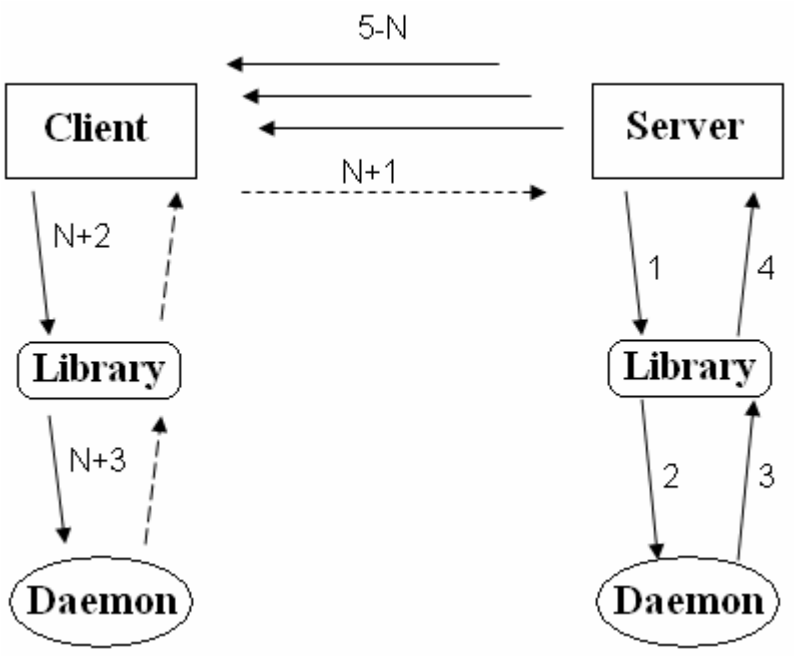
Toteutin päivitystoimintoa varten komentoviestin ”UPDATE” sekä myös tarvittavan metodin laitelistan serialisoitujen objektiesitysten kasaamiseksi ja lähettämiseksi. Pystyin lähettämään listan server –ohjelmalta client –ohjelmalle, mutta päivitys client –ohjelmalta sen oman laitteen daemonille tuotti ongelmia.

Kuvasta 4 ilmenee suunniteltu viestiliikenne, seuraavasta listasta ilmenee numeroitujen viestien tarkoitus.

1. Server –sovellus pyytää kirjastolta laitetietokannan.
2. Kirjasto pyytää laitetietokantaa daemonilta.
3. Daemon lähettää N kappaletta serialisoituja laiteobjekteja.
4. Kirjasto kasaa omaan tietorakenteeseen laitelistan ja välittää osoittimen listasta palvelimelle.
5. Palvelin lähettää ”UPDATE” viestin. 6. viesti sisältää lähetettävien laitteiden määrän. Viestit 7 – N ovat serialisoituja laiteobjekteja.

N+1. Asiakasohjelma kuittaa ”OK” viestillä.
N+2. Asiakasohjelma kutsuu päivitysfunktiota ja antaa syötteeksi saadut objektit. Kirjasto kokoaa uuden laitetietokannan serialisoiduista objekteista.
N+3. Kirjasto lähettää laitetietokannan daemonille.

Vaiheen N+3 kohdalla ongelmaksi muodostuu se, että kirjastotason laiteobjektit ovat epäyhteensopivia daemon-tason laiteobjektien kanssa.



Kuva 4. Päivitystoiminnon suunniteltu viestiliikenne.

Lyhyen virheidenmäärityksen jälkeen huomasin oman virheen suunnittelussa; en ollut huomannut, että vaikka daemonin laitelista voidaan selialisoida ja kasata hieman erityyppiseksi listaksi kirjastolle, tätä toimenpidettä ei kuitenkaan voi tehdä automaattisesti toiseen suuntaan. Syy tähän oli tietorakenteiden ja objektien pienet eroavaisuudet. Listan olisi voinut tulkata takaisin daemonin ymmärtämään muotoon, mutta tällöin ei voida olla varmoja välittyykö kaikki tarpeellinen informaatio, joten päätin etsiä toisenlaista ratkaisua.

3.4 IV –vaihe: Päivitystoiminnallisuus

Verkkonaapurustotiedon päivitysmenetelmän ensimmäisen version epäonnistuttua päätin etsiä toisenlaisen menetelmän. Tavoitteena oli kehittää päivitystoiminto daemonin kautta käyttäen daemonin tietorakenteen mukaista laitetietokantaa ja tietokannan siirron tuli tapahtua daemonin verkkonaapuruston päivitysmetodien avulla. Tällöin laitetietokantaa ei tarvitsisi muuntaa eri muotoon ja sille olisi valmiit serialisointimetodit.

Tutkittuani perusteellisesti daemonin menetelmän verkkonaapurustotiedon päivittämiseen, päätin toteuttaa oman etäpäivitykseni muokkaamalla jo valmiita menetelmiä. Normaali päivitysmenetelmä pyytää uuden laitteen löytäessään sen daemonia lähettämään joko omat tietonsa tai koko laitetietokannan riippuen PeerHood:n asetustiedoista.

3.4.1 PeerHood:n naapurustoinformaation lähetys

Käytännössä päivitys toimii seuraavasti laitteiden A ja B välillä: Laitteen A Inquiry-säikeen löytäessä uuden laitteen B, laite A lähettää päivityspyynnön ja PeerHood:n asetustiedoston mukaisen päivitystasolipun laitten B daemonille. Laitteen B daemonin saman verkkoteknologian pluginin Advert-säie vastaanottaa pyynnön ja lähettää pyydetyt tiedot laitteelle A, joka on käynnistänyt pyyntöään vastaavan vastaanotto metodin.

Päätin puukottaa menetelmää siten, että huolimatta PeerHood:n asetustiedoston määrittelemästä päivitystasosta, lähettäisi daemon pyynnön koko verkkonaapurustotietokannasta. Daemonin tulisi myös kertoa ohjelmalle paluuarvolla päivityksen onnistumisesta. Koska PeerHood –ohjelmalla ei ole suoraa yhteyttä daemoniin, täytyi sekä funktioiden toiminnan aloittava data sekä niiden paluuarvot siirtää lokaalien käyttöjärjestelmätason socket –yhteyksien kautta.

Menetelmästä oli kuitenkin muutamia ongelmia, verkkonaapuruston päivitys toimii hakutyyppisesti, eli päivityksen haluavan laitteen täytyy lähettää pyyntö, kuitenkin ”rMon” –palvelussa tarkoitus on hoitaa tiedon päivitys silloin kun palvelin haluaa päivittää asiakkaiden tietoja. Toisekseen mobiililaitteiden daemonien Inquiry-säiettä ei voida käyttää suoraan menetelmässä, koska tämä säie täytyy pysäyttää IDLE –tilassa olevan laitteen daemonista.

