Suunnitteludokumentti

DHT – Distributed Hash Table

Helsinki 4.4.2004

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Marko Räihä
Risto Saarelma
Antti Salonen
Tuomas Toivonen
Tomi Tukiainen
Simo Viitanen

Asiakas

Jussi Lindgren

Johtoryhmä

Juha Taina
Turjo Tuohiniemi

Kotisivu

`http://www.cs.helsinki.fi/group/dht/`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
0.9	3.4.2004	Tarkastettava versio

Sisältö

1 Johdanto

Tämä dokumentti liittyy kevään 2004 ohjelmistotuotantoprojekti-kurssiin. Dokumentissa kuvataan Kademlia-algoritmiin perustuvan DHT-ohjelmiston toteutuksen suunnitelma.

Dokumentti on tarkoitettu projektiryhmän, asiakkaan ja ohjelmistotuotantoprojekti-kurssin henkilöstön käyttöön.

1.1 Terminologiaa

Seuraavassa taulukossa on selitetty lyhyesti dokumentissa käytettävää terminologiaa:

Vertaisverkko	Tasa-arvoisista solmukoneista koostuva verkko, P2P-verkko
DHT	Distributed Hash Table, hajautettu hajautustaulu
GNUnet	avoin sovelluskehys turvallisille vertaisverkkoratkaisuille (http://www.ovmj.org/GNUnet/)
Kademlia	eräs kirjallisuudessa [MM] kuvattu DHT-toteutus

1.2 Dokumentin rakenne

Luvussa 2 annetaan kokonaiskuva tuotettavasta ohjelmistosta. Luvussa 3 esitetään ohjelmiston tarjoaman ohjelmointirajapinnan suunnitelma. Luvussa 4 esitetään DHTn ja siihen liittyvien algoritmien toiminnan suunnitelma. Luvussa 5 määritellään DHT-solmujen välisen kommunikoinnin toteuttavien etäproseduurikutsujen toteutus. Luvussa 6 esitetään DHT-solmujen datakerroksen suunnitelma. Luvussa 7 vedetään yhteen toteutuksen eri osien tarvitsemat tietorakenteet ja näihin mahdollisesti liittyvät käsittelijäfunktiot. Luvussa 8 esitetään testauksen suunnittelu ja luvussa 9 esitetään suunnitelma toteutettavasta esimerkkisovelluksesta.

2 Kokonaiskuva

Tämä luku sisältää kuvauksen toteutettavasta ohjelmistosta. Kuvaus on korkealla tasolla ja painopiste on järjestelmäarkkitehtuurissa. Yksittäisten komponenttien määrittelyyn paneudutaan tarkemmin dokumentin muissa luvuissa.

2.1 DHT

Projektin tarkoituksena on tuottaa vertaisverkkoon perustuva toteutus hajautetusta hajautustaulusta (distributed hash table, DHT). Hajautettu hajautustaulu tarjoaa samat perustoiminnot kuin normaali hajautustaulu, mutta mahdollistaa taulun käsittelystä aiheutuvan kuorman jakamisen useammalle koneelle, ja jos tauluun tallennettua dataa monistetaan useampaan solmuun, voidaan hajautetulle hajautustaululle saada käyttökelpoinen luotettavuus perusperiaatteeltaan epäluotettavassa vertaisverkossa.

Toteutettava DHT-verkko koostuu mielivaltaisesta joukosta tasa-arvoisia solmuja (jatkossa "DHT-solmu"), joista jokainen tarjoaa saman toiminnallisuuden. Kukin DHT-solmu on ohjelmaprosessi, ja niitä voi siten sijaita useampia yhdessä verkon fyysisessä palvelimessa. DHT-verkon toimintalogiikan kannalta solmujen sijaitsemisella samassa fyysisessä palvelimessa tai aliverkossa ei ole merkitystä. DHT-solmujen välinen viestintä perustuu etäproseduurikutsuihin, ja koska solmujen välinen verkko voi olla julkinen, on DHT-solmujen välinen viestintä salattua.

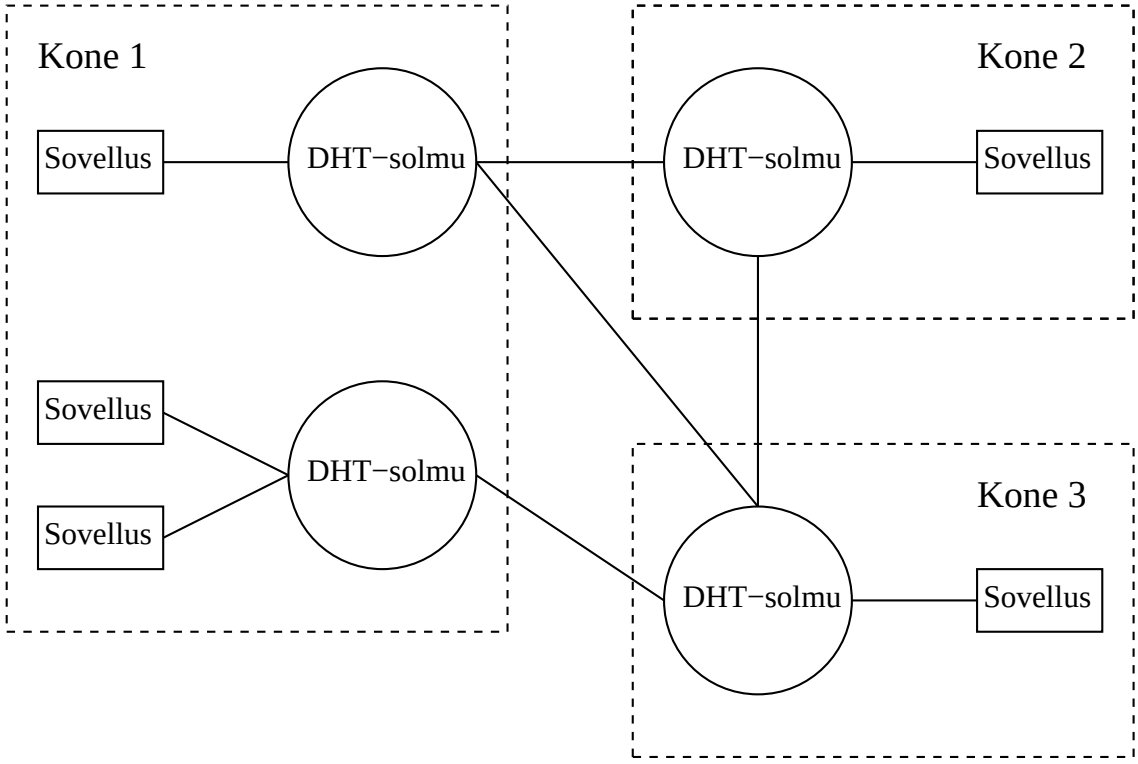
DHT-solmu tarjoaa toteuttamansa toiminnallisuuden ohjelmointirajapinnan muodossa. Ohjelmointirajapinnan avulla voidaan rakentaa erilaisia DHT-verkkoa hyödyntäviä sovelluksia. DHT-verkkoa hyödyntävät sovellukset toimivat verkossa kytkeytymällä asiakas-palvelinsuhteeseen yhden DHT-solmun kanssa. Sovelluksen ja sitä palvelevan solmun oletetaan sijaitsevan vähintäänkin samassa turvallisessa aliverkossa, sillä näiden välinen viestintä on suojaamatonta.

2.2 Kademlia

Hajautetun hajautustaulun toteutus perustuu Petar Maymounkovin ja David Mazieresin julkaisussaan [MM] esittämään Kademlia-algoritmiin. Algoritmin kantavana ajatuksena hajautustaulun jakamisessa usean solmun kesken on käyttää hyväksi verkossa tapahtuvaa hyötyliikennettä (haut, tallennukset) verkon reitityksessä niin suuressa määrin että pelkkää reititysliikennettä ei tarvita lainkaan.

2.3 GNUnet

GNUnet on GPL-lisenssin alainen turvallinen sovelluskehys vertaisverkkototeutuksille. Verkossa ei ole mitään keskitettyä tai luotettua palvelua vaan kaikki on hajautettua. Liikenne vertaisverkossa on salattua ja microtaloutta käytetään liikenteen ja käytön tasapuolistamiseksi. Tämä tarkoittaa sitä että mitä enemmän solmu tekee verkon hyväksi niin sitä



Kuva 1: Verkon fyysisten koneiden, DHT-solmujen, niihin kytkeytyneiden asiakkaiden suhteet.

enemmän se saa käyttää verkon resursseja. DHT-verkko toteutetaan tämän sovellyskehyksen päälle erillisenä moduulina.

2.4 Toteuttettavat osa-alueet

Toteutettava DHT-moduuli voidaan jakaa pääpiirteissään seuraaviin toteutettaviin osa-alueisiin.

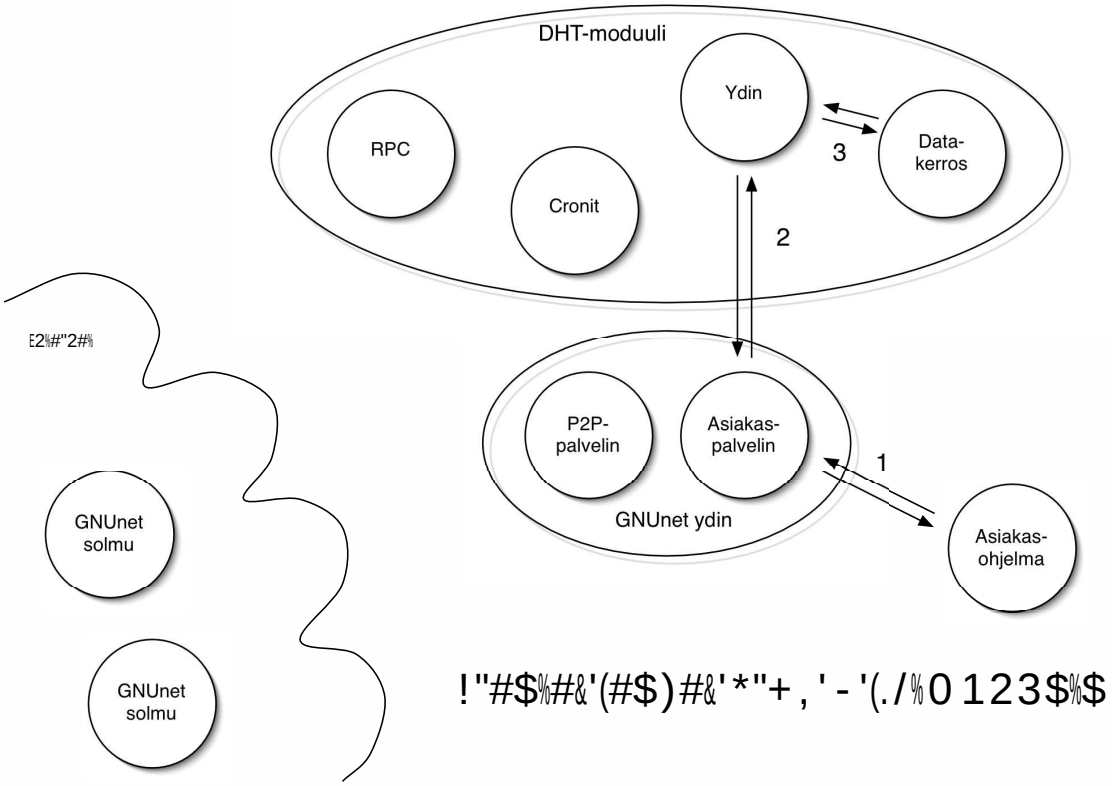
- ydin - käsittää kademlian algoritmit ja asiakasrajapinnan
- data-kerros - vastaa solmun vastuulla olevan tiedon tallentamisesta
- cronit - verkon ylläpitoon liittyvät ajastetut toiminnot
- RPC - verkon solmujen välinen kommunointi

2.5 Osa-alueiden yhteistoiminta

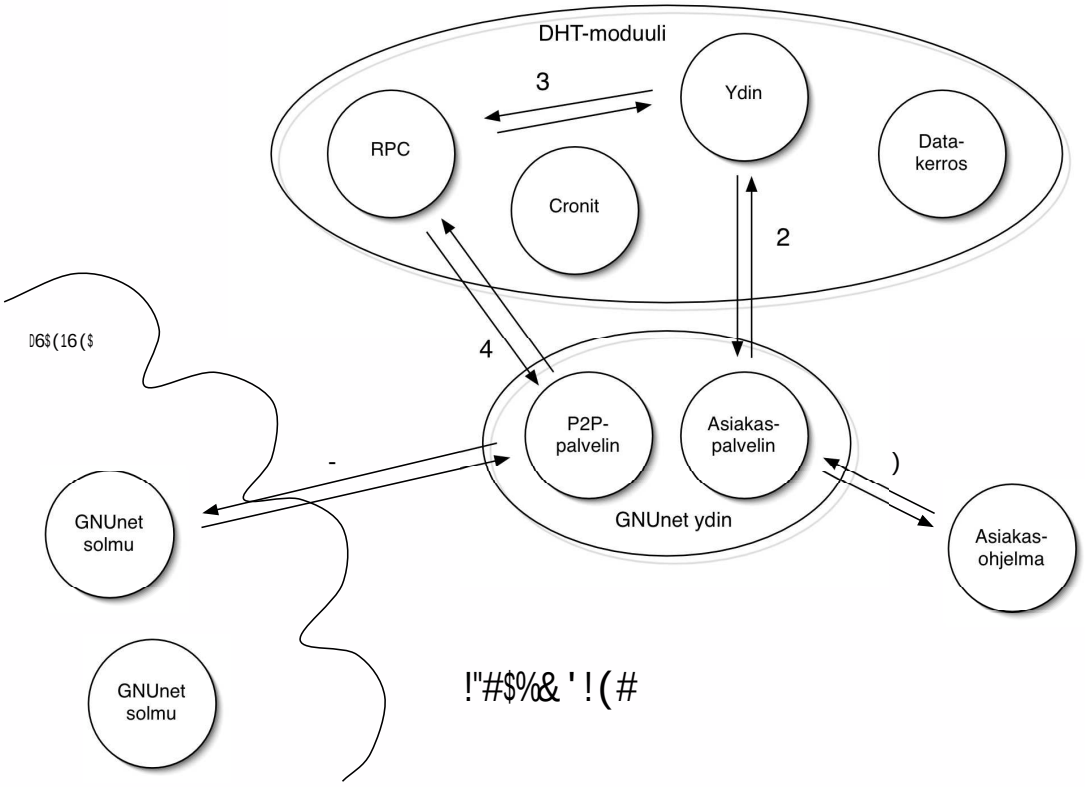
Oheisissa kontrollivuokaavioissa esitetään kontrollin siirtyminen asiakasohjelman, GNUnet-ytimen, DHT-moduulin sekä GNUnet-verkon välillä. Kuvattuna on kaikki asiakasohjelman käytettävissä olevat operaatiot sekä tärkeimmät DHT:n ylläpitoon tarvittavat ope-

raatiot. Kun useammalla operaatiolla on sama kontrollivuo, niin ne on kuvattu samassa kuvassa. DHT-moduulin osista on kerrottu yllä. GNUnetin ydin on jaettu kuvissa kahteen osaan kaavioiden selkiyttämiseksi kun sillä on kaksi erillistä rooliakin. Se tarjoaa palvelut sekä asiakasohjelmalle että GNUnet-verkon suuntaan. Kumpikin palvelu on toteutettu ohjelmakehyksessä siten että kehys kutsuu sitä käyttävän ohjelman, tässä tapauksessa DHT-moduulin, rekisteröimiä funktioita. GNUnet-ydin ja DHT-moduuli muodostavat yhdessä GNUnet-solmun.

