Hakualgoritmit järjestämättömissä vertaisverkoissa

Tommi Myöhänen

Tietotekniikan (ohjelmistotekniikka)
pro gradu -tutkielma
4. kesäkuuta 2008

Jyväskylän yliopisto
Tietotekniikan laitos

Tekijä: Tommi Myöhänen

Yhteystiedot: tommi.myohanen@gmail.com

Työn nimi: Hakualgoritmit järjestämättömissä vertaisverkoissa

Title in english: Search algorithms in unstructured Peer-to-Peer networks

Työ: Pro gradu -tutkielma

Sivumäärä: 64

Linja: Ohjelmistotekniikka

Avainsanat: vertaisverkko, P2P, potenssijakauma, hakualgoritmi

Keywords: peer-to-peer, P2P, power-law, search algorithm

Tiivistelmä

Vertaisverkot tarjoavat täysin uudenlaisen tavan tiedon ja resurssien jakamiseen verkossa. Tietoverkkojen kasvaessa ja tietomäärän lisääntyessä on entistä vaikeampaa löytää haluamaansa kohdetta tästä kokonaisuudesta. Tällöin joudutaan kehittämään uusia hakualgoritmeja, jotka pystyvät löytämään tarvittavan tiedon nopeasti ja kustannustehokkaasti. Tässä tutkielmassa paneudutaan muutamaan vertaisverkon hakualgoritmiin ja kerrotaan, kuinka ne toteutettiin ja testattiin P2PCore-ohjelmiston avulla.

Abstract

Peer-to-Peer networks provide a totally new way of sharing information and all kind of resources in a network. Growing network sizes and increasing amount of information makes it even harder to find something from this entity. Therefore new search algorithms are needed to retrieve the desired information quickly and efficiently. This thesis examines some Peer-to-Peer search algorithms and explains how they were implemented and tested in P2PCore -software.

Termiluettelo

ABFS	Adaptive Breadth-First-Search. Adaptiivinen leveyshaku.
BFS	Breadth-First-Search, leveyshaku.
Client-Server	Asiakas-Palvelin -arkkitehtuuri.
DBFS	Directed Breadth-First-Search, suunnattu leveyshaku.
Gnuplot	Numeerisen tiedon ja matemaattisten funktioiden visualisointiin tarkoitettu työkalu.
HDEG	Highest Degree Search, korkeimman asteen haku.
HTTP	HyperText Transfer Protocol. WWW-sivujen siirtoprotokolla.
Lehtisolmu	Solmu, jolla on vain yksi yhteys.
PDA	Personal Digital Assistant, kämmentietokone.
P2P	Lyhenne termistä Peer-to-Peer, vertaisverkko.
RWALK	Random Walker Search, satunnaiskävelijä-haku.
TCP/IP	Transmission Control Protocol / Internet Protocol. Internetin tiedonsiirtoon käytetty protokolla.
TTL	Time-To-Live.
Ydinsolmu	Solmu, jolla on paljon yhteyksiä ja joka välittää suuren osan verkkoliikenteestä.

Sisältö

1 Johdanto **1**

2 Vertaisverkot **2**

2.1 Mitä vertaisverkot ovat? 2

2.2 Käyttökohteet 4

2.3 Ominaisuudet 5

2.3.1 Tiedonvälitys 5

2.3.2 Tehokkuus 6

2.3.3 Tietoturva 6

2.3.4 Vikasietoisuus 7

2.4 Tulevaisuus 8

3 Vertaisverkkojen rakenne ja toimintaperiaate **9**

3.1 Verkkotyypit 9

3.2 Mittakaavaton verkko 10

3.3 Potenssijakautuneet verkot 10

3.4 Toimintaperiaate 11

3.4.1 Tiedon kulku verkossa 11

3.4.2 Resurssien haku 13

4 Hakualgoritmit **14**

4.1 Algoritmien toimintaperiaate 14

4.1.1 Viestien reititys 15

4.1.2 Ongelmakohtia 16

4.2 Määritelmiä 16

4.3 Leveyshaku (Breadth-First Search (BFS)) 17

4.3.1 Reititys 18

4.4 Adaptive Breadth-First Search (ABFS) 19

4.4.1 Reititys 20

4.5 Satunnaiskävelijä (Random Walker Search (RWALK)) 21

4.5.1 Reititys 22

4.6 Korkeimman asteen haku (Highest Degree Search (HDEG)) 23
4.6.1 Reititys 24
4.7 Muita vertaisverkkoihin liittyviä menetelmiä 27
4.7.1 Topologian hallinta-algoritmit 28
4.7.2 Adaptiiviset hakualgoritmit 28
4.7.3 Verkon historiatiedon hyödyntäminen 28