3.4.2 ”rMon” –palvelun verkkonaapurustoinformaation päivitys

Verkkonaapurustotiedon päivitysmenettelyn toteutin lopullisesti siten, että ”rMon” –palvelin lähettää UPDATE-komennon jokaiselle asiakkaalle yksitellen, ”rMon” –client sovellukset lähettävät PeerHood –kirjastoon toteutetun funktion avulla palvelin koneeseen käytetyn verkkoyhteyden prototyypin ja verkkolaitteen osoitteen omalle daemonille.

Daemonin vastaanotettua verkkolaitteen tyypin ja osoitteen, se etsii ladatuista plugineista vastaavan pluginin ja lähettää päivityspyynnön sekä päivitystasolipun palvelun daemonille, jonka Advert–säie reagoi tähän lähettämällä laitetietokannan samaa yhteyttä pitkin takaisin. Kun kyseessä oleva verkkoplugin on vastaanottanut laitetietokannan se koettaa päivittää oman daemonin tietokantaa ja päivitysfunktion paluuarvo palautetaan daemonille. Riippuen lipun arvosta daemon vastaa kirjastolle lokaalisoketin lävitse paluuarvon operaatiosta joka välitetään edelleen ohjelmalle.

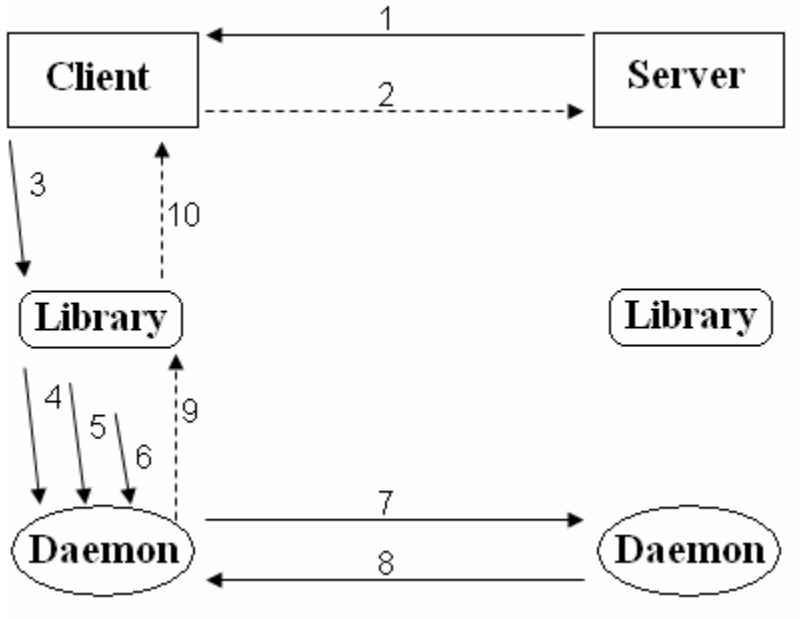
Palvelua ei ole suunniteltu käytettäväksi GPRS –yhteyksillä, joten GPRS –verkkopluginin vastaava päivitysmenetelmä palauttaa aina automaattisesti virheen, eikä edes yritä päivittää tietoja. Koska PeerHood abstraktoi kaikki verkkopluginit, täytyy jokaisesta pluginista kuitenkin löytyä kaikki Interface–luokan määrittelemät funktiot.

Tämän vuoksi PeerHood –kirjasto lähettää päivityspyynnön daemonille myös GPRS –yhteyden tapauksessa, mutta daemonin pyytäessä verkkopluginia tekemään päivityksen,

plugin palauttaa automaattisesti FALSE –arvon. Kirjastofunktion dokumentaatioon on merkitty huomautus ohjelmoijia varten, että sitä ei ole tarkoitus käyttää GPRS -yhteyksillä

Kuvasta 5 ilmenee päivitysmenettelyn viestiliikenne asiakas- ja palvelinohjelmien välillä. Numeroitujen viestien sisältö ja merkitys:

1. Palvelin lähettää UPDATE –komentoviestin asiakkaalle.
2. Asiakasohjelma kuittaa ”OK” viestillä. (Ping)
3. Asiakasohjelma kutsuu kirjaston funktiota ja syöttää parametreiksi palvelimen osoitteen ja verkkopluginin prototyypin.
4. Kirjasto lähettää lokaalille daemonille enumeroidun PDU –käskyn (PH_REQUEST_REMOTE_DEVICE_LIST)
5. Kirjasto lähettää daemonille laitteen verkko-osoitteen
6. Kirjasto lähettää daemonille laitteen verkkolaitteen prototyypin
7. Daemon lähettää palvelun daemonin Advert–säikeelle päivityspyynnön ja lipun. (D_GET_NEIGHINFO)
8. Palvelun daemon lähettää laitetietokantansa asiakasohjelman daemonille.
9. Asiakkaan daemon vastaa PeerHood –kirjastolle enumeroidulla PDU –käskyllä PK_OK jos päivitys onnisttuu, päivityksen epäonnistuessa vastataan käskyn PDU:lla (PH_REQUEST_REMOTE_DEVICE_LIST)
10. Kirjaston funktio palauttaa TRUE tai FALSE paluuarvon ohjelmalle.



Kuva 5. Naapurustontiedon päivitysoperaation viestiliikenne.

Huomattavaa viestimenettelyssä on, että asiakkaan täytyy kuitata ensimmäinen komentoviesti, jotta palvelin tietää asiakkaan saaneen viestin. Viestiliikenteessä ei kuitenkaan tarkisteta onnistuuko daemonien välinen päivitystoiminto, päivityksen epäonnistuessa uudelleen yrittäminen jää asiakasohjelmiston vastuulle.

Päivitetyn ”UPDATE” -viestiketjun MSC –kuva löytyy liitteenä dokumentin lopusta.

3.4.3 Ohjelmien testaus ja Bluetooth –ongelma

Neljännen toteutusvaiheen aikana sain käyttööni lisää Bluetooth –usb-sovittimia ja pystyin ajamaan ohjelmieni välillä oikeita testejä. Testejä ajaessa ohjelmista löytyi muutamia pieniä virheitä, jotka olivat nopeasti korjattavissa. Suurempia ongelmia aiheutti Bluetooth –protokollan ensimmäisen version verkkolaitteiden hakutoiminnon (HCI_INQUIRY) epäonnistuminen mikäli samaa laitetta monitoroitiin aktiivisesti, tai se monitoroi jotain toista laitetta.