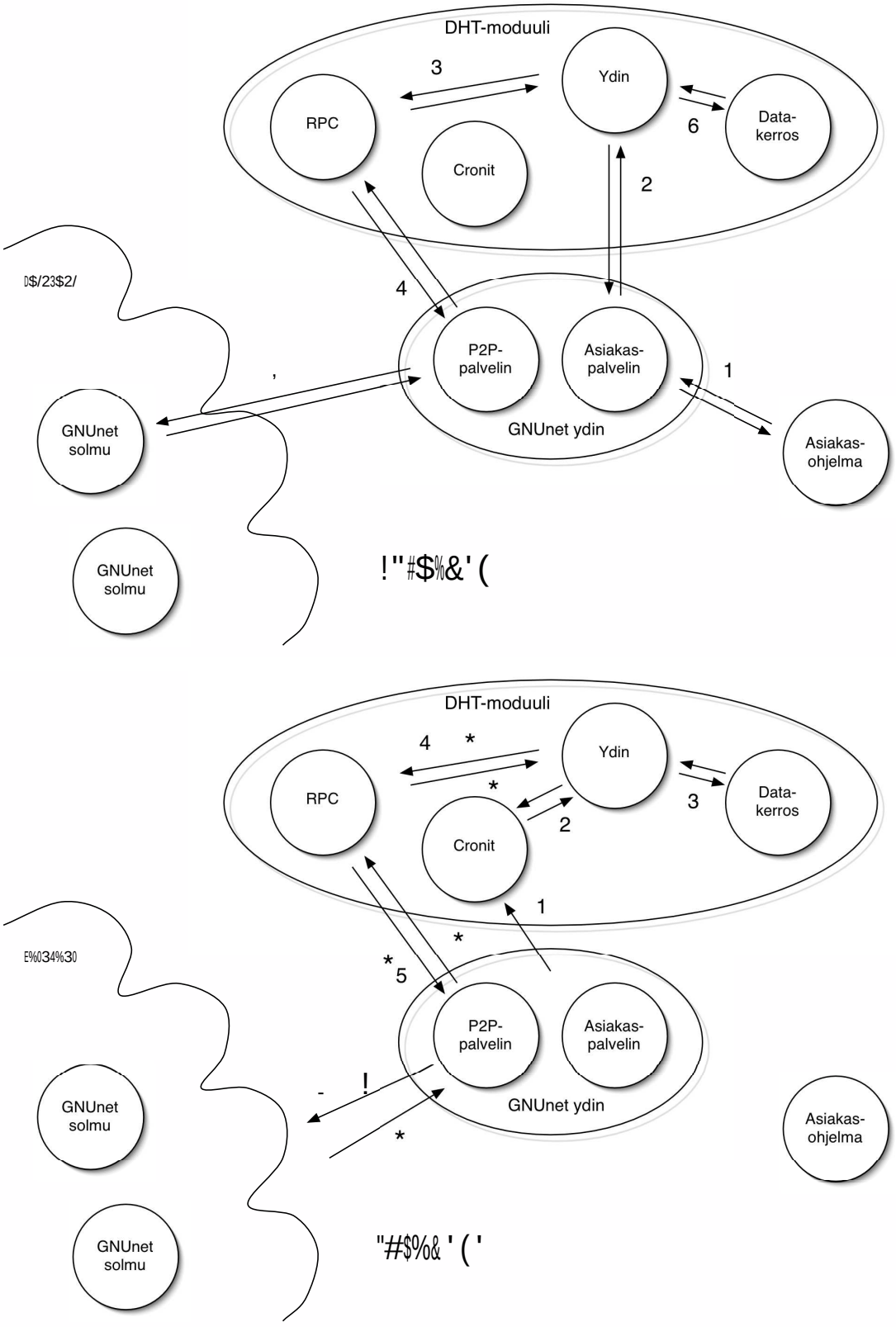
Hajautustaulun luonti, hajautustaulusta poistuminen, oman tiedon uudelleenjulkaisun estäminen sekä oman tiedon listaaminen tapahtuvat asiakasohjelman ja solmun sisäisesti.



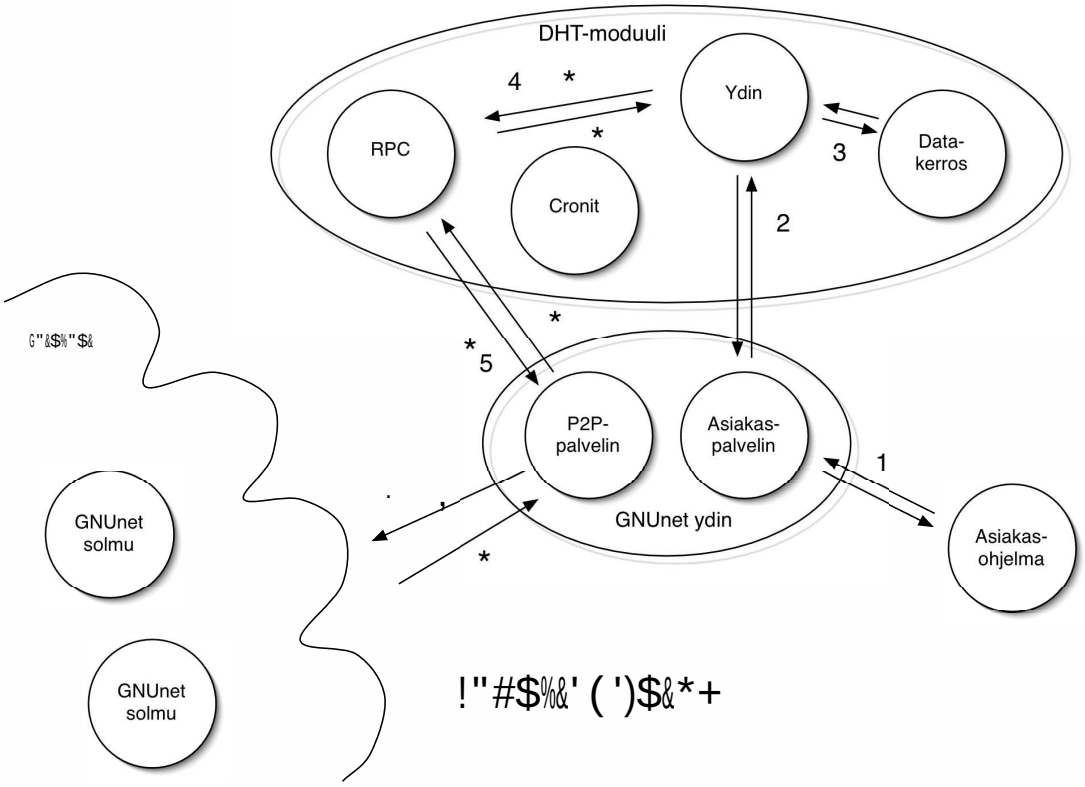
Toisen tunnetun solmun sisältämät hajautustaulut voidaan saada selville, kun tiedetään vähintään toisen solmun osoite.



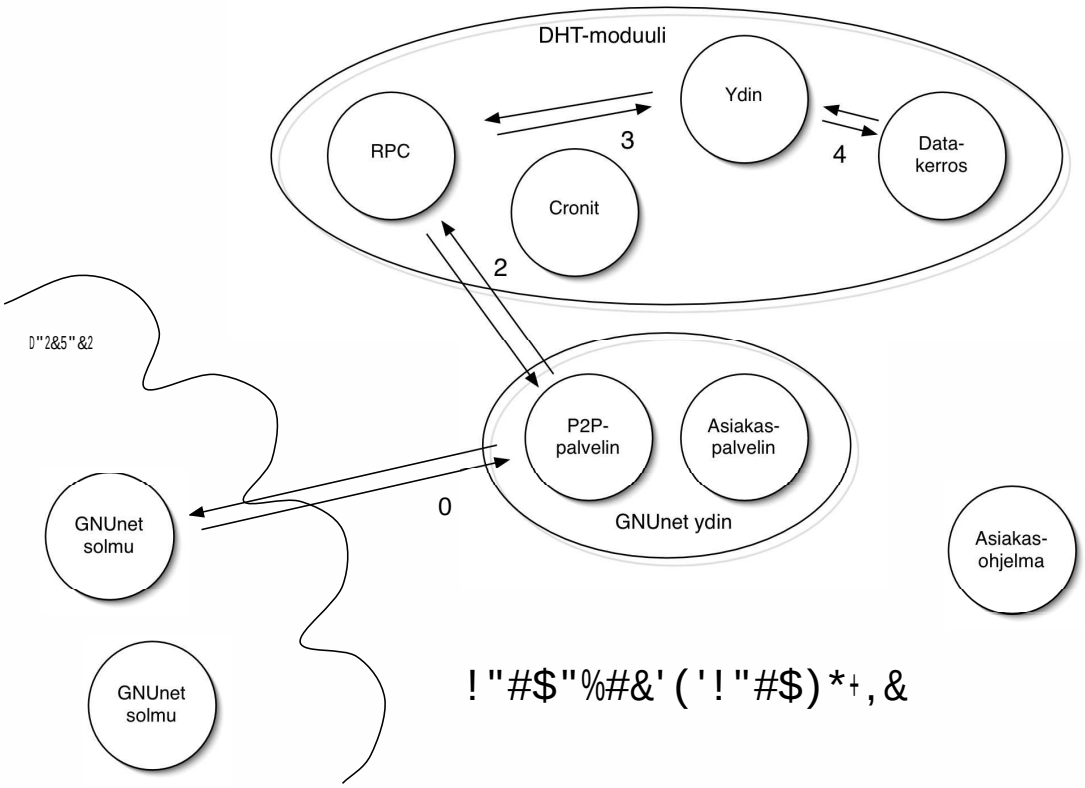
Olemassaolevaan hajautustauluun liittyminen on monivaiheinen tapahtuma. Ensin haetaan toiselta solmulta kaikki siihen liittyvä tieto. Saatuja tietoja alkuarvoina käyttäen päivitetään ne omalle solmulle sopiviksi kademlian sääntöjen mukaisesti.



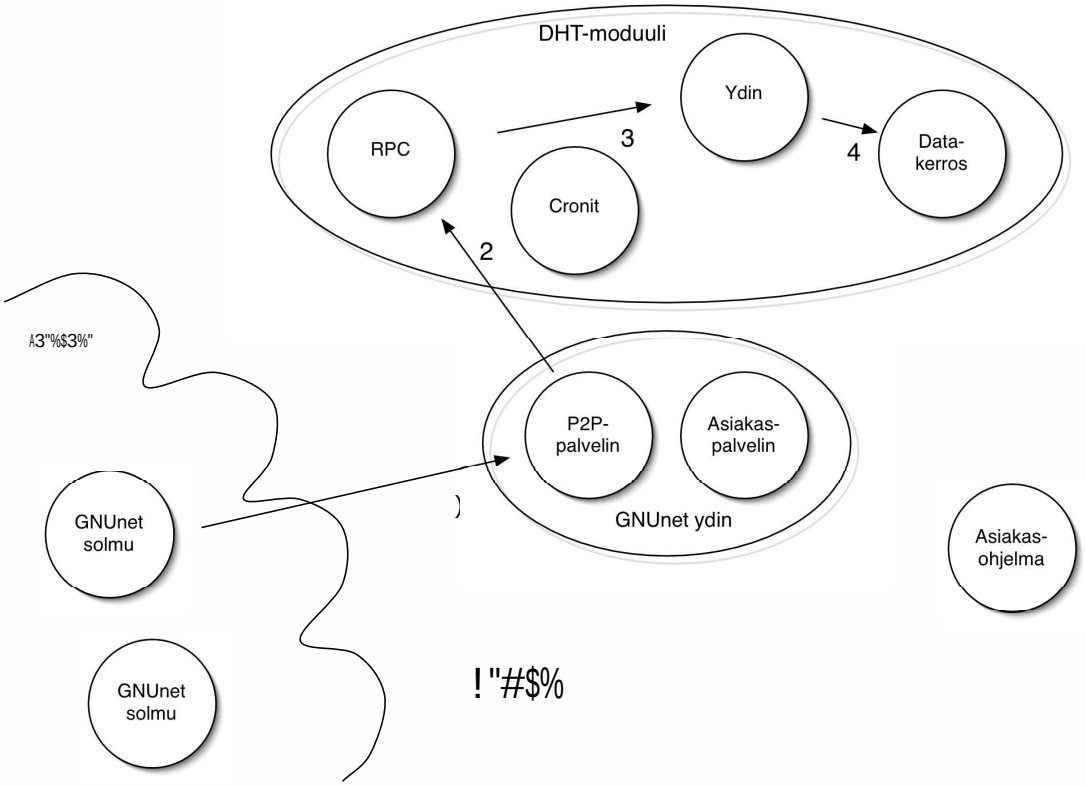
Hajautustaulusta haku ja sinne tallennus käsittää liikennettä moneen suuntaan. Tässä kuvaus asiakasohjelman ja sen käyttämän solmun näkökulmasta.



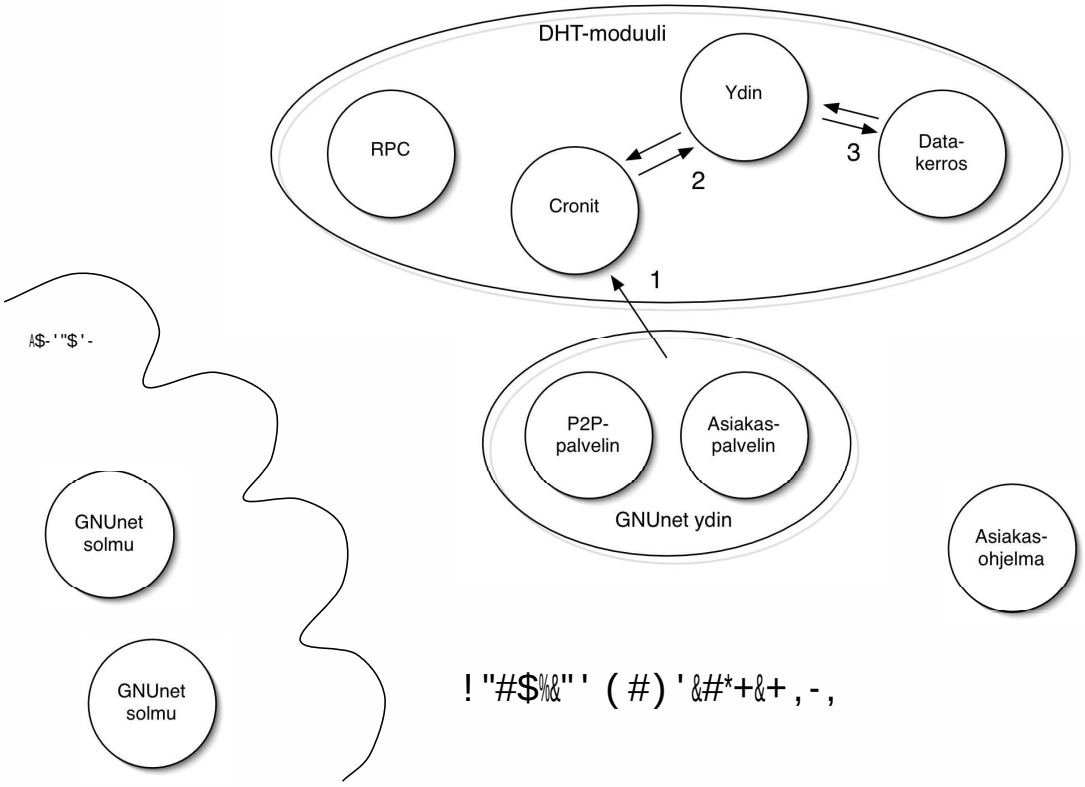
GNUnet-verkosta tulevat find_value-kyselyt etenevät datakerrokseen asti ja find_node-kyselyt pysähtyvät jo DHT-moduulin ytimeen.



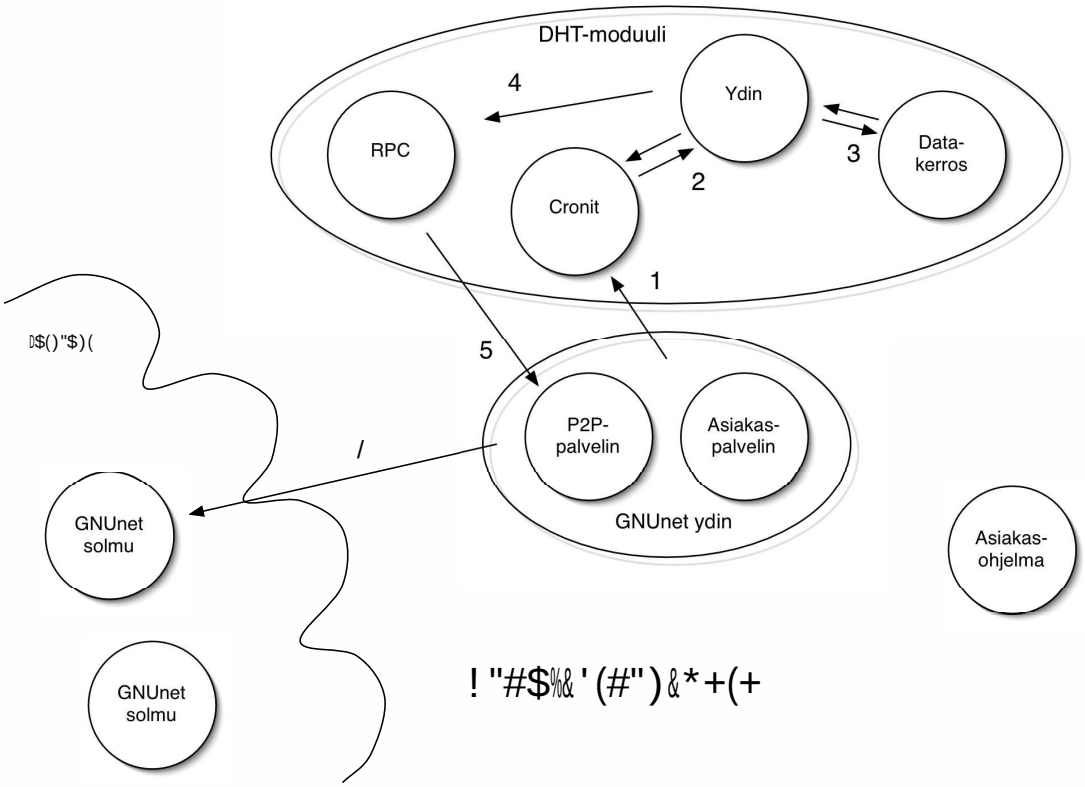
Toiselta solmulta tuleva tallennuskomennon vaikutuksesta data tallennetaan datakerrokseen.



GNUnet-solmu huolehtii automaattisesti säännöllisin väliajoin vanhan datan poistosta.