5 Tutkimusympäristö **29**
5.1 P2PCore 29
5.1.1 Ominaisuudet 29
5.1.2 Ohjelmistokomponentit 30
5.1.3 Työkalut 31
5.1.4 Käytetyt algoritmit 33
5.1.5 Protokolla 33
5.1.6 Tiedonsiirto 34
5.1.7 Suorituskyky 34
5.2 Tutkimusmenetelmä 35
5.2.1 Testausprosessi 36
5.2.2 Testaamisen ongelmakohtia 37

6 Hakualgoritmien suorituskykyvertailu **38**
6.1 Nopeus 39
6.1.1 Yhden resurssin etsintä 39
6.1.2 Etsittäessä 50% resursseista 40
6.2 Verkkoliikenteen määrä 41
6.2.1 Yhden resurssin etsintä 41
6.2.2 Etsittäessä 50% resursseista 42

7 Johtopäätökset **43**

8 Aiheeseen liittyviä tutkimuksia sekä jatkotutkimusideoita **44**

9 Yhteenveto **47**

Viitteet **49**

10 Liitteet **52**

10.1 Liite 1: Algoritmikohtaiset tulosdiagrammit 52

10.2 Liite 2: Numeeriset testitulokset 54

10.3 Liite 3: P2PCore-vertaisverkkoalustan tietoja 55

10.4 Liite 4: Esimerkki *P2PTool*-työkalun ohjaustiedostosta 56

1 Johdanto

Vertaisverkkojen hyödyt tulevat parhaiten esiin hajautusta ja vikasietoisuutta vaativissa tehtävissä, joita ovat esimerkiksi hajautettu laskenta tai tilastollisen datan kerääminen verkosta. Ne soveltuvat myös mainiosti tehtäviin, joissa ei vaadita tiettyä verkkorakennetta, vaan pikemminkin kykyä toteuttaa haluttu toiminto jatkuvasti muuttuvassa topologiassa.

Tässä pro gradu -tutkielmassa tutustutaan leveyshakuun (Breadth-First-Search, BFS), korkeimman asteen hakuun (Highest Degree Search, HDEG) ja satunnaiskävelijähakuun (Random Walker Search, RWALK) sekä tutkitaan niiden tehokkuutta sekä nopeutta tätä tarkoitusta varten kehitetyllä P2PCore-ohjelmistolla.

Seuraavassa kappaleessa tutustutaan lähemmin vertaisverkkoteknologiaan ja mitä niillä pystytään tekemään. Kappaleessa 3 tarkastellaan vertaisverkkojen rakennetta ja kappaleessa 4 keskitytään itse hakualgoritmeihin ja niiden toteutuksiin. Kappale 5 esittelee P2PCore-sovellusalustan ja varsinaisen tutkimusympäristön ja kappaleessa 6 analysoidaan saatuja tuloksia.

2 Vertaisverkot

Perinteiset Asiakas-Palvelin -tyyppiset verkkoratkaisut ovat parin viime vuoden aikana saaneet uusia kilpailijoita. Laajakaistaliittymien yleistyminen ja tiedon lisääntyminen verkossa ovat edesauttaneet näiden uusien verkkomallien yleistymistä. Useissa nykypäivän verkkosovelluksissa ei ole enää edullista käsitellä ja siirtää valtavia tietomääriä keskitetyn palvelimen kautta. On siis jouduttu keksimään uusia, tehokkaampia ratkaisuja. Vertaisverkot ovat yksi näistä uusista verkkomalleista.

2.1 Mitä vertaisverkot ovat?

Termi *vertaisverkko* (Peer-to-Peer, P2P) tarkoittaa hajautettua verkkotopologiaa, jossa kaikki verkon jäsenet eli solmut (engl. *peer*) ovat nimensä mukaisesti samanvertaisia. Jokainen solmu toimii itsenäisesti tiettyjen pelisääntöjen mukaan.

Erona perinteiseen Asiakas-Palvelin -malliin vertaisverkoilla ei ole ennalta määrättyä topologiaa tai hierarkiaa, vaan verkko voi olla hyvinkin hajanainen ja jatkuvasti muuttuva. Verkkorakenteeseen voi tulla uusia solmuja ja jo olemassaolevat solmut voivat poistua verkosta millä hetkellä hyvänsä.