PeerHood:n aktiivinen monitorointi on lähinnä ”PING-PONG” –viestitilanne, jossa valvottavaa laitetta pingataan tietyin aikavälein. Aktiivisen PING –liikenteen aikana Bluetooth protokollan naapureidenhakumetodit epäonnistuvat satunnaisesti. Ongelma vaivasi kaikkia Bluetooth-laitteita, mutta pahiten ongelmaa esiintyi 3COM –merkkisten Bluetooth-sovittimien kanssa.

Tästä johtuen palvelinohjelman toimintaa oli muutettava siten, että asiakkaiden antamia valvottavia osoitteita ei enää valvottu aktiivisesti PING –viesteillä. Lohkon 2 (disconnect) sisältä kommentoin koodista pois osiot jotka tarkistivat aktiivisesti valvottavien laitteiden listaa sekä poistivat sieltä laitteet joita kukaan ei halunnut valvoa.

Jotta asiakkaat saisivat valvottavista laitteista ilmoitukset, lohkot 3 ja 4 muutin vertaamaan ilmoitusviestinä (CallBack) saatavaa osoitetta asiakkaiden omiin listoihin, mikäli asiakkailla on sama osoite omassa listassaan, muutetaan asiakkaan ”needUpdate” –lipun arvo arvoon ”TRUE”.

Aktiivisen valvonnan aiheuttamien ongelmien korjaamisen jälkeen ohjelmisto tuntui toimivan vakaasti. Myös päivitysmenettely toimi testeissä, huomattavaa oli, että etäpäivitys on ainakin Bluetooth –protokollaa käytettäessä huomattavasti nopeampi kuin protokollan oma laitteidenetsintämenetelmä, eli verkkolaitetta ainakin teoriassa kuormitetaan lyhyemmän ajan.

3.5 V –vaihe: Daemonin tilanmuutokset ja viimeistely

Viides toteutusvaihe alkoi neljännen vaiheen ohjelmiston katselmoinnin jälkeen. Katselmoinnissa ei kuitenkaan etsitty lähdekoodista mahdollisesti löytyviä pieniä virheitä vaan käytiin läpi kuinka client- ja server –ohjelmat toimivat. Katselmoinnin tuloksena tein ohjelmiin pieniä muutoksia ja korjauksia sekä lähdin toteuttamaan daemonin lepotilan käyttöä vaativia metodeja.

Pääasioina katselmoinnissa käytiin lävitse päivitysmenettely sekä päivitysperusteet. Katselmoinnissa todettiin, että yleiset päivitykset kaikille asiakkaille hoidettaisiin ajastimen ja laskurin avulla, koska koko laitetietokannan seuraaminen ja muutoksien vertailu veisi suhteellisen paljon suoritinaikaa palvelimella. Yleisen päivityksen ehdoiksi laitettiin ajastin, joka pakottaa päivityksen tietyin aikavälein. Jos verkontilassa tapahtuu muutoksia nopeasti, ylittyy laskurin raja-arvo ennen ajastinta, tällöin lähetetään päivitys kaikille asiakkaille.

Katselmoinnissa myös hyväksyttiin asiakkaan valvottavien laitteiden seurannan muuntaminen aktiivisesta valvonnasta (ping), reagoimaan ilmoitusviestimenettelyn (CallBack) ilmoituksiin verkosta poistuneisiin tai verkkoon liittyneisiin laitteisiin. Päätin kuitenkin säilyttää vanhat menetelmät alkuperäisen aktiivisen valvonnan toteuttamiseksi, joten kommentoin lähdekoodista osioita pois.

Kommentoidut osiot on mahdollista palauttaa käyttöön, mikäli ominaisuus halutaan takaisin, kun Bluetooth-ajureista julkaistaan uudempi versio. Uudemman Bluetooth-

version pitäisi korjata HCI_INQUIRY –ongelma, koska Bluetooth:in 2. versio käyttää tietoliikenteelle eri taajuutta kuin laitteiden haulle.

Itse daemonin lepotilan toteutus ei ollut kovin hankalaa, daemonin verkkonaapurustotiedon päivitys tapahtuu plugineiden kautta toimivien säikeiden avulla. Päivitys on jaettu kahden säikeen vastuulle, joista Inquiry-säie etsii laitteita ja uuden laitteen löytyessä kysyy laitteelta tietoja. Kyselyyn vastaa verkkopluginissa ajettava Advert –säie. Molemmille säikeille löytyy jokaisesta pluginista start ja stop-kontrollimetodit.

Koska neljännessä toteutusvaiheessa toteutettu ”UPDATE” –rutiini pystyy päivittämään daemonin laitetietokannan ilman Inquiry –säiettä, voidaan client –sovelluksen puolella pluginien Inquiry –säie pysäyttää. GPRS –plugin käyttää hieman erilaista tapaa päivittää laitetietokantansa, koska sen täytyy ottaa yhteyttä GPRS –välityspalvelimeen jolta tiedot saadaan. Koska ”rMon” –palvelua ei ole tarkoitettu käytettäväksi GPRS –yhteyksien kanssa, sammutetaan myös GPRS –pluginin Inquiry –säie asiaa enempää tutkimatta.

Itse lepotoiminto toimii niin, että ”rMon” –client sovellus kutsuu kirjaston funktiota joka lähettää daemonille viestin samaan tapaan kuin päivitystapauksessa. Daemonin käsiteltyä viestin, se yrittää pysäyttää jokaisen verkkopluginin Inquiry –säikeen ja vastaa kirjastolle lokaalisoketin kautta PH_OK jos pysäytys onnistuu, tai alkuperäisellä käskyn PDU:lla mikäli toiminto epäonnistuu. Lepotilasta palautus tehdään samalla tavalla.

Itse client –sovelluksessa lepotilaan siirto tehdään onnistuneen yhteydenoton jälkeen. Client –sovellus ottaa yhteyden palveluun ja lähettää mahdollisesti listan seurattavista osoitteista, mikäli palvelin kuittaa yhteydenoton ”OK” viestillä, asiakasohjelma siirtää oman daemoninsa lepotilaan ja siirtyy odottamaan palvelin sovelluksen ohjausviestejä.

Mikäli palvelin ei suostu vastaanottamaan asiakasta (”ABORT” –viesti) tai asiakasohjelma ei pysty siirtämään daemonia lepotilaan, keskeytetään ohjelman suoritus ja tarvittaessa palautetaan daemon työtilaan.