GNUnet-solmu julkaisee automaattisesti säännöllisin väliajoin voimassa olevat datat.



Tarkempi toimintojen kuvaus löytyy dokumentin teknisemmistä luvuista.

3 Ohjelmointirajapinta

Tässä kappaleessa määritellään DHT:n ohjelmointirajapinnan, eli APIn, tarjoamat funktiot ja niiden toiminta. APIn tarjoamat funktiot on tässä jaettu kahteen osaan: palvelufunktioihin ja apufunktioihin. Palvelufunktiot tarjoavat rajapinnan DHT-moduulin tarjoamaan toiminnallisuuteen, kun apufunktiot liittyvät virheiden havaitsemiseen ja tulosten käsittelyyn.

Apufunktiot ovat APIa käyttävässä koneessa lokaalisti suoritettavia yksinkertaisia funktioita, joiden avulla Asiakasohjelma voi tarkastaa, onnistuiko jonkin palvelufunktion suoritus, ja jos ei onnistunut, mikä meni pieleen. Asiakasohjelman on myös tarkoitus käyttää apufunktioita saamiensa tulosten läpikäymiseen, jolloin tulosjoukon sisäisen rakenteeseen liittyvä monimutkaisuus on piilotettu Asiakasohjelmalta.

Kaikkien palvelufunktioiden osalta funktion suoritukseen liittyy kolme toimijaa. Toimijat ovat nimeltään Asiakasohjelma, API ja GNUnetDHT. Asiakasohjelma ja API kommunikoivat keskenään s.e. asiakasohjelma kutsuu APIn funktioita tavallisin funktiokutsuin. API ja GNUnetDHT kommunikoivat keskenään luotettavan TCP-kanavan välityksellä client-server-tyylisesti s.e. API on asiakas. Muita yhteyksiä toimijoiden välillä ei ole.

Minkä tahansa palvelufunktion suorituksen aloittava toimija on Asiakasohjelma, joka kutsuu jotain DHT:n tarjoaman APIn palvelufunktiota. Tässä vaiheessa Asiakasohjelma jää odottamaan funktion paluuta synkronisesti.

Seuraavassa vaiheessa API luo funktiokutsua vastaavan TCP-viestin, jonka se lähettää luotettavan TCP-kanavan yli GNUnetDHT-moduulille. Tämän jälkeen API jää odottamaan GNUnetDHT:n antamaa tulosta funktiokutsulle.

Kolmannessa vaiheessa GNUnetDHT-moduuli suorittaa APIlta saamansa viestin perusteella jonkin tarjoamansa palvelun, esimerkiksi <avain,arvo>-parien hakemisen. Tässä vaiheessa palvelua suorittava GNUnetDHT-moduuli voi olla yhteydessä useisiin ulkopuolisiin GNUnetDHT-moduuleihin.

Neljännessä vaiheessa GNUnetDHT-moduuli lähettää suorittamansa palvelun tulokset luotettavan TCP kanavan yli APIlle.

Viidennessä vaiheessa API muodostaa tuloksista Asiakasohjelman käyttöön soveltuvan tietorakenteen ja palauttaa sen Asiakasohjelmalle.

Tarkempi kuvaus kontrollin siirtymisestä palvelufunktion suorittamisen yhteydessä toimijalta toiselle löytyy dokumentin osasta TODO.

3.1 Rajapinnan funktioiden palautusarvot

Sovellusrajapinta palauttaa funktioiden arvoina osoittimia tietorakenteisiin, joiden avulla funktion tulosta pääsee käyttämään DHT:n API-rajapinnan tarjoamien apu funktioiden avulla. Lisäksi joitain saatuja tietorakenteita käytetään palvelufunktioiden parametreina.

3.2 Rajapinnan palvelufunktiot

Palvelufunktiot ovat niitä funktioita, joiden suorittamiseen API-rajapinta tarvitsee GNUnetDHT-moduulia. DHT:n tarjoamassa API-rajapinnassa on määritelty ainoastaan tässä kappaleessa määritellyt palvelufunktiot.

3.2.1 create

Luo uuden hajautustaulun, johon luojasolmu liittyy automaattisesti. Onnistuneen luonnin jälkeen solmu voi alkaa suorittaa hakuja ja tallennuksia tauluun.

toteutus DHT_TableHandle *create(const DHT_TableMetaData *meta, const DHT_TableConfig *config)

parametrit

- meta - Sisältää kaiken luotavaan tauluun liittyvän metadatan, ks dokumentin kohta TODO1.
- config - Sisältää kaiken luotavaan tauluun liittyvät asetustiedot, ks dokumentin kohta TODO2.

paluuarvo

- Sisältää luodun taulun tilainformaation, ks.dokumentin kohta TODO3.

huomiot

- Funktion suorittamisen yhteydessä DHT-moduuli muodostaa luotavalle taululle tunnisteen, joka toteutetaan globaalisti yksikäsitteiseksi. Tämä tehdään taulun luojasolmun GNUnet osoitteen ja kellonajan avulla. Asiakasohjelma ei kuitenkaan missään vaiheessa ole suoraan tekemisissä taulun tunnisteen kanssa, vaan se on kapseloitu APIn palauttamaan DHT_TableHandle-tietueeseen.
- Funktion suorittamisen yhteydessä DHT-moduuli varaa kekomuistista tilaa yhdelle DHT_TableConfig-tietueelle, johon se kopioi taulun asetukset APIlta saamastaan viestistä. Samalla DHT-moduuli luo tauluun liittyvän reititystaulun, jota Kademlian RPC-kutsut hyödyntävät, ks. dokumentin kohta TODO.
- Kun DHT-moduuli on luonut taulun, lähettää se luodun taulun tiedot TCP-yhteyden yli APIlle. API varaa lopuksi kekomuistista tilaa yhdelle DHT_TableHandle-tietueelle, johon se kirjoittaa kaiken sovellusohjelman tarvitseman tiedon taulusta, ja palauttaa osoittimen tähän tietueeseen. Sovellusohjelma voi käyttää tätä osoitinta esimerkiksi taulun tietojen hakemiseen, tauluun tehtävien operaatioiden parametrina tai mahdollisen virhetilanteen havaitsemiseen.

3.2.2 join

Liittää sen DHT-moduulin (solmun), johon kutsu kohdistuu, hajautustauluun. Liittyminen tapahtuu toisen, taulussa jo olevan, solmun avulla. Onnistuneen liittämisen jälkeen solmu voi alkaa suorittaa hakuja ja tallennuksia tauluun.

toteutus DHT_TableHandle *join(DHT_NodeAddress *address, HASH160 *table_id)

parametrit

- address - sen solmun osoite, jonka kautta tauluun liitytään, ks. dokumentin kohta TODO.
- table_id - id-numero, joka identifioi liityttävän taulun globaalisti.

paluuarvo

- Sisältää liitytyn taulun tilainformaation, ks. dokumentin kohta TODO3.

huomiot

- Funktio palauttaa kontrollin sovellusohjelmalle, kun API saa DHT-moduulilta operaation tuloksen, joka sisältää ne liitytyn taulun tiedot, jotka API ja sovellusohjelma tarvitsevat käyttääkseen taulua myöhemmin.
- Funktion suorittamisen jälkeen DHT-moduuliin voi tallentua tauluun tallennettuja <avain,arvo>-pareja.
- Funktion suorittamisen yhteydessä DHT
  - lataa toisesta, taulussa jo olevasta, solmusta taulun config-tiedot, joita se noudattaa suorittaessaan tauluun liittyviä operaatioita
  - luo tauluun liittyvän reititystaulun, ks. dokumentin kohta TODO4.

3.2.3 leave

Erottaa sen DHT-moduulin (solmun), johon kutsu kohdistuu, hajautustaulusta. Eroamisen jälkeen solmu ei enää sisällä tauluun liittyviä <avain,arvo>-pareja, eikä sen kautta voi myöskään suorittaa hakuja tai tallennuksia tauluun.

toteutus DHT_LeaveResult *leave(DHT_TableHandle *table)

parametrit

- table - sen taulun handle, josta asiakasohjelma haluaa erota.

paluuarvo

- Tietorakenne, jonka avulla voidaan selvittää operaation onnistuminen tai epäonnistuminen syineen.

huomiot

- Eroamisen yhteydessä solmu lähtee hajautustaulusta vieden kaikki sisältämänsä <avain,arvo>-parit mukanansa. Solmu ei siis ala siirtämään <avain, arvo>-pareja taulun konsistenssin säilyttämiseksi. Normaalissa tilanteessa taulun konsistenssi säilyy, koska kaikki <avain,arvo>-parit on tallennettu useampaan solmuun, ks dokumentin kohta TODO5.

- Eroamisen yhteydessä DHT-moduuli poistaa kaikki tauluun liittyvät tiedot itseltään, jonka jälkeen se ei enää tunnista edes taulun id:ta.

3.2.4 insert

Tallentaa <avain,arvo>-parin hajautustauluun. Tallentamisen jälkeen hajautustauluun liittyneet solmut voivat hakea avaimen avulla tallennetun <avain,arvo>-parin taulusta.

toteutus DHT_InsertResult *insert(DHT_TableHandle *table, DHT_DataContainer *key, DHT_DataContainer *value)

parametrit

- table - liityttävän taulun handle
- key - DHT:n ymmärtämässä muodossa oleva tietue, joka sisältää tallentamisessa käytettävän avaimen.
- value - DHT:n ymmärtämässä muodossa oleva tietue, joka sisältää talletettavan datan.

paluuarvo

- Tietorakenne, jonka avulla voidaan selvittää operaation onnistuminen tai epäonnistuminen syineen.

huomiot

- Tallentamisen yhteydessä pyynnön saava DHT-moduuli tallentaa <key,value>-parin Kademlia-algoritmin mukaisesti hajautustauluun liittyneisiin solmuihin siten, että sen voi myöhemmin hakea hajautustaulusta.

- API palauttaa kontrollin sovellusohjelmalle heti, kun DHT-moduuli (solmu) ilmoittaa tallentamisen onnistuneen tai epäonnistuneen.

3.2.5 fetch

Hakee kaikki avaimeen liittyvät data-alkiot hajautustaulusta. Hakemisen jälkeen sovellusohjelma voi hyödyntää data-alkioita vapaasti.

toteutus DHT_FetchResult *fetch(DHT_TableHandle table, DHT_Data_container *key)

parametrit

- table - sen taulun handle, josta haku suoritetaan.
- key - DHT:n ymmärtämässä muodossa oleva tietue, joka sisältää hakemisessa käytettävän avaimen.

paluuarvo

- Tietorakenne, jonka avulla voidaan käydä tulosjoukkoa läpi tai selvittää operaation epäonnistuminen syineen.

huomiot

- DHT-moduuli (solmu) suorittaa hakemisen Kademlia-algoritmin mukaisesti hajautustauluun liittyneistä solmuista.

- Funktio palauttaa kontrollin sovellusohjelmalle kun DHT-moduuli on lähettänyt sille tulosjoukon.

3.2.6 listTables

Hakee joltain DHT-solmulta kaikkien taulujen, joihin solmu on liittynyt, tiedot. Hakemisen jälkeen sovellusohjelma voi hyödyntää saamaansa tietoa vapaasti.

toteutus DHT_ListTablesResult *listTables(DHT_NodeAddress *address)

parametrit

- address - sen solmun osoite jolta pyydetaan liityttyjen taulujen tiedot

paluuarvo

- Tietorakenne, jonka avulla voidaan käydä tulosjoukkoa läpi tai selvittää operaation epäonnistuminen syineen.

huomiot

- Funktio palauttaa kontrollin sovellusohjelmalle heti kun DHT voi päätellä hakemisen onnistuneen tai epäonnistuneen.

- Saatu tieto ei sisällä sellaisia tauluja, joita ei ole pyynnön kohdesolmussa merkitty julkisiksi.

3.2.7 listInsertedData

Listaa kaikki APIn kautta itse tallennetut avaimet, joita DHT-moduuli uudelleenjulkaisee säännöllisesti.

toteutus DHT_ListInsertedDataResult *listInsertedData(DHT_TableHandle table)

parametrit

- table - sen taulun handle, johon tallennetut data-alkiot asiakasohjelma haluaa tietää.

3.2.8 dropInsertedData

Lopettaa APIn kautta itse tallennetun <avain,arvo>-parin uudelleenjulkaisemisen.

toteutus int dropInsertedData(DHT_TableHandle table, DHT_DataContainer key)

parametrit

- table - sen taulun handle, josta <avain,arvo>-parin uudelleenjulkaiseminen halutaan lopettaa
- key - poistettavan <avain,arvo>-parin yksilöivä avain

3.3 APIn ja GNUnetDHT-moduulin välinen protokolla

APIn ja GNUnetDHT-moduulin (solmun) välinen protokolla vaikuttaa suoraan APIn toimintaan, joten sen määrittely sovellusrajapinnan yhteydessä lienee järkevää. Protokollan suunnittelussa lähtökohtana on pidetty sitä, että APIn ja DHT-solmun välille luotava TCP-yhteys on luotettava.

Koska useat APIt voivat käyttää yhtä DHT-moduulia samanaikaisesti, täytyy protokollan mahdollistaa se, että DHT-moduuli voi tunnistaa sitä käyttävät APIt ja siten lähettää oikeat vastaukset oikealle APIlle. Tästä johtuen jokaisella APIlla on tunniste, joka identifioi sen yksilöllisesti muiden samaan DHT-moduuliin yhteydessä olevien APIen joukosta. Tämä tunniste lähetetään jokaisen viestin mukana APIlta DHT-moduulille. Tämän jälkeen protokollan toimintaa kuvaa seuraava toimintosarja.

1. Sovellusohjelma kutsuu APIn funktiota 1.1 API luo ja liittää yksilöllisen tunnisteen pyyntöön - myöhemmin request_id 1.2 Riittää, että request_id on yksilöllinen APIn luomien tunnisteiden joukossa

2. API lähettää palvelupyynnön TCP-kanavan kautta DHT-moduulille (solmulle) 2.1 Jos timeout tai muu virhe keskeyttää siirron, API palauttaa virheen sovellusohjelmalle

3. DHT-moduuli lähettää kuittausviestin APIlle 3.1 DHT-moduuli alkaa suorittamaan palvelua, ellei se ole jo suorittamassa sitä - tämä tarkastetaan request_id:n avulla

4. DHT-moduuli lähettää tiedon palvelun onnistumisesta tai epäonnistumisesta 4.1 Jos timeout tai muu virhe keskeyttää siirron, DHT tallentaa palvelun tulokset toistaiseksi

5. DHT-moduuli palauttaa palvelun tulokset 5.1 Jos timeout tai muu virhe keskeyttää siirron, DHT tallentaa palvelun tulokset toistaiseksi

3.3.1 Protokollan toiminta epänormaaleissa tilanteissa

Epänormaalin tilanteen sattuessa API käyttää yleensä palvelun tilakyselyviestiä päättääkseen seuraavan toimintonsa. Palvelun tilakyselyviestin mukana kuljetetaan lisäksi DHT-moduulille tieto siitä, mitä API odottaa.

Mahdolliset tilakyselyn paluuarvot DHT-moduuli palauttaa jonkun seuraavista arvoista, kun se saa tilakysely-viestin.

1. Operaatio suorituksessa – DHT-moduuli ei vielä tiedä onnistuuko vai epäonnistuuko operaation suoritus

2. Operaatio onnistuu, kesken – DHT-moduuli tietää, että palvelun tuottaminen onnistuu, mutta ei ole vielä valmis

3. Operaatio onnistui, valmis – DHT-moduuli on saanut palvelun tuottamisen valmiiksi onnistuneesti

4. Operaatio epäonnistui – DHT-moduuli on keskeyttänyt palvelun tuottamisen epäonnistuneesti

5. Tuntematon operaatio – DHT-moduulilla ei ole tietoa palvelupyynnöstä

APIn odotusaika operaation onnistumiselle tai epäonnistumiselle päättyy API lähettää palvelun tilakyselyviestin DHT-moduulille käyttäen request_id:ta. Tämä viesti sisältää tiedon, että operaation tuloksia odotetaan. Tässä kuvataan, miten API käyttäytyy eri paluuarvojen tapauksissa. Numero tarkoittaa tilakyselyyn saatua paluuarvoa, ja selitys tarkoittaa sitä, miten API reagoi paluuarvoon.

1. Mikäli maksimiodotusaika ei ole ohi, API palaa odottamaan. Muuten palautetaan virhe sovellusohjelmalle.

2. Aletaan odottamaan tulosjoukkoa.

3. Aletaan odottamaan tulosjoukkoa.

4. API palauttaa virheen sovellusohjelmalle.

5. API palauttaa virheen sovellusohjelmalle.

Mikäli API ei saa vastausta tai tiedonsiirto epäonnistuu, palauttaa API virheen sovellusohjelmalle.

APIn odotusaika operaation tulosjoukolle päättyy API lähettää palvelun tilakyselyviestin DHT-moduulille käyttäen request_id:ta. Tämä viesti sisältää tiedon, että operaation tuloksia odotetaan. Tässä kuvataan, miten API käyttäytyy eri paluuarvojen tapauksissa. Numero tarkoittaa tilakyselyyn saatua paluuarvoa, ja selitys tarkoittaa sitä, miten API reagoi paluuarvoon.

1. Koska tätä paluuarvoa ei pitäisi enää tässävaiheessa tulla, palautetaan virhe sovellusohjelmalle.

2. Mikäli maksimiodotusaika ei ole ohi, API palaa odottamaan. Muuten palautetaan virhe sovellusohjelmalle.

3. Mikäli maksimiodotusaika ei ole ohi, API palaa odottamaan. Muuten palautetaan virhe sovellusohjelmalle.

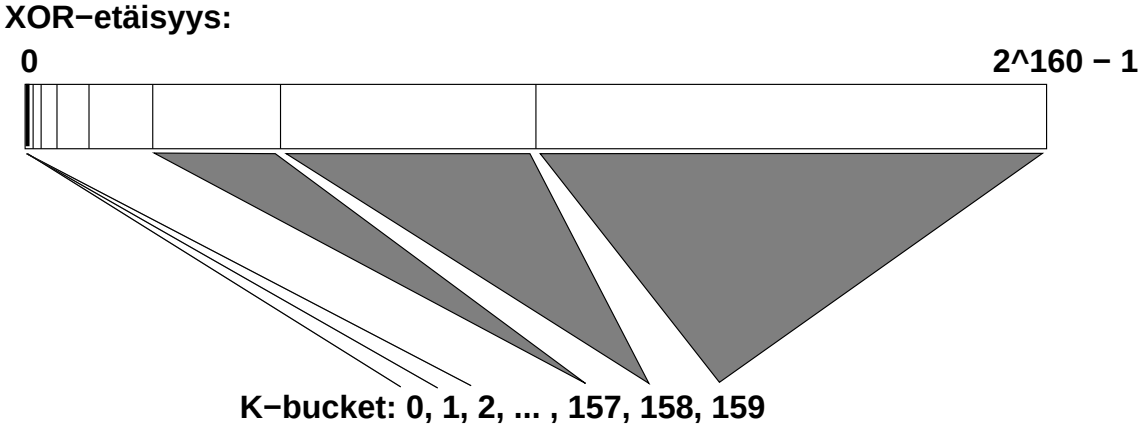
4. Koska tätä paluuarvoa ei pitäisi enää tässävaiheessa tulla, palautetaan virhe sovellusohjelmalle.

5. API palauttaa virheen sovellusohjelmalle.

Mikäli API ei saa vastausta tai tiedonsiirto epäonnistuu, palauttaa API virheen sovellusohjelmalle.

API saa onnistumis- tai epäonnistumisviestin ennen pyynnön kuittausta API siirtyy odottamaan tulosjoukkoa, ja hylkää kuittausviestin, mikäli sellainen ylipäänsä tulee.

API saa tulokset ennen kuittaus-, onnistumis- tai epäonnistumisviestiä API toimii samoin, kuin siinä tilanteessa, että se olisi saanut tulokset k.o. viestin jälkeen.



Kuva 2: XOR-osoiteavaruus ja sen osittaminen K-bucketeihin.

4 DHTn toiminta

Tässä luvussa kuvaillaan tuotettavan DHT-toteutuksen rakenne. Hajautettu hajautustaulu, eli DHT, toteutetaan GNUnet-ympäristöön mukautetulla Kademlia-algoritmilla. Kademlia-algoritmin keskeinen idea on reititys XOR-metriikan avulla. XOR-metriikassa kahden luvun etäisyys on luku, jonka binääriesitys on muodostettu XOR-operaatiolla tarkasteltavien lukujen binääriesityksistä. XOR-metrinen reititys takaa minkä tahansa verkossa olevan solmun löytymisen logaritmisessa ajassa verkon alkioiden määrään nähden.

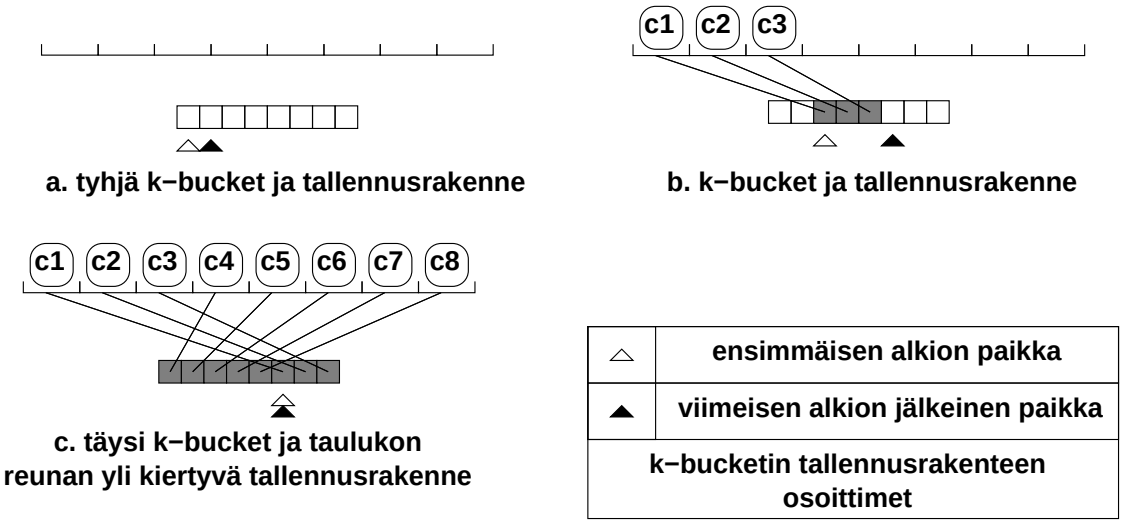
4.1 XOR-metriikka

Solmun K-Bucket -indeksi määrittyy sen mukaan, mikä on solmun ID:n ja lokaalin solmun ID:n XOR-arvon suurin (eniten merkitsevä) päällä oleva bitti. Mitä suurempi bitti on päällä, sitä kauempana solmu XOR-metriikan mukaan on paikallisesta solmusta. K-Bucketin indeksiä suoraan merkitsevän bitin paikan palauttaa funktio `LARGEST_BIT`

4.2 K-Bucketit

K-Bucketit ovat tietorakenteita, jotka tallentavat tietoa enimmillään K:sta solmusta. Tallennettujen solmujen XOR-etäisyydet paikallisesta solmusta ovat välillä $[2^i, 2^(i+1) - 1]$ [Kademlia 2.2, artikkelin tekstissä väli on $[2^i, 2^(i+1)]$, mistä seuraisi yhden arvon päällekkäisyys vierekkäisillä intervalleilla].

K-Bucketeissa olevat solmut säilytetään järjestyksessä kauimmin aikaa sitten nähdystä viimeksi nähtyyn. K-Bucket ylläpitää solmuja järjestyksessä kauimmin aikaa sitten nähdystä viimeksi nähtyyn. Alkioita voidaan poistaa mistä tahansa kohtaa listaa, ja niitä lisätään listan perään. Tietorakenteen voi toteuttaa linkitettynä listana. Koska alkioita ei koskaan ole enempää kuin K kappaletta, voidaan käyttää myös taulukkorakennetta, jossa alkiojono voi alkaa ja päättyä mielivaltaisen indeksin kohdalla, kiertyen tarvittaessa taulukon lopusta takaisin alkuun. Tässä toteutuksessa alkioita voidaan lisätä ja poistaa jo-



Kuva 3: Kontaktitietojen tallennus K-bucketissa.

non alusta ja lopusta vakioajassa, jonon keskeltä poistaminen vaatii ajan O(n), mutta ei välttämättä tuota ongelmia, koska $n \leq k$, eli se ei ole kovin suuri.

4.2.1 `void updateKBucket(DHT_NodeID spottedNode)`

Kutsutaan aina kun solmu tulee tietoiseksi jostain toisesta solmusta. K-Bucket pidetään järjestyksessä kauimmin aikaa sitten nähdystä solmusta viimeksi nähtyyn solmuun.

```
UPDATE_K_BUCKET(NODE):
    TARGET_ID := NODE.ID
    BUCKET_ID := LARGEST_BIT(XOR(NODE_ID, TARGET_ID))
    IF NODE NOT IN K_BUCKET[BUCKET_ID]:
        IF K_BUCKET[BUCKET_ID] is not full:
            add NODE to K_BUCKET[BUCKET_ID]
        ELSE IF least recently seen node of K_BUCKET[BUCKET_ID]
        (the first node in the bucket) responds to ping:
            move least recently seen node of K_BUCKET[BUCKET_ID]
            to the tail of the bucket, abandon NODE
        ELSE:
            remove the least recently seen node, add NODE
            to the tail of the bucket
```

4.3 Solmujen määrä avaimen ja paikallisen ID:n välissä

Jotta tiedettäisiin, kuuluuko jonkin solmun säilyttää tietty <avain, arvo>-pari pysyvästi, halutaan tietää onko XOR-metrisessä osoiteavaruudessa <avain, arvo>-parin ja ky-

seisen solmun välissä vähemmän kuin k solmua. Jos välissä on vähemmän kuin k solmua, solmun kuuluu säilyttää pari, muuten solmu ainoastaan cachettaa parin määräajaksi. Kademlia-algoritmissa cachetusajan määrittäminenkin tapahtuu välissä olevien solmujen määrän perusteella [Kademlia 2.3], joten määrän palauttava funktio on hyödyllinen. Määrä olisi myös hyvä pystyä määrittelemään paikallisen verkkotuntemuksen perusteella, niin ettei verkkoliikennettä tarvitsisi tuottaa enempää. Koska solmut eivät tunne koko verkkoa, ei tarkan solmumäärän määrittäminen ole mahdollista, ainakaan mikäli välissä on paljon solmuja. Kysymykseen, onko välissä alle vai yli k solmua voidaan kuitenkin vastata varsin tarkasti.

Solmujen k-bucketit on indeksoitu arvoilla $0 \leq i < 160$. Jokainen k-bucket i sisältää sellaisia solmuja, joiden ID:n ensimmäiset $160 - i - 1$ bittiä ovat samat kuin paikallisen solmun ID:ssä ja bitti $160 - i$ on eri kuin paikallisen solmun ID:ssä. Jos avain-ID sijoittuu paikallisen solmun k-buckettiin i, voidaan avain-ID:n ja paikallisen solmun suhteista paikallisen k-bucket -rakenteen perusteella päätellä seuraavaa:

- Indeksiltään i:tä suurempien k-buckettien sisältö on paikallista ID:tä kauempana avain-ID:stä.

- Koska k-bucketissa i olevien solmujen ID:n bitti $160 - i$ on sama kuin avain-ID:llä ja eri kuin paikallisella ID:llä, kaikki k-bucketin solmut ovat paikallista solmua lähempänä avain-ID:tä.

- Indeksiltään i:tä pienempien k-buckettien sisältämät solmut ovat kaikki paikallista solmua lähempänä avain-ID:tä, jos niiden ID:n ensimmäinen paikallisesta ID:stä eroava bitti on sama kuin avain-ID:ssä, muussa tapauksessa ne ovat kaikki paikallista solmua kauempana avain-ID:stä

Kolmannen tapauksen perustelu on seuraava: Tarkastellaan ensimmäistä bittiä, joka eroaa paikallisessa ID:ssä ja k-bucketissa olevien alkoiden ID:issä. Tällä bitillähän on kaikkien k-bucketin alkioiden ID:issä eri arvo kuin paikallisessa ID:ssä. Jos tämä bitti on sama paikallisessa ID:ssä ja avain-ID:ssä, kaikki k-bucketin solmut ovat kauempana avaimesta kuin paikallinen solmu, koska tällöin paikallisella ID:llä on vähintään yhtä bittiä pidempi yhteinen alku avain-ID:n kanssa. Jos taas tarkasteltava bitti saa paikallisessa ID:ssä ja avain-ID:ssä eri arvot, ovat kaikki k-bucketin solmut paikallista solmua lähempänä avain-ID:tä, koska niillä on vastaavasti yhtä bittiä paikallista solmua pidempi yhteinen ID:n alku avain-ID:n kanssa.

Tiettyä avainta paikallista solmua lähempänä olevien ja paikallisesti tunnettujen solmujen kokonaismäärä voidaan siis laskea seuraavalla algoritmilla:

4.3.1 `int minNodesBetweenSelfAndKey(DHT_NodeID key)`

```
KEY_BUCKET_ID := LARGEST_BIT(XOR(NODE_ID, KEY))
RESULT := number of values in K_BUCKET[KEY_BUCKET_ID]
```

```
FOR I = 0 to KEY_BUCKET_ID - 1:
    IF BITWISE_AND(NODE_ID, 2^I) != BITWISE_AND(KEY, 2^I):
        RESULT := RESULT + number of values in K_BUCKET[I]
RETURN RESULT
```

4.3.2 Analyysiä

Solmun k-bucketit ovat pienillä indekseillä käytännössä aina tyhjiä, ja alkavat täyttyä kun indeksi kasvaa. Jos solmun ja avaimen välissä olevien solmujen lukumäärä näyttää k-bucket -listan perusteella olevan samaa tai pienempää suuruusluokkaa kuin k, lukumäärä on todennäköisesti oikea tai lähellä oikeaa todellisen verkonkin suhtee. Kun tulos kasvaa, se jää jälkeen todellisesta luvusta. Jos todellinen välisolmujen määrä on n, suurilla n:n arvoilla paikallisen tiedon perusteella saatava lukumäärä lähestyy arvoa $k \log n$.

Mikäli avain-ID:n sisältävä k-bucket on täynnä, nähdään heti, että avainta lähempänä on vähintään k solmua, eikä avaimen säilöminen kuulu paikalliselle solmulle. Suurilla indekseillä k-bucketit ovat luultavasti täynnä, joten kaukana olevien avainten kuulumattomuus paikallisen solmun säilöttäväksi on yleensä helppoa tarkistaa.

4.4 RPC-kutsut ja niihin liittyvät funktiot

Varsinaiset RPC-kutsut eivät tee paljoakaan työtä, varsinainen solmujen välinen operointi hoidetaan erillisissä lookup- tms. funktioissa [Kademlia 2.3].

4.4.1 FIND_NODE (rpc)

```
FIND_NODE(TARGET_ID):
    BUCKET_ID := LARGEST_BIT(XOR(NODE_ID, TARGET_ID))
    RESULT := contents of K_BUCKET[BUCKET_ID]
    IF SIZE(RESULT) < K:
        FOR I = BUCKET_ID - 1 TO 0, BUCKET_ID + 1 TO 159:
            RESULT := union of RESULT and
            K - SIZE(RESULT) items from K_BUCKET[I]
            IF SIZE(RESULT) == K:
                BREAK
    RETURN RESULT
```

4.4.2 DHT_NodeSet nodeLookup(DHT_NodeID targetID)

```
NODE_LOOKUP(TARGET_ID):
    QUERIED_NODES := empty set
    NODE_SET := ALPHA closest known nodes
```

```
    WHILE NODE_SET contains nodes not in QUERIED_NODES:
        RPC_RESULT := empty_set
        concurrent: FOR I = 1 TO ALPHA:
            NODE := A randon node from NODE_SET
            not in QUERIED_NODES
            QUERIED_NODES := QUERIED_NODES union {NODE}
            call NODE.FIND_NODE(TARGET_ID), add result
            to RPC_RESULT
        (if NODE fails to respond to the RPC call, remove
        it from NODE_SET and continue)
        NODE_SET := K nodes closest to TARGET_ID from
        NODE_SET union RPC_RESULT
    RETURN K nodes from QUERIED_NODES closest to TARGET_ID
```

4.4.3 FIND_VALUE (rpc)

```
FIND_VALUE(KEY):
    IF this node contains the desired <KEY, VALUE> pair:
        RETURN VALUE
    ELSE:
        TARGET_ID := KEY
        RETURN FIND_NODE(TARGET_ID)
```

4.4.4 DHT_ResultSet valueLookup(DHT_Key key)

```
VALUE_LOOKUP(KEY):
    TARGET_ID = HASHFUNCTION(KEY)

    QUERIED_NODES := empty set
    NODE_SET := ALPHA closest known nodes
    VALUE := NULL
    WHILE NODE_SET contains nodes not in QUERIED_NODES
    AND RESULT = NULL:
        RPC_RESULT := empty_set
        concurrent: FOR I = 1 TO ALPHA:
            NODE := A randon node from NODE_SET
            not in QUERIED_NODES
            QUERIED_NODES := QUERIED_NODES union {NODE}
            call NODE.FIND_VALUE(TARGET_ID),
            add result to RPC_RESULT
            if FIND_VALUE returns the actual value
            instead of a node set, store this in VALUE
        (if NODE fails to respond to the RPC call, remove
        it from NODE_SET and continue)
```

```
        NODE_SET := K nodes closest to TARGET_ID from
        NODE_SET union RPC_RESULT
    IF VALUE NOT NULL:
        store <KEY, VALUE> into closest node in QUERIED_NODES
        to TARGET_ID which did not contain <KEY, VALUE>
        RETURN VALUE
    ELSE:
        RETURN NULL
```

4.4.5 STORE (rpc)

RPC-kutsu STORE tallettaa <avain, arvo> -parin paikalliseen solmuun. Algoritmisesti toteutus on triviaali.