3.7 Tulosten yhteenveto

Lyhyt tuloksien yhteenveto samassa järjestyksessä kuin muukin dokumentaatio, ensin käydään lyhyesti lävitse parametrisaattori ja sen jälkeen itse tehtävänsiirto-ongelma.

3.7.1 Tuloksien yhteenveto – parametrisaattori

Kuten edellä mainittu, yritin toteuttaa parametrisaattorin niin, että siitä olisi mahdollista nähdä suoritinarkkitehtuurien aiheuttamat erot. Parametrisaattorin oli myös tarkoitus kuormittaa suorittimia siten, että suorittimen kellotaajuus ei vaikuta silmukoiden suoritusnopeuteen yksinään. Silmukoihin on sisällytetty riittävästi laskutoimituksia, jotta suoritin ei yleensä pysty suorittamaan silmukkaa yhdellä kierroksella.

Taulukosta 1 nähdään eri suorittimien suoritusnopeudet. Indeksit ovat laskukierrosten määrä sekuntia kohden. Jokainen laskukierros sisällyttää 4 lasku- ja 4 sijoitusoperaatiota, sekä silmukkalaskurin lisäys ja sijoitus operaation. Eli yhdellä laskukierroksella suoritetaan 5 LOAD- , 5 laskenta- ja 5 SAVE –operaatiota.

Suoritin	**Kellotaajuus**	**Int32**	**Float**	**Double**	**Keskiarvo**
Celeron	433 Mhz	5.95k	6.64k	6.68k	6.42k
Pentium III	600 Mhz	8.60k	8.51k	8.53k	8.54k
Via Epia C3	800 Mhz	7.45k	3.5k	3.51k	4.48k
Barton 2500+ (K7)	1.83 Ghz	23.1k	25.3k	25.4k	24.6k
(Dothan) Pentium-M	1.60 Ghz	29.5k	22.5k	22.4k	24.8k
Core 2 Duo e6300	2.13 Ghz	39.6k	28.5k	28.4k	32.16k
Athlon 64 3500+ (s754)	2.40 Ghz	32.5k	31.6k	34.1k	32.73k
ARM Suoritin	**Kellotaajuus**	**Int32**	**Float**	**Double**	
Nokia 770 tablet	252 Mhz	3.59k	0.72k	0.45k	1.57k

Taulukko 1. Suorittimien tehomittauksien tulokset.

Ilman tarkempia muistin viivemittauksia (Latency) voidaan todeta, että normaalien tietokoneiden osilta muistin nopeudella ei ole testissä merkitystä. Testatuista laitteistoista löytyy 2 laitteistoparia jotka ovat laskentateholtaan hyvin samalla viivalla, mutta niissä

on toisistaan huomattavasti eroavat muistiteknologiat.

Ensimmäisenä kokonpanoparina ovat Intel Celeron (Pentium2-ydin) varustettuna normaalilla SDR (Single Data Rate) muistilla. Celeron:ia vastassa oleva Pentium III kokoonpano on varustettu nopealla Rambus:n kehittämällä RD-RAMM () muistilla, joka on karkeasti noin neljä kertaa nopeampaa ja omaa huomattavasti pienemmät hakuviiveet kuin SDR. Kuitenkin suorittimien nopeuserot ovat samaa luokkaa kellotaajuudet huomioiden, vaikkakin suoritinytimien arkkitehtuurissa on pieniä eroja.

Toisena melko tasaväkisenä kokoonpanoparina ovat Intelin Core 2 Duo e6300 ja Amd Athlon 64 3500+. Intel –konepaketti hyödyntää uudempaa DDR2 muistiteknologiaa Amd:n vanhemman koneen käyttäessä DDR1:stä. Vaikka Intel-kokonpanossa on nopeammat muistit, suoriutuvat suorittimet nopeustestistä samalla keskiarvolla. Näiden koneiden testituloksista ilmenee selvästi myös, että parametrisaatorin avulla erotetaan selvästi myös suorittimien laskuteho erityyppisien muuttujien suhteen.

Mittaustuloksista nähdään selvästi suorittimien nopeusindeksinä, päätin kuitenkin säilyttää myös erillisten muuttujien suhteen lasketut nopeusindeksit, koska selvästi eri suorittimet suoriutuvat eri muuttujatyyppien laskennasta eri nopeudella.

Tällä hetkellä tuloksista mobiililaitteiden erottaminen on helppoa ja tulevaisuudessa tilanne tuskin muuttuu. Testissä käytetty Nokian 770 tablet käyttää suhteellisen uutta ARM9 –arkkiteehtuurin suoritinta, laite suoriutuu nopeasti kokonaislukujen laskennasta, mutta laitteen heikko liukulukulaskentateho tiputtaa rankasti myös keskiarvoa.

Vaikka parametrisaattorilla ei saatu testeissä eroja eri muistiteknologioiden välille, suoriutuu parametrisaattori tehtävästään riittävän hyvin. Suorittimien mitatuista nopeusindekseistä pystytään erottamaan selvästi mobiililaitteet ja tietokonelaitteet toisistaan.

3.7.2 Tuloksien yhteenveto – tehtävänsiirto

Verkkonaapuruston ylläpitotehtävien siirto mobiililaitteelta kiinteälle laitteelle toteutui lopulta melkoisen hyvin, naapurustotiedon päivitystoiminnot onnistuvat ja etäpäivityksiä saava laite näkee verkossa samat laitteet kuin palvelua ylläpitävä tietokone. ”rMon” –palvelun verkkoprotokolla on pääsääntöisesti palvelulta asiakkaalle päin koko palvelun käytön ajan, joten palvelun tulisi myös säästää mobiililaitteen akkutehoa.

Varsinaista akkutehomittausta ei ole keritty suorittaa, kuitenkin Bluetooth –protokollaa käytettäessä 6 laitteen verkkonaapurusto informaatio siirtyy nopeammin palvelun päivitysmenettelyn kautta, kuin mitä Bluetoothin oma Inquiry –metodi vie aikaa. Seurattaessa liikenteen määrää hcidump –ohjelmalla nähdään, että verkkolaite on hyvin aktiivinen Inquiry –metodin ajon aikana.

Bluetooth –protokollan käyttämä oma laitteiden hakumenetelmä tuntuu toimivan hieman epävakaasti muutenkin, usein laitteita etsiessä se ei löydä jokaista kantomatkalla olevaa laitetta. Joskus jokin jo havaittu laite jättää vastaamatta Inquiry-kyselyyn eikä sitä löydetä, välillä taas verkosta löytyy yllättäen uusi Bluetooth-laite, vaikka laite on ollut päällä pitkän aikaa ja sen olisi pitänyt löytyä jo aikaisemmin.