4.4.6 storeKValues(DHT_DataStoreUnit keyValue)

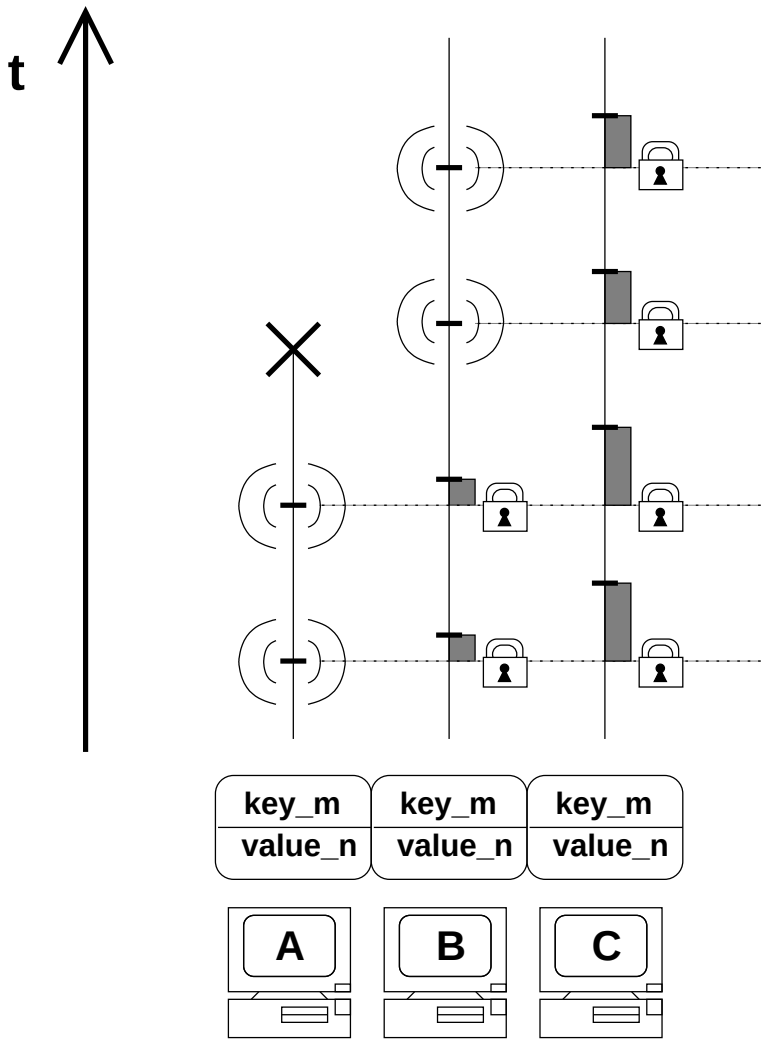
Talletetaan <avain, arvo> -pari K avaimen hashia lähinnä olevaan solmuun.

```
STORE_K_VALUES(KEY, VALUE):
    TARGET_ID := HASHFUNCTION(key)
    NODES := NODE_LOOKUP(TARGET_ID)
    IF NODES is empty:
        ERROR
    ELSE: FOR NODE IN NODES:
        call NODE.STORE(KEY, VALUE)
```

4.5 Tietojen uudelleenjulkaisu

<Avain, arvo>-parit täytyy julkaista uudelleen säännöllisesti, jotta arvot pysyisivät verkossa vaikka verkosta poistuu ja siihen liittyy solmuja. Kademlia-artikkelissa esitetään, että solmut tulisi julkaista tunnin välein [Kademlia 2.5]. Naiivissa toteutuksessa jokainen tiedosta vastaava solmu uudelleenjulkaisisi tiedon tunnin välein, ja eli saman tunnin aikana samat tiedot uudelleenjulkaisisi k solmua. Verkkoliikenteen vähentämiseksi DHT-algoritmi käyttää `doNotRepublish`-bittiä.

Aina kun solmu vastaanottaa STORE- RPC-kutsun, se asettaa talletetun <avain, arvo>-parin kohdalla `doNotRepublish`-bitin päälle. Kun solmu alkaa uudelleenjulkaista tietojaan, se uudelleenjulkaisee vain ne <avain, arvo>-parit, joiden kohdalla `doNotRepublish`-bitti ei ole päällä. Jos bitti on päällä, se asetetaan pois päältä. Nyt tarpeettoman moni solmu uudelleenjulkaisee tietoja vain siinä tapauksessa, että solmujen uudelleenjulkaisuajat ovat hyvin lähekkäin. Muuten ensimmäisena uudelleenjulkaisemaan ehtivä solmu saa muiden tiedot omistavien solmujen `doNotRepublish`-bitit päälle, eivätkä nämä enää julkaise samaa tietoa.



Kuva 4: Tietojen uudelleenjulkaisu ja `doNotRepublish`-bitti. Koneet A, B ja C vastaavat kaikki samasta <avain, arvo>-parista. A ehtii uudelleenjulkaista parin ensin, ja kytkee päälle B:n ja C:n `doNotRepublish`-bitit. Myöhemmin A poistuu verkosta, jonka jälkeen B julkaisee tiedot uudelleen. B:täkin myöhemmin aktivoituva C ei edelleenkään julkaise tietoja.

5 Etäproseduurikutsujen toteutus

Etäproseduurikutsut (RPC) tapahtuvat siten, että lähdesolmu ensin lähettää kutsun kohdesolmuun. Kun kohdesolmu saa kutsun, suorittaa se siihen liittyvän toiminnallisuuden ja palauttaa vastauksen kohdesolmulle. Ottamatta kantaa tarkkaan toteutukseen, sisältää kutsu seuraavaa informaatiota:

- Etäproseduurikutsun nimi (esim. "PING"tai "STORE")
- Kutsukerran tunniste (lineaarisesti kasvava järjestysnumero). Riittää, että tunniste on yksilöllinen kutsuja lähettävälle solmulle.
- Kutsun parametrit <nimi,arvo>-pareina

Vastaus sisältää vastaavasti seuraavaa informaatiota:

- Kutsukerran tunniste (jonka lähettäjä siis on generoinut)
- Paluun parametrit <nimi,arvo>-pareina

5.1 GNUnetin tarjoamat palvelut

RPC-toteutus käyttää GNUnetin CoreAPIForApplication-rajapintaa viestien lähettämiseen kohdesolmuun. Viestien koolla on maksimiraja, joka riippuu siitä, millä kuljetuskerroksen protokollalla viestit välitetään. GNUnetin tukemat kuljetuskerroksen protokollat ovat TCP, UDP, HTTP ja SMTP, ja jokaisen GNUnetd:n tulee tukea ainakin yhtä näistä. DHT:n suhteen on kuitenkin lähdetty siitä, että käytettäisiin yksiselitteisesti UDP:tä tehokkuussyistä.

Riippumatta käytetystä kuljetuskerroksen protokollasta, GNUnet ei takaa, että viestit toimitetaan perille. GNUnetin vastuualuetta tietoliikenteessä on lähinnä salaus, muu jää sovelluksen, eli tässä tapauksessa DHT-moduulin huoleksi. Nämä muut vastuualueet ovat seuraavia:

- Siirrettävän datan pilkkominen ja kokoon kasaaminen
  - Tosin GNUnet saattaa yhdistää useita viestejä yhtään lähetettävään kuljetuskerroksen pakettiin.
- Luotettava lähetys - Eli käytännössä lähetettävien viestien ajastaminen, kuittaus ja tarvittaessa uudelleenlähetys
- Ruuhkanhallinta

Jos haetaan yhtymäkohtia yleisiin kuljetuskerroksen toteutuksiin, seuraavat osa-alueet erityisesti EIVÄT ole DHT-moduulin vastuualueita:

- Yhteyden perustaminen (GNUnet hoitaa APIn kautta pyytämällä - HELO-PING-PONG-SKEY-PING-PONG -viestirimpsu)
- Duplikaattien karsiminen (Kuljetuskerrokset tarjoavat GNUnetille)
- Virheellisten lähetysten karsiminen (Kuljetuskerrokset/GNUnet hoitavat)

Luotettava lähetys on toteutettava, mikäli DHT-moduulille tarjottavasta RPC-rajapinnasta halutaan semantiikaltaan luotettava. Suunnittelussa on lähdetty siitä olettamuksesta, että näin tulee olla.

GNUnet hoitaa ruuhkanhallintaa ainoastaan sillä tasolla, että se rajoittaa kaistankulutuksen käyttäjän asettamiin rajoihin. Käytännössä GNUnet-sovelluksen hoitavat ruuhkanhallintaa siten, että ne antavat GNUnet-ytimelle antamilleen viesteille prioriteetin ja maksimiodotusajan, jolloin tärkeät tai kiireiset viestit ”ehkä” pääsevät läpi, vaikka kaista olisikin ruuhkautunut. Kovin hienovaraista ruuhkanhallintaa ei liene mahdollista toteuttaa, koska viestipuskureiden purkaminen verkkoon on viime kädessä GNUnet-ytimen vastuualuetta, tosin käyttäjän viesteihin liittämien parametrien osin ohjaamana.

5.2 Pilkkominen ja kokoon kasaaminen

Koska GNUnet siirtää dataa viesteissä, joiden maksimikoko on hyvin rajallinen, ja toisaalta etäproseduurikutsuissa kutsun ja vastauksen sisältämän informaation määrä on mielivaltainen, täytyy siirrettävää informaatiota mahdollisesti pilkkoa. Etäproseduurien kutsuista ja paluista koostetaan paketti, joka pilkotaan joukoksi segmenttejä, jotka sitten lähetetään yksitellen GNUnet-viesteihin kapseloituna verkon yli.

Kuten yleensäkin tietoliikenteessä, kaikki kokonaislukuarvot välitetään ns. Network Byte Order -muodossa, eli eniten merkitsevä tavu ensimmäisenä (big-endian).

5.2.1 GNUnet-viestit

Toimitaan niin, että DHT:n tarpeita varten määritellään neljä uutta GNUnet-viestityyppiä. Määritellyt viestityypit sidotaan kokonaislukuarvoihin, jotka määritellään GNUnetin yleisessä otsaketiedostossa include/gnunet_util.h. Uudet viestityypit ovat seuraavat:

- REQ - Etäproseduurin kutsuun liittyvää dataa välittävä viesti (kutsun lähdesolmusta kohdesolmuun)
- RES - Etäproseduurin paluuseen liittyvää dataa välittävä viesti (kutsun kohdesolmusta lähdesolmuun)
- ACK - REQin tai RESin kuittaus

Jos tehdään sellainen yksinkertaistettu oletus, että kahden DHT-solmun välillä toteutetaan yksi RPC-kutsu (siis ei useita rinnakkain), on viestiliikenne sen kaltaista, että ensin

kutsuja lähettää joukon REQ-viestejä jotka kuitataan ACK-viesteillä. Kun kaikki REQ-viestit on vastaanotettu, toteuttaa kohdesolmu vastaavat toiminnallisuuden ja lähettää joukon RES-viestejä, jotka kuitataan ACK-viesteillä. Ns. ”piggy-backing” -tekniikasta ei siis ole varsinaista hyötyä, sillä "dataviestejä"(RES ja REQ) ei siirrettä samanaikaisesti molempiin suuntiin.

Käytännössä useita RPC-kutsuja saatetaan tehdä samojen solmujen välillä rinnakkain, mutta yksinkertaisuuden nimissä ”piggy-backing” -tekniikkaa ei toteuteta. Ainoa tapaus, milloin kuittausta ei lähetetä on se, kun RES-viestit valmistuvat lähetettäväksi ennen kuin kaikki edeltäneet REQ-viestit on ehditty kuitata, jolloin ”mikä tahansa RES-viesti implisiittisesti kuittaa kaikki vastaanotetut REQ-viestit”.

REQ-viesti REQ-viesti välittää etäproseduurin kutsun lähdesolmusta kohdesolmuun. Kutsu välitetään yhdessä tai useammassa REQ-viestissä, jonka omat kentät ovat järjestyksessä seuraavat:

- Järjestysnumero (16-bittinen etumerkitön kokonaisluku)
- Aikaleima (32-bittinen etumerkitön kokonaisluku, lähetyshetken ajankohta millisekunneissa kuluvassa olevan tunnin alusta mitattuna)
- Etäproseduurin kutsukerran yksilöllinen tunniste (32-bittinen etumerkitön kokonaisluku, esim. kasvava järjestysnumero)
- Segmentin järjestysnumero (32-bittinen etumerkitön kokonaisluku)
- Joukon segmenttien kokonaismäärä (32-bittinen etumerkitön kokonaisluku)
- Datakenttä, eli yksi pala pilkotusta paketista. Pituudeltaan niin suuri kuin mitä mahtuu GNUnetin viestiin, viimeisen segmentin tapauksessa vajaapituinen.

RES-viesti RES-viesti välittää etäproseduurin paluun kohdesolmusta lähdesolmuun. RES-viestin kentät ovat samat kuin REQ-viestissä, eli looginen ero on viestin tyypissä itsessään. Kentät ovat siis seuraavat:

- Järjestysnumero (16-bittinen etumerkitön kokonaisluku)
- Aikaleima (32-bittinen etumerkitön kokonaisluku, lähetyshetken ajankohta millisekunneissa kuluvassa olevan tunnin alusta mitattuna)
- Etäproseduurin kutsukerran yksilöllinen tunniste (32-bittinen etumerkitön kokonaisluku, esim. kasvava järjestysnumero)
- Segmentin järjestysnumero (32-bittinen etumerkitön kokonaisluku)
- Joukon segmenttien kokonaismäärä (32-bittinen etumerkitön kokonaisluku)
- Datakenttä, eli yksi pala pilkotusta paketista. Pituudeltaan niin suuri kuin mitä mahtuu GNUnetin viestiin, viimeisen segmentin tapauksessa vajaapituinen.

ACK-viesti ACK-viesti kuittaa yhden tai useamman REQ- tai RES-viestin. Sen kentät ovat järjestyksessä seuraavat:

- Kuitattavien viestien määrä (16-bittinen etumerkitön kokonaisluku)
- Jokaista kuitattavaa viestiä kohden seuraava kenttä:
  - Kuitattavan viestin järjestysnumero

5.2.2 Kutsu- ja paluupaketit

Itse etäproseduurikutsun suorittamiseen tarvittava data kootaan kutsupaketiksi, joka välitetään yhtenä tai useampana segmenttinä datakentässä, yhteen tai useampaan REQ-viestiin kapseloituna. Vastaavasti etäproseduurikutsun paluuarvot kootaan paluupaketiksi, joka välitetään yhdessä tai useammassa RES-viestissä.

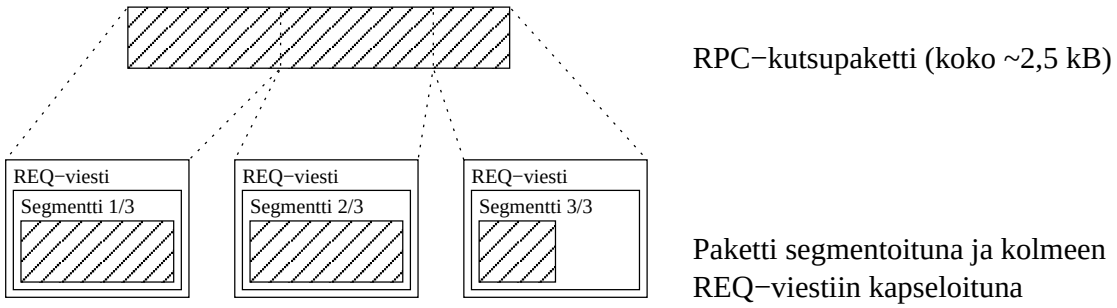
Kutsupaketti sisältää seuraavat kentät seuraavassa järjestyksessä:

- Etäproseduurikutsun nimi (NUL-terminoitu merkkijono)
- Parametrien määrä (8-bittinen etumerkitön kokonaisluku)
- Parametrit, joista jokaisesta seuraavat kentät:
  - Parametrin nimi (NUL-terminoitu merkkijono)
  - Parametrin arvon pituus (32-bittinen etumerkitön kokonaisluku)
  - Parametrin arvo (Mielivaltaista dataa, pituus edellisen parametrin osoittama)

Vastauspaketti sisältää seuraavat kentät seuraavassa järjestyksessä:

- Parametrien määrä (8-bittinen etumerkitön kokonaisluku)
- Parametrit, joista jokaisesta seuraavat kentät:
  - Parametrin nimi (NUL-terminoitu merkkijono)
  - Parametrin arvon pituus (32-bittinen etumerkitön kokonaisluku)
  - Parametrin arvo (Mielivaltaista dataa, pituus edellisen parametrin osoittama)

Oheisessa kuvassa on esitetty REQ-paketin segmentointi ja segmenttien kapselointi GNUnet-viesteiksi käsitteellisellä tasolla. "Segmentti"on siis fyysisesti osa GNUnet viestin otsakkeen tietoja.



Kuva 5: REQ-paketin pilkkominen segmenteiksi ja GNUnet-viesteihin kapselointi

5.2.3 Viestien käsittelystä

Segmentit sisältävät siis kaiken sen informaation, minkä perusteella vastaanottaja voi pistää segmentin talteen odottamaan sitä, että kaikki samaan pakettiin (joko kutsu tai vastaus) liittyvät segmentit on saatu. Kun kaikki paketin segmentit on saatu, koostetaan paketti ja joko suoritetaan etäproseduurikutsuun kytketty toiminto (kutsu) tai palautetaan paluuarvot kutsun tekijälle (paluu).

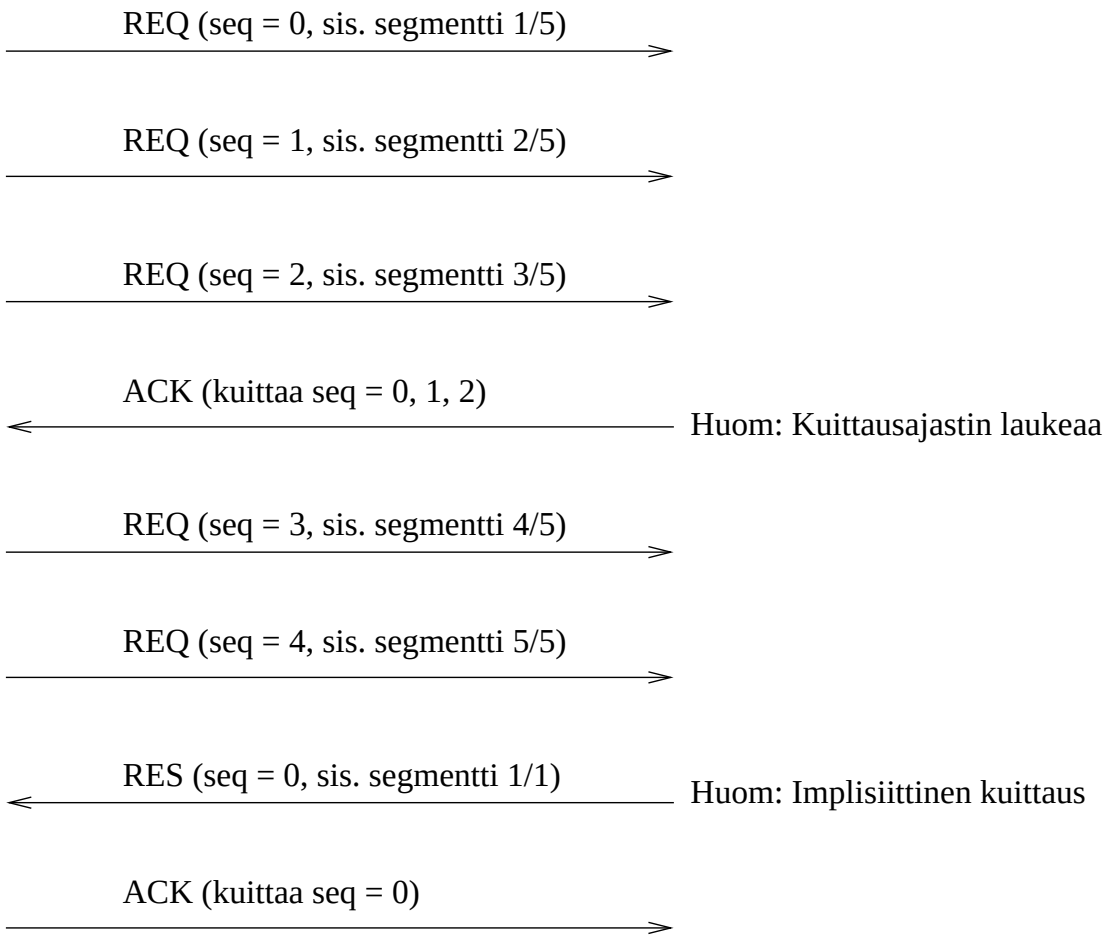
Kutsukerran yksilöllinen tunniste tarvitaan, jotta ei tarvitse asettaa rajoitteita sen suhteen, miten rinnakkaisesti suoritetaan etäproseduurikutsuja samasta solmusta A samaan solmuun B. Lisäksi jos kutsun kuittaus epäonnistuu, tajuaa kohdesolmu olla suorittamatta uudestaan etäproseduurikutsua, jos se uudelleenlähetetään.

5.3 Kuittaukset ja uudelleenlähetys

Ongelma on lyhyesti se, että vaikka GNUnet ”saattaa” lähettää viestit luotettavaa kanavaa pitkin, saattaa GNUnet itse tiputtaa viestejä lattialle sikäli kun sitä itseään huvittaa niin tehdä. Jos RPC:istä halutaan tehdä "varmoja", täytyy segmenttejä kuittailla ja tarvittaessa uudelleenlähettää. Problematiikka on siis samankaltaista kuin mitä esim. TCP-protokollan toteutuksessa on jouduttu huomioimaan, ja siis tarpeeksi jalostettuna huomattavan epätriviaalia.

Lähtökohta on se, että kaikki REQ- ja RES-viestit kuitataan tavalla tai toisella. Jos kuittausta ei saada aikarajan sisällä, lähetetään viesti uudestaan. Jos kuittaus katoaa seuraa turha uudelleenlähetys, jonka vastaanottaja helposti havaitsee segmentin kenttien ja oman kirjanpitonsa perusteella. Jos kuittaus saapuu myöhässä (uudelleenlähetyksen jälkeen), indikoinee se sitä, että ajastin oli liian tiukka, ja että sen pituutta tulisi ehkä kasvattaa. Kuittauksiin liittyy olennaisesti sellainen havainto, että mikä tahansa RES-viesti implisiittisesti kuittaa kaikki REQ-viestit.

Oheisessa kuvassa on esitetty yhteen etäproseduurikutsuun liittyvä viestiliikenne, kun kutsu välitetään viidellä REQ-viestillä ja paluu yhdellä RES-viestillä. Kutsun kolme ensimmäistä viestiä kuitataan ajastimen lauetessa yhdellä ACK-viestillä ja viimeiset kaksi viestiä kuitataan implisiittisesti RES-viestillä. Lisäksi RES-viesti kuitataan ACK-viestillä ajastimen lauetessa. Kuvassa ei ole esitetty GNUnet-ytimen määrittelemillä viesteillä ta-



Kuva 6: Esimerkki etäproseduurikutsuun liittyvästä viestiliikenteestä

pahtuvaa yhteyden muodostamista (tai sen purkamista).

Jos samaistetaan etäproseduurikutsut TCP-virtoihin, on kenties havaittavissa sellainen ero, että etäproseduurikutsuihin liittyvä tietomäärä saattaa olla keskimäärin hyvin pieni, mutta kutsujen määrä suuri. Tämä seikka otetaan huomioon vuonhallinnan ja ruuhkanhallinnan toteutuksessa, että vuon- ja ruuhkanhallinta toteutetaan kohdesolmukohtaisesti, eikä esim. etäproseduurikutsukohtaisesti. Tällöin RPC-moduulin tehtäväksi tulee miettiä myös sitä, kuinka samaan kohdesolmuun kohdistuvat samanaikaiset etäproseduurikutsut lomitetaan, kun näiden viestejä on valmiita lähetettäväksi. Lomitus tapahtuu yksinkertaisella ”round-robin” -menetelmällä, eli jos samaan kohdesolmuun on lähetysvalmiina useisiin eri kutsuihin liittyviä viestejä, lähetetään vuorotellen yksi kutakin kutsua kohden.

5.3.1 Vuonhallinta

Vuonhallinnan malli on liukuvan ikkunan protokolla (ikkunan koko n) olennaisesti siten, kuin se kirjallisuudessa esitetään (esim. Tanenbaum, kappale 3.4.3. "Sliding window protocols - A protocol with selective repeat"). Vuonhallinta tapahtuu siis seuraavasti:

- Lähetettävät REQ- ja RES-viestit on järjestysnumeroitu nollasta alkaen. Lähdesolmun suorittama viestien järjestysnumerointi on kohdesolmukohtaista.

- Lähettäjällä on lähetysikkuna, jolla on tarkkalleen määritelty koko. Kun lähetysikkunan ensimmäinen viesti on kuitattu, poistetaan se ikkunasta ja ikkunan loppupäähän tulee tilaa uudelle viestille. Koska kuittaukset saattavat tulla epäjärjestyksessä, saattaa ikkunan "kiertäminen"tapahtua purskeittain.

- Jokaiselle lähetetylle viestille asetetaan uudelleenlähetysajastin (millisekunneissa), jonka lauetessa viesti lähetetään uudestaan. Ajastin puretaan kun viesti kuitataan onnistuneesti.

  – Uudelleenlähetysajastimen sopivaa arvoa seurataan aikaleimaamalla lähetettävät viestit. RTT:tä kaavalla RTT = alpha RTT + (1 - alpha) M, missä M on juuri saapuneen kuittauksen viive ja alpha vakio välillä 0-1. Tyypillinen arvo alphalle on 7/8, ja tämä on määritelty DHT-moduulin asetuksissa.

- Viestejä vastaanotettaessa asetetaan vastaanotetuille viesteille kuittausajastin, jonka lauetessa kuitataan kaikki kuittaamatta olevat viestit. Erityisesti jos kyseessä on kuittaamattomia REQ-viestejä ja kutsun paluupaketti valmistuu, kuitataan vielä kuittaamattomat REQ-viestit implisiittisesti lähettämällä vain kutsuun liittyvä (ensimmäinen) RES-viesti.

  – Kuittausajastimen arvon tulee olla "hyvin pieni"verrattuna uudelleenlähetysajastimeen, tai muuten seurauksena on helposti mittavia uudelleenlähetyksiä ja RTT:n heilahteluita.

Vuonhallinta tapahtuu kohdesolmukohtaisesti, eli jokainen liukuvan ikkunan protokollan sovellus yhtä kohdesolmua kohden tapahtuu mahdollisesti useamman säikeen toimesta. Käytännössä jokaista RPC-kutsua toteuttava säie pilkkoo kutsunsa ja siirtää sitä vastaavat segmentit kutsukohtaiseen lähetyspuskuriin. Mikäli lähetysikkunassa on tilaa, siirretään segmenttejä välittömästi viesteinä GNUnet-ytimelle, muuten ne jäävät odottamaan puskuriin. Tämän jälkeen kutsua suorittava säie jää odottamaan omaa signalointisemaforiaan.

Kuittaus (tai kuittaava RES-viesti) vastaanotettaessa ikkunaan saattaa tulla tilaa, jolloin tässä callbackia suorittavassa säikeessä saatetaan siirtää uusia viestejä verkkoon. RES-viesti vastaanotettaessa siirretään sen sisältämä segmentti kutsukohtaiseen paluupuskuriin. Mikäli puskuri lisäyksen jälkeen sisältää kaikki paluuseen liittyvät segmentit, herätetään kutsun tehnyt odottava säie.

REQ-viestejä vastaanotettaessa siirretään niiden sisältämät segmentit kutsukohtaiseen puskuriin, ja mikäli puskuri sisältää kaikki kutsuun liittyvät segmentit, suoritetaan kutsu paikallisesti. Paluuseen liittyvät segmentit siirretään heti verkkoon, mikäli lähetysikkunassa on tilaa, tai muuten ne sijoitetaan kutsukohtaiseen puskuriin. Lähtöä odottavat RES-viestit lomitetaan tasa-arvoisesti lähtöä odottavien REQ-viestien kanssa.

5.3.2 Ruuhkanhallinta

Ruuhkanhallinta tapahtuu kohdesolmukohtaisesti. Ruuhkanhallinnan toteutukseen on haettu yleiset periaatteet TCP:n toteutuksesta siltä osin, kun niiden ajatellaan olevan käyttökelpoisia. Pohjimmiltaan ongelma on se, että pelkkä kiinteää ikkunakokoa käyttävä liukuva ikkunan protokolla ei toimi tyydyttävästi verkkokerroksen päällä, koska verkkokerroksen tarjoama palvelu on luonteeltaan hyvin epädeterministinen.

- Lähetysikkunan koko on aluksi DHT-moduulin asetuksissa määrätty (pieni) määrä viestejä (esim. 1). ”Raja-arvo” on aluksi DHT-moduulin asetuksissa määrätty määrä viestejä.

- Jos ikkunan koon verran viestejä on lähetetty peräkkäin onnistuneesti, kaksinkertaistetaan ikkunan koko. Mikäli raja-arvo ylittyy ikkunan koko kaksinkertaistamalla, asetetaan ikkunan kooksi raja-arvo ja tämän jälkeen ikkunaa kasvatetaan lineaarisesti.

- Jos viestejä tippuu, asetetaan raja-arvoksi puolet sen hetkisen lähetysikkunan koosta ja ikkunan koko palaa alkuarvoonsa.

5.4 Etäproseduurikutsujen ohjelmointirajapinta

RPC-abstraktio tarjoaa funktiorajapinnan DHT-moduulin käyttöön, jonka tulee peittää GNUnetin käyttämä viestiliikenne kokonaan, jolloin DHT-moduulin muissa osissa voidaan tietoliikennettä lähestyä puhtaasti Kademlian määrittelemien RPC:iden pohjalta.

RPC-moduuli pilkkoo lähtevät RPC:t GNUnetin viesteiksi ja sijoittaa ne oikein parametrisoituna (miten?) GNUnetin puskureihin, ja valvoo, että ne datasegmenttejä välittävät GNUnet-viestit (vrt. kuittausviestit) todella saavuttavat kohteensa. RPC-moduuli pitää siis lähettämistään segmenteistä kirjaa sen suhteen, onko ne kuitattu ja koska nämä tarvittaessa uudelleenlähetetään.

RPC-moduuli rekisteröityy käsittelemään ne GNUnetin välittämät viestityypit, jotka ovat RPC-moduulin käytössä. Vastaanotettavien RPC-viestien kohdalla RPC-moduuli puskuroi vastaanotettuja segmenttejä, kunnes kaikki saapuvan RPC:n segmentit on vastaanotettu, jolloin voidaan suorittaa DHT-moduulin kyseistä RPC:tä varten rekisteröimä käsittelijä. Vastaanotettavat datasegmentit kuitataan ja vastaanotettavien kuittausten kohdalla päivitetään lähetettyihin segmentteihin liittyvää kirjanpitoa.

5.4.1 Funktiot

int rvalue = RPC_Layer.executeRPC (const HostIdentity *receiver, const char *name, const DHT_RPCParam *request_param, DHT_RPCParam *return_param, int timeout)

Funktio suorittaa etäproseduurikutsun toiseen GNUnet-solmuun ja blokkaa kunnes etäproseduuri on joko onnistunut tai lopullisesti epäonnistunut. Kohdesolmun tulee olla paikalliselle solmulle tunnettu. Funktio on säieturvallinen.

- receiver - Kohdesolmu, GNUnetin host ID:llä ilmaistuna, eli 160-bittinen hajautusavain solmun julkisesta avaimesta
- name - Kutsun nimi merkkijonona, esim. "PING"tai "STORE"
- request_param - Kutsuparametrit, tietorakenne joka sisältää joukon <nimi,arvo>-pareja
  - Parametrien nimien ei tule alkaa alaviivalla (’_’), koska näin nimetyt parametrit on varattu RPC-toteutuksen sisäiseen käyttöön
- return_param - Paluuparametrit, tietorakenne johon sijoitetaan joukko <nimi,arvo>-pareja
- timeout - Sekunteina aikaraja suoritukselle (0 = ei rajoitettu) - Toteutetaan jos on aikaa tai jos havaitaan erityistarvetta.

Paluuarvo on jokin seuraavista:

- 0 = onnistunut suoritus - Paluuparametrit löytyvät kutsujan osoittamasta tietorakenteesta.
- 1 = kohdesolmua ei ole tunnettu
- 2 = kohdesolmua ei tavoitettu
- 3 = tietoliikenneongelma
- 4 = kohdesolmu ei ole rekisteröitynyt käsittelemään kutsua
- 5 = suoritukselle asetetun aikarajan ylittyminen

void RPC_Layer.registerRPC (const char *name, void (*callback) (const DHT_RPCParam *, DHT_RPCParam *))

Rekisteröi annetun funktion käsittelemään tietyn etäproseduurikutsun paikallisessa solmussa. Rekisteröitävän funktion tulee olla säieturvallinen. Huomionarvoista on myös se, että RPC-moduuli ei tiedä mitään siitä, mitä ja minkä muotoisia parametreja etproseduurikutsu mahdollisesti tarvitsee, vaan niiden tarkistaminen on käsittelijäfunktion vastuulla. Funktio palaa "heti".

- name - Etäproseduurikutsun nimi, esim. "PING"tai "STORE"
- callback - Osoitin funktioon, joka ottaa parametreinaan kaksi osoitinta DHT_RPC_Param-rakenteisiin

5.4.2 Standardisoidut parametrit

Paluussa käytetään parametria "_STATUS", joka ilmaisee mitä kutsulle kävi kohdesolmussa. Arvo yksi niistä numeroarvoista, joita lopulta palautetaan kutsun tekijälle executeRPC-kutsun paluuarvona. Mielekkäät arvot kohdesolmun palauttamina ovat siis 0, 4 ja 5.

5.4.3 Kontrollin siirtyminen

Kontrolli siirtyy RPC-moduuliin kahdella tavalla. Joko saman GNUnetd:n sisältä joku suorittaa toiseen solmuun (mahdollisesti itseensä) etäproseduurikutsun, eli kutsuu yhtä RPC-moduulin tarjoamaa funktiota, joka blokkaa. Näitä kutsuja voi suorittaa useampi säie samanaikaisesti. Funktiokutsu palaa kun etäproseduurikutsu on yksiselitteisesti joko suoriutunut tai epäonnistunut. ”RPC-moduulin käyttäjän kannalta rajapinta on siis joukko säieturvallisia, blokkaavia funktioita”.