Joka tapauksessa ”rMon” –palveluun toteutettu päivitysrutiini suoriutuu jo pienen verkkonaapurustotiedon kohdalla päivityksestä nopeammin kuin tavallinen Bluetooth-laitteiden etsintä. Koska ”rMon” –palvelu toteuttaa päivitykset harvemmin ja nopeammin kuin normaali PeerHood:n Inquiry, pitäisi palvelun käytön säästää laitteen akkutehoa ainakin Bluetooth-yhteydellä.

WLAN –yhteyksien tapauksessa naapurustotiedon siirrossa ja PeerHood:in omassa laitteiden etsintämenetelmässä ei ole nopeudellisesti eroa pienellä testiverkolla. WLAN –yhteyksien tapauksessa tulisi kuitenkin akkutehoa säästyä myös. Normaalissa toimintatilassa PeerHood etsii uusia laitteita verkosta 10 – 30 sekunnin välein (riippuen asetuksista). ”rMon” –palvelu päivittää asiakaslaitteiden naapurustoinformaatiota hieman

pidemmällä ajastimella mikäli verkossa ei tapahdu paljon muutoksia.

”rMon” –palvelu tuntuisi olevan yhteensopiva muidenkin PeerHood –ohjelmien kanssa. Vaikka palvelu kytkee asiakaslaitteen daemon prosessin lepotilaan, toimivat muut ohjelmat samalla tavalla. Ohjelmien käyttäessä laitetietokantaa ei ohjelmien toiminnassa ilmenny eroa, vaikka laitetietokanta ei ollutkaan daemonin itse kokoama vaan toiselta daemonilta ladattu versio.

Ohjelmat saavat otettua yhteyttä laitetietokannasta löytyviin palveluihin (jos ne ovat kuuluvuusalueella), myös muut PeerHood –laitteet saavat yhteyttä palvelua käyttävän tietokoneen omiin palveluihin. Asiakassovelluksen poistuessa palvelusta se onnistuu uudelleen käynnistämään oman daemoninsa Inquiry -rutiinit ja laitteen toiminta palautuu normaaliksi.

Homma toimii muuten moitteetta, mutta jos laitetietokannassa on laitteita, jotka eivät ole asiakaslaitteen kuuluvuusalueella, niiden poistuminen laitetietokannasta kestää tovin. Ongelma johtuu siitä, että laitteet voivat olla palvelun kuuluvuusalueella, mutta eivät toistensa kuuluvuusalueella. Tällöin palvelusta poistuttaessa asiakaslaitteen oma laitetietokanta sisältää katvealueella olevan laitteen, PeerHood poistaa verkosta kadonneen laitteen, jos sitä ei löydetä uudestaan 3 yrityksellä, jolloin laite roikkuu tietokannassa haamuna keskimäärin 1,5 minuuttia.

4 JOHTOPÄÄTÖKSET

Työssä toteutetun parametrisaattorin voidaan katsoa täyttävän esitetyt vaatimukset, eli sen tuloksista voidaan suoraan nähdä onko verkossa näkyvä laite mobiililaite, tietokone tai kannettava tietokone. Parametrisaattorin tulisi toimia myös tulevilla laitteilla, koska se yrittää automaattisesti tasapainottaa laskentatehtävän raskauden käytettävissä olevaan laitteistoon.

Parametrisaattorin tasapainotus ei toimi aivan täydellisesti, vaan hitaammilla mobiililaitteilla testiajon kesto venyy noin 30-40 sekunttiin, eritehoisilla tietokonelaitteilla testiajat pysyivät säännöllisesti 10 ja 20 sekunnin välillä. Parametrisaattorin avulla voidaan myös erottaa eri suorittimien tehoerot riippuen käytetyistä muuttajatyypeistä, joten vain yhdellä osa-alueella vahvoilla olevat suorittimet saavat heikommat pisteet kuin tasaiseen suorituskykyyn pystyvät laitteet.

Verkkonaapuruston valvontatehtävien siirron toteutus onnistui pääpiirteissään hyvin, palvelu toteuttaa työssä annetut vaatimukset, mutta kaikkia suunniteltuja ominaisuuksia ei voida hyödyntää ainakaan vielä. Muun muassa aktiivinen laitteiden valvonta (ping) jouduttiin kytkemään pois käytöstä Bluetoothin ensimmäisen version ongelmien vuoksi. Oletettavasti Bluetooth-laitteiden seuraavan sukupolven ei pitäisi kärsiä samoista ongelmista, koska laitteiden etsintään ei käytetä samaa tiedonsiirtokanavaa kuin datan siirtoon.

”rMon” –palvelun toteutus kuitenkin täyttää vaatimukset, ja palveluun liittyvä asiakaslaite voi kytkeä oman daemon-prosessinsa lepotilaan. Laite pystyy käyttämään muiden laitteiden PeerHood –palveluita samalla tavalla kuin aktiivisessakin tilassa, laite voi myös tarjota palveluita muille laitteille. Palvelusta eroamisen jälkeen asiakaslaite siirtää daemon-prosessinsa aktiivitilaan.

Aktiivitilaan siirryttäessä on tosin mahdollista, että asiakaslaitteella on käytössään virheellinen laitetietokanta, osa laitetietokannan laitteista ei välttämättä ole

kuuluvuusalueella, koska laitetietokanta on näkymä PeerHood –verkkoon palvelimen verkkolaitteen kuuluvuusalueelta. Laitetietokanta kuitenkin korjaantuu automaattisesti noin minuutin sisällä, joten en tehnyt daemoniin omaa laitetietokantaa tyhjentävää toimintoa.

Varsinaisia akkutehon säästön tehomittauksia ei ole suoritettu. Kuitenkin melkoisella varmuudella voidaan sanoa, että ”rMon” –palvelun käyttäminen säästää laitteen akkutehoa.

WLAN yhteyksien tapauksessa päivitystoiminto on suunnilleen yhtä nopea kuin laitteiden etsiminen, mutta laitelistan etäpäivitys ei vaadi kuin yhden yhteyden avaamista ja lyhyttä tiedonsiirtoa, kun taas laitteiden haku vaatii oman yhteyden jokaiseen löydettyyn laitteeseen sekä tiedonsiirron.

Bluetooth yhteyksillä ero on huomattavasti suurempi, Bluetooth –protokollan ensimmäisen version hakutoiminto on sekä epävakaa että hidas. Pienessä testiverkossa kaikki laitteet löytävä hakutoiminto ei yleensä onnistu yhdellä yrityksellä, ja onnistuessaankin sen suorittaminen vie jopa 30 sekuntia aikaa. Bluetooth-yhteyden läpi lähetettynä ”rMon” –etäpäivitys kestää alle yhden sekunnin samassa testiverkossa.

Sen lisäksi, että palvelun etäpäivitys on vähintään yhtä nopea kuin verkkorajapintojen käyttämät laitteidenetsintämenetelmät, optimitilanteessa verkkotiedot lähetetään laitteelta toiselle keskimäärin vain noin 2 minuutin välein, kun taas aktiivinen haku etsii laitteita 10-30 sekunnin välein. Toisin sanoen verkkolaitetta kuormitetaan vain 20-40% siitä kuormitusasteesta, mitä normaali aktiivinen laitteiden haku kuormittaa. Samalla tarvittavien yhteyksien määrä laskee useista yhteyksistä yhteen.

LÄHTEET

[1] **Use of distributed resources in mobile environment**
Tommi Kallonen and Jari Porras
Lappeenranta University of Technology, Department of Information Technology
P.O Box 20, 53851 Lappeenranta, Finland
E-mail: {tommi.kallonen, jari.porras}@lut.fi

[2] **Dynamic Power Management Strategies**
W ith in th e IE E E 8 0 2 .1 1 Standard _
Andrea Acquaviva, Edoardo Bont`a, Em anuele L attanz i
Universit`a d i Urb ino C a rlo B o .
Istitu to d i S c ienze e T ecno lo g ie d ell Info rm a z io ne
P ia z z a d ella R ep u b b lic a 1 3 , 6 1 0 2 9 Urb ino , Ita ly
{acquaviva,bonta,lattanzi}@sti.uniurb.it
http://www.sti.uniurb.it/bonta/book_chapters/sfm05moby_02.pdf
[Viittaus pvm. 7.9.2007]

[3] **An Introduction to Bluetooth**
By David Blankenbeckler
http://www.wirelessdevnet.com/channels/bluetooth/features/bluetooth.html
[Viitaus pvm. 7.9.2007]

[4] **PEERHOOD SUBSYSTEM SPECIFICATION**
14.5.2004
version 0.2

[5] **Ars File: CPU Theory & Praxis**
http://arstechnica.com/articles/paedia/cpu.ars?page=1
[ensimmäinen tutustuminen 2005, kandityön parissa 9.9.2007]

[6] **The Pentium: An Architectural History of the World's Most Famous Desktop Processor (Part I)**
http://arstechnica.com/articles/paedia/cpu/pentium-1.ars/1
[9.9.2007]

[7] **White Paper - ARM11 Microarchitecture**
http://www.arm.com/pdfs/ARM11MicroarchitectureWhitePaper.pdf
[11.9.2007]

[8] **ARM architecture**
ARM7, ARM9, TDMI...
http://tisu.it.jyu.fi/embedded/TIE345/luentokalvot/Embedded_3_ARM.pdf
[13.9.2007]

[9] **The Pentium: An Architectural History of the World's Most Famous Desktop Processor (Part I)**
http://arstechnica.com/articles/paedia/cpu/pentium-1.ars
[17.9.2007]

[10] **The Pentium: An Architectural History of the World's Most Famous Desktop Processor (Part II)**
http://arstechnica.com/articles/paedia/cpu/pentium-2.ars/2
[17.9.2007]

[11] **Process vs. Task Migration**
Yves Paindaveine and Dejan S. MilojiEiC
OSF Research Institute
Proceedings of the 29th Annual Hawaii International Conference on System Sciences - 1996

[12] **Dynamic task migration in home-based software DSM systems**
Weisong Shi; Weiwu Hu; Zhimin Tang; Eskicioglu, M.R.;
High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on
3-6 Aug.

[13] **Adaptive offloading inference for delivering applications in pervasive computing environments**
Xiaohui Gu; Nahrstedt, K.; Messer, A.; Greenberg, I.; Milojicic, D.;
Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on
23-26 March 2003 Page(s):107 - 114

[14] **Extending the lifetime of a network of battery-powered mobile devices by remote processing: a Markovian decision-based approach**
Rong, P.; Pedram, M.;
Design Automation Conference, 2003. Proceedings
2-6 June 2003 Page(s):906 - 911

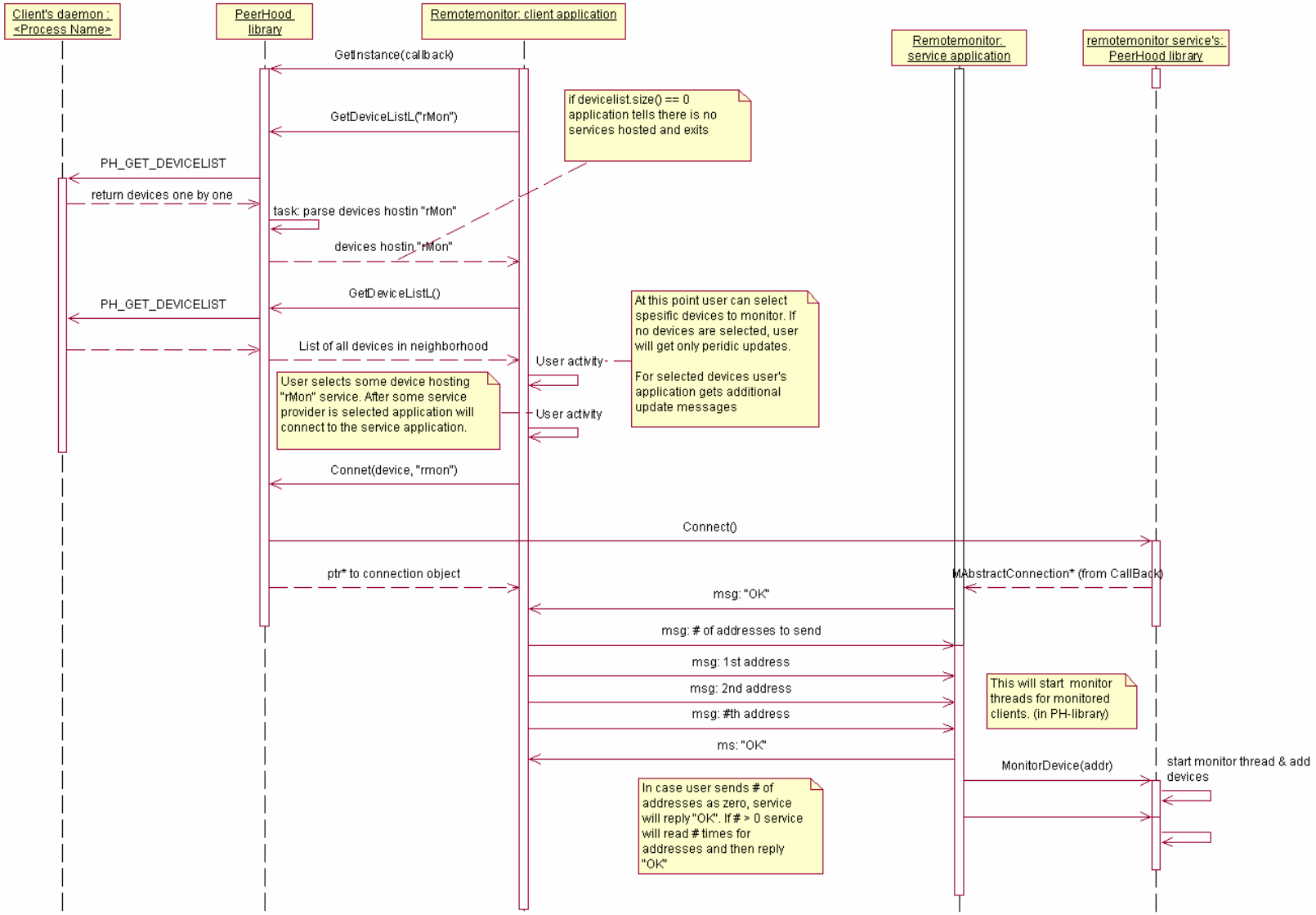
[15] **Mobile to base task migration in wireless computing**
Gitzenis, S.; Bambos, N.;
Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on
2004 Page(s):187 – 196