Toinen tapa kontrollin siirtymiselle RPC-moduulin on se, kun GNUnet vastaanottaa viestin, joka on sellaista tyyppiä, mille RPC-moduuli on rekisteröinyt käsittelijän. Kuten yleisesti GNUnetin tapauksessa, näitäkin kutsuja saattaa tapahtua mielivaltaisesti rinnakkain. Näistä käsittelijöistä saattaa myös seurata funktiokutsu RPC-moduulin käyttäjän rekisteröimään jonkin etäproseduurikutsun toteuttavaan funktioon. Nämä kutsut käsittelijöihin tapahtuvat siis eri säikeistä, kuin mistä RPC-moduulin käyttäjä mahdollisesti itse tekee RPC-kutsuja, eli solmusta ulos lähtevät ja solmuun sisään tulevat etäproseduurikutsut saattavat rajapinnassa lomittua mielivaltaisesti.

Tiivistettynä rajapinta siis koostuu seuraavista osista:

- Säieturvallisista funktioista, joilla voidaan tehdä seuraavia asioita:
  - Rekisteröidä jollekin etäproseduurikutsulle käsittelijä (nopea operaatio)
  - Suorittaa toiseen solmuun etäproseduurikutsu (blokkaava hidas operaatio)
- Kutsuista etäproseduurien käsittelijöihin, joita saattaa tapahtua mielivaltaisesti rinnakkain

Lisäksi on mahdollista, että jokin RPC-moduulin funktio suoritetaan ajastettuna toimintona GNUnet-ytimen palvelun avulla.

6 Datakerros

Datakerros on rajapinta, jota kautta DHT-moduuli voi perustaa paikalliseen tallennusmediaan hajautustauluja ja tallentaa niihin <avain,arvo>-pareja (tietoalkioita). Talletusavaimena datakerros käyttää oikeasta avaimesta muodostettua hashia tyypiltään HashCode160. Oikea avain tallennetaan osaksi arvoa. Hajautustauluissa yhdelle avaimelle voi tallentaa useita erilaisia arvoja. Haettaessa tällaisella avaimella datakerros palauttaa listan kaikista avaimeen liittyvistä arvoista. Erilaisten arvojen määrä määritellään taulun määritystiedon valuesPerKey alkiossa. Tietoalkioiden lisäksi datakerrokseen talletetaan jokaisen sinne perustetun taulun yleiset määrtystiedot.

Tauluihin talletettavat tietoalkioit sisältävät seuraavat tiedot:

- hash - avaimesta hashattu 160b tunniste
- timeout - aikaleima, jolloin <avain,arvo>-pari vanhenee
- flags - erinäisiä tietoalkioon liittyviä flageja, näistä tähän toteutukseen määritellään
  - tieto onko tietoalkio solmun paikallisesti tallentama. (IS_OWN)
  - "ei uudelleenjulkaista-bitti. Tieto tuleeko tietoalkio julkaista uudestaan seuraavalla uudelleenjulkaisukierroksella. (NOT_REPUBLISH)
- data - sisältää <avain,arvo>-parin bittijonona

```
struct DHT_DataStroreUnit {
  Hash160b * hash;
  TIME timeout;
  unsigned char flags;
  DHT_DataContainer * data;
}
```

Tallennetuista tietoalkioista tarvitaan tietoa seuraavissa kohdissa:

- haetaan paikallisesta storesta avaimen perusteella
- tallennetaan paikalliseen storeen
- ylläpidetää (tyhjennetään) cachea
- haetaan itsejulkaistut avaimet avaimia
  - uudelleen julkaistamista varten
  - sovellusrajapinnan pyyntöä varten.
- haetaan avaimia tauluun liittyneelle nodelle

Näiden toteuttamista varten datakerroksesta tarvitaan neljää eri hakua:

- kaikki yhteen avaimeen liittyvät tietoalkiot
- timeoutin ylittäneet tietoalkiot
- itse julkaistut tietoalkiot
- tietoalkiot, joiden hash arvon XOR etäisyys on pienempi toisesta id:stä kuin solmun omasta id:stä

Datakerros pitää myös kirjaa kuinka paljon dataa tavuina solmu on tallentanut paikallisesti. Datakerrokselle määritellään asetuksena raja-arvo kuinka paljon tallennuskapasiteettia se voi käyttää. Kun raja-arvo ylittyy uudet tallentamisyritykset palauttavat virheilmoituksen. Datakerroksen operaatioihin onnistumistietoon liittyy neljä tilatietoa:

- OK - operaatio on onnistunut
- DATASTORE_FULL - paikallinen tallennuskapasiteetti on täyttynyt
- KEYSTORE_FULL - yksittäiseen avaimeen liittyvien arvojen määrä ylittää suurimman sallitun
- DATASTORE_ERROR - operaation suorituksessa on tapahtunut muu virhe

Datakerros jakaantuu loogisesti kahteen kerrokseen: hakukerrrokseen ja pysyväiskerrokseen. Tämä on tehty siksi, että tiedon pysyvyyden toteuttaminen voidaan jättää tämän toteutuksen ulkopuolelle, kuitenkin siten että sen toteuttaminen on mahdollista ilman suurempia muutoksia kokonaisuuteen. Kerrokset on jaettu siten että DHT-moduuli käyttää omien algoritmien toteuttamiseen hakukerroksen palveluja. Hakukerros tallentaa tietoalkiot omiin paikallisiin tietorakenteisiin ja pysyväiskerrokseen. Tiedon pysyvyyttä tarvitaan, kun DHT verkon solmu käynnistetään uudestaan.

6.1 Hakurajapinta

Datakerrokseen toteutettu toiminnallisuus tarjotaan sitä käyttävälle DHT-moduulille tässä kappaleessa määritetyn hakurajapinnan kautta.

6.1.1 localStoreListTables

DHT_TableSet localStoreListTables();

Funktio listaa kaikki taulut, jotka on luotu datakerrokseen. Kun DHT-moduuli käynnistyy ajetaan initialisointi, joka rekisteröi kaikki tallentamansa taulut DHT-moduulin tietoon. Palvelu tukeutuu pysyväiskerroksen tarjoamaan listausoperaatioon.

palautusarvo DHT_TableSet - handle settiin tauluja. handle pitää sisällään operaation onnistumis flägin. Kukin setin alkio pitää sisällään DHT_TableMetaDatan, DHT_TableConfigin ja DHT_TableID:n

6.1.2 localStoreCreateTable

int localStoreCreateTable(const DHT_TableID table_id, const DHT_TableMetaDatan metadata, const DHT_TableConfig config);

Funktio localStoreCreateTable luo datakerrokseen uuden taulun, johon voi myöhemmin lisätä <avain, arvo> -pareja. DHT-moduuli kutsuu tätä palvelua kun

1. sovellusrajapintaa käyttävä sovellus haluaa luoda uuden DHT-taulun
2. sovellus haluaa liityttyä uuteen tauluun

parametrit

- table_id - luotavan taulun id DHT:ssä
- metadata - luotavan taulun metatieto
- config - luotavaan tauluun liittyvät määreet

palautusarvo Kokonaisluku johon on koodattu operaation onnistumis/epäonnistumisinformaatio.

toiminta

1. luo persistancelayeriin taulun
2. alustaa hakukerrokseen sen tarvitsemat tietorakenteet

6.1.3 localStorePut

DHT_DataStroreUnit localStorePut(DHT_TableID table_id, DHT_DataStroreUnit * unit_to_store);

Funktio localStorePut tallentaa <avain, arvo> -parin määritettyyn tauluun. DHT-moduuli kutsuu tätä palvelua kun

1. sovellusrajapintaa käyttävä sovellus tallentaa DHT:hen uuden <avain,arvo>-parim
2. DHT-algoritmin value_lookup kutsussa talletetaan cachetettavaksi uusi
3. DHT-algoritmin free_cache tallentaa omat uudestaan julkaistut arvot

Funktio tarkistaa jo tallennetuista tietoalkioista mahdolliset duplikaatit siten, että jos samalle avaimelle tallennetaan sama arvo kirjoitetaan uusi alkio vanhan päälle.

parametrit

- table - pointteri tauluun
- unit_to_store - tietoalkio, johon on laitettu tarvittava aikaleima yms. tiedot. Tietojen määrittäminen siis jää funktion kutsujan huoleksi.

palautusarvo DHT_DataStroreUnit. Tietoalkio, joka talletettiin. Sisältää virhekoodin.

toiminta

1. hakee avaimelle tallennetut tietoalkiot ja käy nämä läpi vertaillen jo talletettujen tietoalkioiden data osiota uuteen tietoalkioon
2. jos samalle avaimelle löytyy sama arvo
   (a) poistaa vanhan tietoalkion persistance layeristä
   (b) poistaa tietoalkion omista tietorakenteista
3. tallentaa tietoalkion persistancelayeriin
4. tallentaa tietoalkion omiin tietorakenteisiin

6.1.4 localStoreDelete

int localStoreDelete(DHT_TableID table_id, DHT_DataStroreUnit unit_to_remove);

Funktio poistaa määritellystä taulusta yhden tietoalkion. DHT-moduuli kutsuu tätä palvelua kun

1. sovellusrajapintaa käyttävä sovellus lopettaa tietyn <avain,arvo>-parin jakamisen
2. DHT-algoritmin free_cache tyhjentää vanhentuneita arvoja datakerroksesta

parametrit

- table_id - taulun tunniste tieto
- unit_to_remove - poistettavan tietoalkion sisältö

palautusarvo Kokonaisluku joka ilmaisee operaation onnistumis/epäonnistumisinformaation.

toiminta

1. hakee kaikki ko. avaimeen liittyvät tietoalkiot
2. käy tietoalkiot läpi vertaillen näiden data osiota uuteen tietoalkioon
3. jos vastaava löydetään
   (a) poistetaan tietoalkio persistencestä
   (b) poistetaan tietoalkio omista tietorakenteista

6.1.5 localStoreGet

DHT_LocalStoreResultSet localStoreGet(DHT_TableID table, HashCode160 key_to_get);

Funktio etsii taulusta kaikki avaimeen liittyvät arvot. Tätä palvelua kutsutaan kun

1. sovellusrajapintaa käyttävä sovellus hakee avainta vastaavia arvoja DHT:stä
2. Kademlian-algoritmin value_lookup hakee avaimelle tallennettuja arvoja omasta taulusta

parametrit

- table - pointteri tauluun
- key_to_get - hashavain, jota vastaavia arvoja etsitään.

palautusarvo DHT_LocalStoreResultSet - handle löydettyihin arvoihin

toiminta

1. Etsii listan key_to_get paramettiä vastaavista arvoista
2. Palauttaa resultsetin joka sisältää tiedon haun onnistumisesta.

6.1.6 localStoreDropTable

int localStoreDropTable(DHT_TableID table_id);

Funktio poistaa taulun datakerroksesta. DHT-moduuli kutsuu tätä palvelua kun kun sovellusrajapintaa käyttävästä sovelluksesta halutaan lopettaa tiettyyn tauluun osallistuminen. Palvelun kutsun jälkeen ko. taulua ei voi enää käyttää paikallisesti.

parametrit

- table_id - poistettavan taulun tunniste

palautusarvo Kokonaisluku joka ilmaisee operaation onnistumis/epäonnistumisinformaation.

toiminta

1. poistaa taulun persistance layeristä
2. vapauttaa tauluun liitetyt omat tietorakenteet

6.1.7 localStoreGetExpired

DHT_LocalStoreResultSet localStoreGetExpired(DHT_TableID table_id, TimeStamp now);

Funktio etsii taulusta kaikki arvot joiden timeout arvo on pienempi kuin annettu aikaleima. DHT-moduuli kutsuu tätä palvelua kun se haluaa etsiä taulun vanhentuneita arvoja.

parametrit

- table_id - taulun tunniste
- now - aikaleima, jota vanhemmat arvot palautetaan.

palautusarvo DHT_LocalStoreResultSet - handle löydettyihin arvoihin ja koodi operaation onnistumisesta.

toiminta

1. Etsii listan lähinnä olevista arvoista.
2. Palauttaa resultsetin joka sisältää tiedon haun onnistumisesta.

6.1.8 localStoreGetCloserTo

DHT_LocalStoreResultSet localStoreGetCloserTo(DHT_TableID table_id, DHT_NodeID myID, DHT_NodeID newID);

Funktio etsii taulusta kaikki arvot lähempänä tiettyä NodeID:tä DHT-moduuli kutsuu tätä palvelua kun se hakee uudelle tauluun liittyneelle solmulle lähimpiä avaimia.

parametrit

- DHT_TableID table_id - taulun tunniste
- DHT_NodeID myID - oman solmun tunniste,
- DHT_NodeID newID - uuden solmun tunniste

palautusarvo DHT_LocalStoreResultSet - handle löydettyihin arvoihin

toiminta

1. hakee talletetuista avaimista ne joiden XOR etäisyys xor(hash,newID) < xor(hash,myID)
2. koostaa avaimiin liittyvistä datoista DHT_LocalStoreResultSetin
3. palautta DHT_LocalStoreResultSetHANDLEN, jossa tiedot operaation onnistumisesta ja pointteri haettuihin arvoihin.

6.1.9 localStoreGetOwnData

DHT_LocalStoreResultSet localStoreGetOwnData(DHT_TableID table_id);

Funktio hakee kaikki tietyyn tauluun itse lisätyt tietoalkiot. Palvelua käytetään kun

1. DHT-moduuli julkaisee kaikki itse julkaistut tietoalkiot
2. sovellusrajapintaa käyttävästä sovelluksesta listataan itse julkaisemat tietoalkiot.

parametrit

- table_id - taulun tunniste

palautusarvo DHT_LocalStoreResultSet - handle löydettyihin arvoihin

toiminta

1. hakee talletetuista datoista ne, joihin on merkattu flag arvo IS_OWN.
2. koostaa avaimiin liittyvistä datoista DHT_LocalStoreResultSetin
3. palautta DHT_LocalStoreResultSetHANDLEN, jossa tiedot operaation onnistumisesta ja pointteri haettuihin arvoihin.

6.1.10 Sisäinen toteutus

Operaatioiden nopeuttamiseksi tarjotaan kolme tietorakennetta:

- lista, jossa tietoalkiot ovat aikaleiman mukaisessa järjestyksessä
- hajautustaulu, jonnea tietoalkiot ovat sijoitettu hash arvonsa mukaiseen paikkaan.
- lista, jossa itse julkaistut tietoalkiot

XOR metriikan luonteen mukaisesti localStoreGetCloserTo funktion toimintaa tukevaa tietorakennetta on hyvin vaikeaa rakentaa, joten se toteutetaan yksinkertaisesti käymällä kaikki talletetut tietoalkiot.

6.2 Rajapinta pysyväiskerrokselle

Pysyväiskerros (persistant layer) tarjoaa mahdollisuuden tallentaa <avain,arvo>-pareja paikalliseen talletusmediaan, siten että ne eivät häviä DHT-solmun sammuttamisen yhteydessä.

6.2.1 persistentCreateTable

int persistentCreateTable(const DHT_TableID id, const DHT_TableMetaDatan metadata, const DHT_TableConfig config);

Tallentaa taulun tiedot talletusmediaan ja luo mahdollisuuden tallentaa siihen dataa ko. taulu id:llä.

parametrit

- DHT_TableID table_id - taulun tunniste
- DHT_TableMetaDatan metadata - taulun metatieto
- DHT_TableConfig config - tauluun liittyvät konfiguraatiot

palautusarvo Kokonaisluku joka ilmaisee operaation onnistumis/epäonnistumisinformaation.

6.2.2 persistentStore

DHT_DataStoreUnit persistentStore(DHT_TableID id, DHT_DataStoreUnit unitToStore);

Tallentaa tallennusmediaan ko tietoalkion siten, että se voidaan myöhemmin hakea sieltä.

parametrit

- DHT_TableID id - taulun tunniste
- DHT_DataStoreUnit - unitToStore arvo joka talletetaan.

palautusarvo DHT_DataStoreUnit - arvo, joka talletettiin

6.2.3 persistentRemove

int persistentRemove(DHT_TableID id, DHT_DataStoreUnit unitToRemove);
Poistaa talletusmediasta ko. tietoyksikön.

parametrit

- DHT_TableID id - taulun tunniste
- DHT_DataStoreUnit - unitToRemove

palautusarvo Kokonaisluku joka ilmaisee operaation onnistumis/epäonnistumisinformaation.

6.2.4 persistentListData

DHT_LocalStoreResultSet persistentListData(DHT_TableID id);
Listaa kaikki yhteen tauluun talletetut yksiköt.

parametrit DHT_TableID id - taulun tunniste

palautusarvo DHT_LocalStoreResultSet - koko taulun sisältö.

6.2.5 persistentListTables

DHT_LocalStoreResultSet persistentListTables();
Listaa kaikki taulut, joita pysyväiskerrokseen on luotu.

6.2.6 persistentDeleteTable

int persistentDeleteTable(DHT_TableID id);
Poistaa yhden pysyväiskerrokseen luodun taulun.

parametrit DHT_TableID id - taulun tunniste

palautusarvo Kokonaisluku joka ilmaisee operaation onnistumis/epäonnistumisinformaation.