[16] **Peer-to-peer communication approach for a mobile environment**
Porras, J.; Hiirsalmi, P.; Valtaoja, A.;
System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on
5-8 Jan. 2004

[17] **Personal trusted device in personal communications**
Porras, J.; Jappinen, P.; Hiirsalmi, P.; Hamalainen, A.; Saalasti, S.; Koponen, R.; Keski-Jaskari, S.;
Wireless Communication Systems, 2004. 1st International Symposium on
20-22 Sept. 2004 Page(s):388 - 392

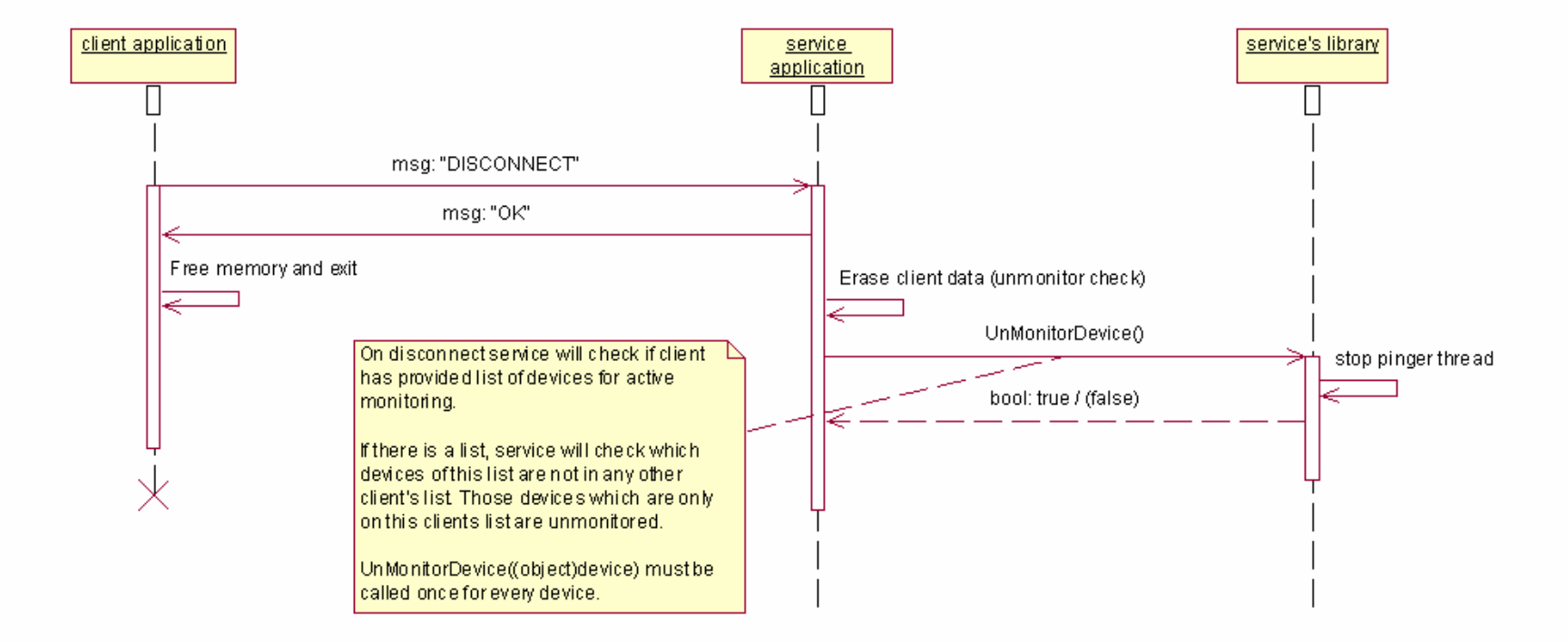
LIITE 1: MSC –KAAVIOT: CONNECT, DISCONNECT, ABORT JA UPDATE

Yhteydemuodostuksen MSC –kaavio. (Connect)



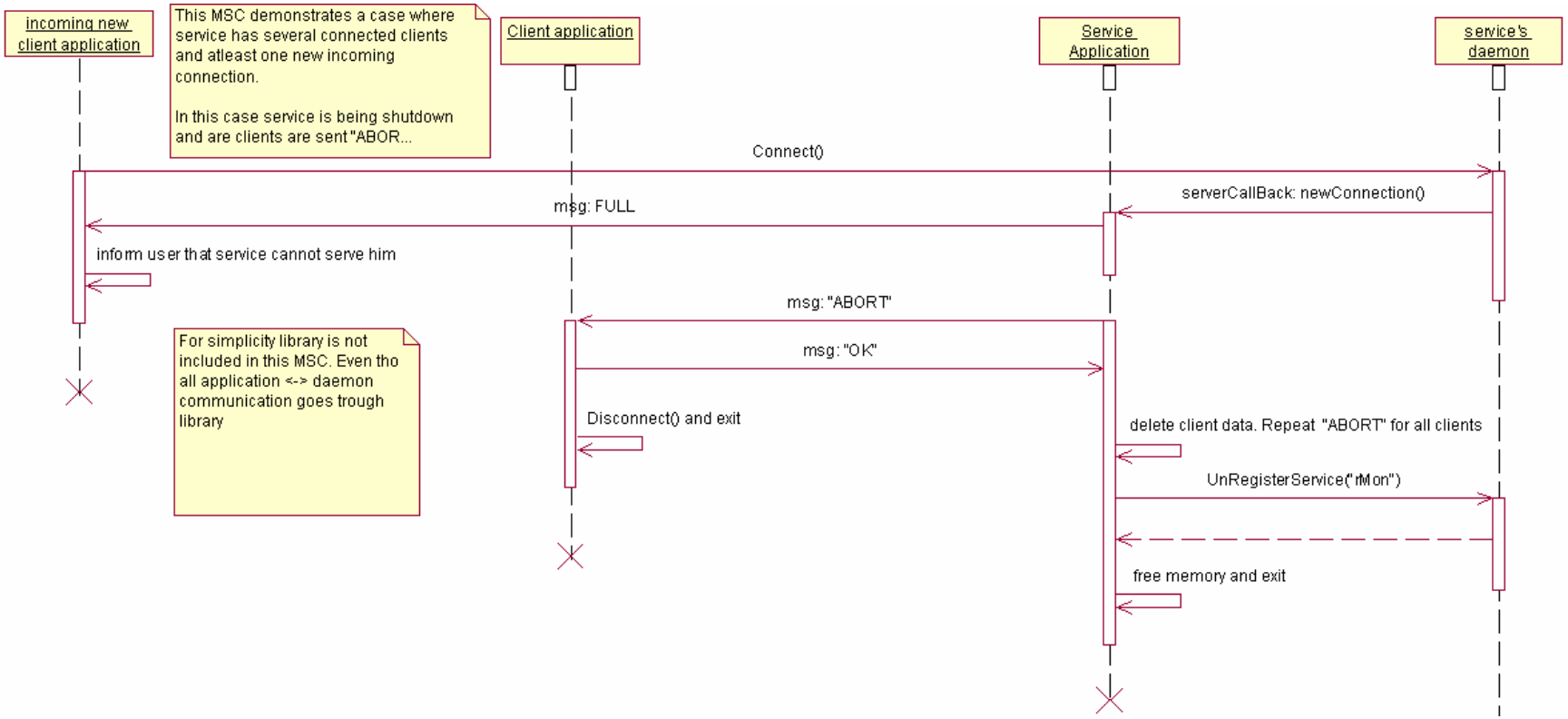
Kuva 6. "rMon" -palvelun yhteydemuodostus (handshake).

Palvelusta poistumisen MSC –kaavio. (Disconnect)



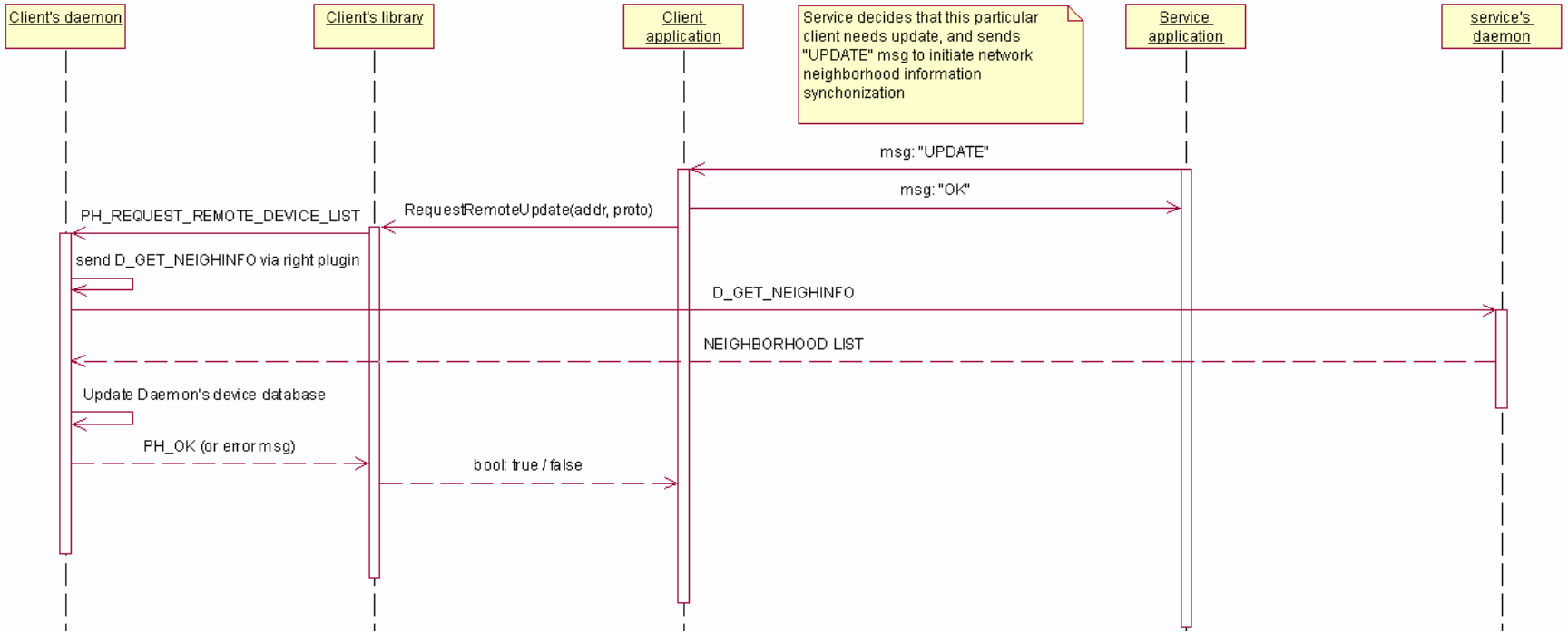
Kuva 7. "rMon" -palvelun yhteydenkatkaisu.

Palvelun alasajon MSC –kaavio. (Abort)



Kuva 8. "rMon" -palvelun alasajo.

Palvelun päivitystapahtuman MSC –kaavio. (Update)



Kuva 9. "rMon" -palvelun verkkonaapuritiedon päivitys.