6.2.7 Sisäinen toteutus

Pysyväiskerroksen toteutus jätetään tuleville projekteille. Nyt tarjotaan vain toteutus, joka ei talleta mitään vaan palauttaa kaikista operaatioista onnistumistiedon. Listaukset palauttavat tyhjän listan.

6.3 Datakerrokseen liittyvät asetukset

Datakerroksen asetukset haetaan GNUnetin asiakaspuolen asetustiedostosta. Parametrit luetaan asetustiedoston osiosta ”[DHT-Store]”.

Asetustiedosto sisältää seuraavat parametrit:

QUOTA Datakerroksen koko megatavuina.

DHTDIR Pysyväiskerroksen totettaman tiedoston sijainti.

7 Tietorakenteet

Tässä osassa määritellään kaikki DHT-moduulissa ja DHT:n APIssa määritellyt tietorakenteet ja tietotyypit. Taulukossa on luetteloituna kaikki tietotyypit, jonka jälkeen jokaisesta on erillinen osio, missä tietotyyppi esitellään tarkemmin.

DHT_NodeId	yhden DHT-solmun identiteetti
DHT_TableId	yhden DHT-taulun identiteetti
DHT_TableMetaData	yhden DHT-taulun metadata
DHT_TableConfig	yhden DHT-taulun konfiguraatiotiedot
DHT_TableHandle	yhden DHT-taulun tilatiedot
DHT_DataContainer	rakenne datansiirtoa varten
DHT_ResultSet	rakenne DHT-taulun hakutuloksille
DHT_DataStroreUnit	paikalliseen solmuun talletettavan tietoalkion rakenne
DHT_LocalStoreResultSet	paikallisesta solmusta tehdyn haun tulos
DHT_NodeSet	lista DHT-solmuje identiteettejä

7.0.1 DHT_NodeId

DHT_NodeId on yhden DHT-solmun yksikäsitteinen identiteetti.

typdef struct HashCode160 * DHT_NodeID

7.0.2 DHT_TableId

DHT_TableId on yhden DHT-taulun yksikäsitteinen identiteetti.

typdef struct HashCode160 * DHT_TableId

7.0.3 DHT_TableMetaData

Yksittäiseen DHT-tauluun liittyvä metadata.

```
typedef struct DHT_TablemetaData {
   char *name;
} DHT_TableMetaData;
```

- ”’char *name”’ - name of the DHT table

kontekstit

- Sovellusrajapinnan funktion create() syöte.

7.0.4 DHT_TableConfig

Yksittäisen DHT-taulun asetustiedot.

```
typedef struct DHT_TableConfig {
   int replicationCount;
   int parallelismCount;
   int valuesPerKey;
   int expirationTimeSeconds;
   float cacheTimeMultiplier;
   int flags;
} DHT_TableConfig;
```

- "'int replicationCount"' - Tauluun liittyvän replikointiaste. Määrittää, kuinka monessa solmussa kutakin tauluun tallennettua dataa säilytetään samanaikaisesti (Kademlian k).
- "'int parallelismCount"' - Tauluun liittyvä rinnakkaisuuskerroin. Vaikuttaa avainhakujen nopeuteen (Kademlian alpha).
- "'int valuesPerKey"' - Tauluun liittyvä kerroin, joka vaikuttaa yhteen avaimeen tallennettavien arvojen määrään.
- "'int expirationTimeSeconds"' - Tauluun liittyvä datan säilytysaika sekunneissa. Määrittää, kuinka kauan tauluun tallennettua dataa säilytetään ennen kuin se poistetaan.
- "'float cacheTimeMultiplier"' - Tauluun liittyvän cachetusaikakerroin. Säätää tauluun liittyviä cachetusaikoja lyhyemmiksi (<1) tai pidemmiksi (>1).
  - cachetusaika on Kademlia-algoritmissa eksponentiaalisesti kääntäen verrannollinen solmun id:n ja tallennettavan avaimen hash-koodia lähimpänä olevan verkon solmun id:n välissä olevien solmujen lukumäärään
  - kerroin vaikuttaa em. cachetusaikoihin niitä edelleen lyhentävästi tai pidentävästi
- "'int flags"' - Tauluun liittyviä tosi- tai epätosi-tietoja.
  - 1. Bitti kertoo onko taulu private, ks. isPrivate(config.flags)
  - 2. Bitti kertoo ...

kontekstit

- Sovellusrajapinnan funktion create() syöte
- Datakerroksen create_table() syöte

- DHT-moduuli saa tällaisen liityttäessä uuteen tauluun
- DHT-moduuli saa datakerroksesta tällaisen jokaista liityttyä taulua kohden, kun solmu käynnistetään uudelleen

7.0.5 DHT_TableHandle

Yksittäisen DHT-taulun käyttötiedot.

```
typedef struct DHT_TableHandle {
  int errorCode;
  DHT_TableId tableId;
} DHT_Table_view_t;
```

- ”’int errorCode”’ - Viimeisimmän operaation virhekoodi
- ”’DHT_TableId tableId”’ - Taulun tunnistetieto.

kontekstit

- Sovellusrajapinnan funktion create() palaute.
- kaikki datakerroksen funktiot kohdistetaan tiettyyn tauluun.

7.0.6 DHT_DataContainer

Tietorakenne datan siirtämistä varten.

```
typedef struct DHT_DataContainer {
  int dataLength;
  void *data;
};
```

Käsittelyfunktiot DHT_DataContainer *dataContainerCreate(key, value);

key *dataContainerGetKey(DHT_DataContainer unit);

value *dataContainerGetValue(DHT_DataContainer unit);

7.0.7 DHT_DataStroreUnit

Tietorakenne, jonka avulla data tallennetaan paikalliseen datakerrokseen.

```
struct DHT_DataStroreUnit {
  Hash160b * hash;
  TIME timeout;
  unsigned char flags;
  DHT_DataContainer * data;
};
```

7.0.8 DHT_ResultSet

DHT_DataContainer *resultSetNext(DHT_ResultSet set);

7.0.9 DHT_RPCParam

Tietorakenne etäproseduurikutsujen kutsu- ja paluuparametrien välittämiseen.

```
struct DHT_RPCParam_t {
  int count;
  char ** name;
  DHT_DataContainer ** value;
};
typedef struct DHT_RPCParam_t DHT_RPCParam;
```

8 Testaus

8.1 Yksikkötestit

Yksikkötestien suoritukseen on kaavailtu XP-ympyröissä käytettyä xUnit-mallia. Yksinkertaisuudessaan xUnit viittaa testitapauksiin, jotka kirjoitetaan ohjelmakoodin yhteyteen itse koodia laadittaessa. Mahdollisesti voidaan myös toteuttaa testitapaukset ennen koodia ja siten seurata koodin valmiusastetta. Jokaiselle funktiolle (metodille) laaditaan joukko testitapauksia, jotka pyrkivät varmentamaan funktion toimivuuden a) normaalitapauksissa ja b) erilaisissa poikkeustapauksissa (ts. funktiolle annettu syöte on odottamatonta tai epäkelpoa). Essentiaalista on kyky ajaa testit automaattisesti siten, että testiajon tuloksena saadaan selkeä numeerinen arvo onnistuneiden (läpäisseiden) ja epäonnistuneiden testien lukumäärästä.

8.2 Testiverkko

DHT-toteutuksen järjestelmätestejä varten on viritettävä testiverkko. Lyhyt testi (22.3.) GNUnetin chat-sovelluksella osoitti, että julkista GNUnet-verkkoa ei ole syytä käyttää testeihin. Ilmeisesti chat-sovellus on pahemman kerran rikki, joten viestit jäivät kiertämään verkossa generoiden liikennettä verkosta esimerkiksi allekirjoittaneen kotikoneelle sisältönä sama muutaman viestin rimpsu yhteensä kolmatta sataa megatavua.

Järjestelmätestausta varten on siis oltava suljettu verkko. Osittain automatisoitu testiympäristö voitaisiin toteuttaa laitoksen koneympäristöön. Etuna laitoksen ympäristössä on jaettu levy. Oletetetaan $HOME = /group/home/dht/testnet/. $HOME/src/ sisältää checkoutatun CVS-puun, joka voidaan päivittää, kun halutaan testata uutta koodia. $HOME/sw/ sisältää käännetyn koodin eli käännettäessä annetaan –prefix=$HOME/sw/.

Yksittäiset GNUnet-solmut elävät hakemistoissa $HOME/nodes/nodename/, esimerkiksi $HOME/nodes/melkki/. Jokaisen solmun hakemistossa on sen asetustiedosto ja ajonaikainen data. Jotta testiverkon solmut tuntevat toisensa (ja vain toisensa), asetetaan ne hakemaan jaetusta tiedostosta verkon kattava joukko solmutunnuksia.

Testiverkon ylläpitoa voidaan automatisoida yksinkertaisilla skripteillä. Voidaan toteuttaa esimerkiksi skriptit, joilla saadaan kerralla käynnistettyä tai ajettua alas koko testiverkko. Lisäksi voidaan automaattisesti skriptillä poistaa kaikkien solmujen ajonaikainen data ja saattaa siten testiverkko nollatilaan uudelleenkäynnistystä varten.

9 Sovellus

DHT-toteutuksen testaamista varten laaditaan esimerkkisovellus, jonka toiminnot käyttävät kattavasti DHT-totetutuksen ominaisuuksia.

9.1 Hajautustaulujen käyttö

Sovellus käyttää kahta hajautustaulua. Sijaintitauluun talletetaan <avain,arvo> -pareina tiedostojenn nimet ja niiden sijainnit. Esimerkiksi ”<"apocalypse-now.avi", "http://example.com/apocaly
now.avi»”. Hakusanatauluun talletetaan tiedostoon liittyvät hakusanat. Esimerkiksi ”<"charlie", "apocalypse-now.avi»”, ”<"don’t", "apocalypse-now.avi»” ja ”<"surf", "apocalypse-now.avi»”.

Suoritettaessa haku esimerkiksi termeillä "charlie don’t surf"tehtäisiin kolme kyselyä hakusanataulusta. Hakutuloksista otetaan leikkaus eli sovelluksen käyttäjälle näytetään vain tiedostot, jotka löytyivät jokaisella hakusanalla. Tietty tiedosto voi löytyä sijaintitaulusta useammin kuin kerran; kaikki löydetyt sijainnit näytetään käyttäjälle.

Hajautustaulujen nimet koostuvat kahdesta osasta. Nimen perusosa on vapaavalintainen. Sijantitaulun nimi muodostetaan liittämällä perusosaan ”.locations” ja hakusanataulun liittämällä ”.keywords”. Esimerkiksi perusosalla ”movies” saataisiin sijantitaulu ”movies.locations” ja hakusanataulu ”movies.keywords”.

9.2 Käyttöliittymä

Sovelluksen käyttöliittymä toteutetaan käyttäen GTK+ -rajapintaa. Sovelluksen käyttöympäristö on Linux.

Käyttöliittymä jakaantuu joukkoon kehyksiä.

9.2.1 Hakukehys

Hakukehyksen yläreunassa on tekstikenttä, johon voidaan syöttää halutut hakutermit. Tekstikentän vieressä on painikkeet ”Hae” ja ”Keskeytä”. Syötettyjen termien perusteella suoritettava haku käynnistetään ”Hae”-painikkeesta. Haun ollessa käynnissä painike ”Keskeytä” on aktiivinen. Tekstikenttään syötettyjä termejä ei voi muokata haun ollessa käynnissä.

Jos hakutermejä vastaavia tuloksia löytyy, listataan ne hakukehyksessä vieritettävänä listana. Kaksoisklikkaamalla tiettyä tulosta käynnistyy siihen liittyvän tiedoston siirto verkosta paikalliselle koneelle. Siirron edistyminen esitetään erillisessä kehyksessä.

9.2.2 Siirtokehys

Jokaisen erillisen tiedoston siirtoa varten avataan kehys. Siirto suoritetaan ulkoisella, komentorivipohjaisella ohjelmalla (esimerkiksi ”wget”[1]). Siirtokehyksessä esitetään ulkoisen ohjelman tuloste. Siirtokehyksen otsakkeessa esitetään ohjelman tila eli käytännössä, onko se edelleen ajossa. Siirtokehystä ei automaattisesti suljeta ohjelman suorituksen päätyttyä.

9.2.3 Jaettavan tiedoston lisäys

Lisättäessä uusi jaettava tiedosto syötetään sen tiedot lomakemaisessa kehyksessä. Tiedostosta syötetään sen nimi, sijainti (URL) ja joukko avainsanoja.

Tiedoston nimestä ja sijainnista muodostetaan <avain,arvo> -pari siten, että nimi muodostaa avaimen ja sijanti arvon. Pari talletetaan sijantitauluun. Jokaisesta avainsanasta muodostetaan erillinen pari hakusanatauluun. Parien avaimina toimivat yksittäiset avainsanat ja arvona jokaisessa on annettu tiedoston nimi.

9.3 Asetukset

Sovelluksen asetukset haetaan GNUnetin asiakaspuolen käyttäjäkohtaisesta asetustiedostosta, joka oletusarvoinen sijainti on ” /.gnunet/gnunet.conf”. Parametrit luetaan asetustiedoston osiosta ”[DHT-FS]” (DHT File Sharing).

Asetustiedosto sisältää seuraavat parametrit:

DOWNLOAD-TOOL Polku ulkoiseen ohjelmaan, jota käytetään tiedostojen siirtoon. Kun ohjelma käynnistetään, annetaan sille parametrina haettavan tiedoston osoite (URL).

DOWNLOAD-DIR Polku hakemistoon, johon siirrettävät tiedostot talletetaan. Ennen ”DOWNLOAD-TOOL”-parametrilla määritellyn siirto-ohjelman käynnistämistä sovellus siirtyy ”DOWNLOAD-DIR”-parametrilla määritettyyn hakemistoon.

BASENAME Käytettävien hajautustaulujen nimien perusosa siten, että sijantitaulun nimi on ”BASENAME.locations” ja hakusanataulun ”BASENAME.keywords”.

[1] http://www.gnu.org/software/wget/

10 Ohjelmointikäytännöt

DHT-moduulin ohjelmakoodissa tulee noudattaa GNUnetin muun lähdekoodin kanssa yhdenmukaisia ohjelmointikäytäntöjä.

10.1 Nimeäminen

GNUnetin nimeämiskäytäntö muistuttaa enemmän Java-kielen käytäntöjä kuin perinteisiä C-nimeämiskäytäntöjä. Koska DHT-moduulin tietueiden halutaan erottuvan GNUnet-ytimen symbolinnimistä, niissä käytetään etuliitettä `DHT_`. Tätä etuliitettä lukuunottamatta käytännöt ovat samoja kuin GNUnetissä.

10.1.1 Tietueet

Tietueiden nimissä erilliset sanat kirjoitetaan yhteen isoilla alkukirjaimilla. Nimen eteen tulee etuliite `DHT_`. Esim: `DHT_TableConfig`.

10.1.2 Funktiot

Funktioiden nimissä erilliset sanat kirjoitetaan yhteen. Ensimmäinen sana on pienellä alkukirjaimella, loput isoilla alkukirjaimilla. Esim: `nodeLookup`.

10.1.3 Muuttujat

Muuttujat nimetään kuten funktiot.

10.1.4 Makrot

Makrojen nimissä sanat kirjoitetaan kokonaan isoilla kirjaimilla ja erilliset sanat erotetaan väliviivoilla. Esim: `DHT_HASH_KEY_NUM_BITS`.

10.2 Ohjelmakoodin ladonta

Ladontakonventio vaihtelee jonkin verran GNUnetin eri kooditiedostoissa. Alla luetellut piirteet näyttävät olevan kohtalaisen yhtenäisessä käytössä GNUnetin eri lähdekooditiedostoissa.

10.2.1 Sisennys

Yksi sisennystaso on kaksi merkkiä.

GNUnetin lähdekoodissa sisennyksessä käytetään sekaisin välilyöntejä ja fyysisiä tab-merkkejä. Fyysinen tab on näytettävä kahdeksan merkin siirtymänä, jotta lähdekoodi näyttäisi oikealta. Tällainen tapa käyttää fyysisiä tabeja ei ole hyvä ohjelmointikäytäntö, joten DHT-moduulissa kaikki sisennykset tehdään välilyönneillä.

10.2.2 Sekalaista

- Erotuksena funktiosta sisäänrakennetut `if`, `while`, `switch` jne. -rakenteet erotetaan niitä seuraavasta sulkulauseesta välilyönnillä.
- Osoitintyyppien *-merkki erotetaan välilyönnillä sekä muuttujatyypistä että muuttujanimestä.
- Koodiblokin aloittava aaltosulje on samalla rivillä sitä edeltävän lauseen kanssa, ei omalla rivillään.
- Useampiparametristen funktioiden deklaraatioissa parametrit ladotaan allekkain. Parametrit ladotaan allekkain tarvittaessa myös funktiokutsussa.

10.2.3 Esimerkki

```
int fooBar(char * xyz
           int zy) {
  printf("Hello, world!\n");
  if (!strcat(xyz, "barFoo") {
    LOG(LOG_REPORT,
        "URGENT: fooBar called with "barFoo"!\n");
  }
  return zy != SOME_MAGIC_NUMBER;
}
```

Lähteet

MM Maymounkov, P. ja Mazières, D., Kademlia: A peer-to-peer information system based on the xor metric.