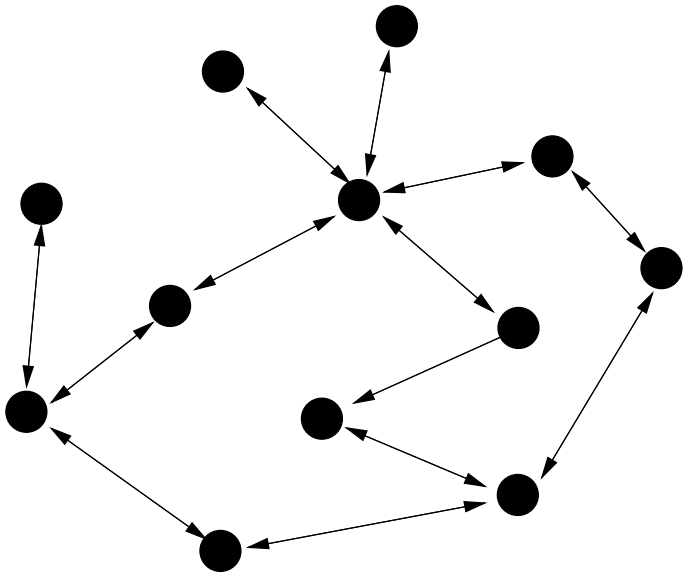
Suurin ero on kuitenkin yksittäisen solmun rakenteessa. Kukin vertaisverkon solmu toimii samanaikaisesti sekä asiakkaana että palvelimena. Asiakkaana toimiessaan solmu käyttää verkon resursseja hyväkseen naapurisolmujen avulla, eli lähettää pyyntöjä verkkoon ja odottaa niihin vastauksia. Palvelimena toimiessaan solmu on velvoitettu kuuntelemaan verkosta tulevia yhteydenottopyyntöjä ja luomaan yhteyksiä muihin solmuihin näiden pyyntöjen mukaisesti. Palvelin on myös vastuussa verkon dataliikenteen uudelleenohjauksesta ja kontrolloinnista. Englanninkielisissä julkaisuissa vertaisverkon solmusta puhutaankin nimellä “servent”, joka tulee sanoista SERVer ja cliENT.

Edellä mainittu toiminnallisuus viittaakin jo siihen, että verkon solmu ei ole yhteydessä pelkästään yhteen naapurisolmuun, vaan liityntöjä verkkoon voi olla jopa kymmeniä. Solmun liittyessä verkkoon sillä on oikeus valita liityntäpisteensä vapaasti. Juuri tämä vapaus tekee verkon rakenteesta monimutkaisen ja tiheän. Verkkosovelluksesta riippuen rakenne voi olla hyvinkin epäoptimaalinen ajatellen perinteiseen staattisen verkkorakenteen tiedonsiirtoon liittyviä näkökulmia, kuten viivettä, kaistanleveyttä ja solmujen välistä maantieteellistä etäisyyttä.

Nämä seikat eivät kuitenkaan muodosta ongelmaa, sillä vertaisverkon vahvuudet

piilevät aivan toisaalla. Palvelin-asiakasmallissa palvelin on suuri riskitekijä verkon toiminnan kannalta, sillä mikäli palvelin lakkaa toimimasta, koko verkko lamaantuu. Parhaiten tällaista ongelmaa kuvaa englanninkielinen termi "single point of failure". Vertaisverkossa tällainen ongelma ei ole mahdollinen, sillä yhden (tai jopa useamman) solmun pettäessä verkko on silti toimintakykyinen. Verkko voi jopa parhaassa tapauksessa osata uudelleenohjata dataliikenteen toista reittiä perille.

Yleistä vertaisverkkosovelluksille on se, etteivät ne yleensä vaadi verkkosolmuilta identiteetin paljastamista tai oleta tiettyjen resurssien olemassaoloa. Päinvastoin, toiminta perustuu hyvin pitkälti vapaaehtoisuuteen ja tasa-arvoisuuteen.



Kuva 2.1: Esimerkki vertaisverkon topologiasta.

Kuva 2.1 esittää kuvitteellista 12 solmun vertaisverkkorakennetta. Kuten voidaan havaita, verkko voi sisältää kehärakenteita ja vaihtoehtoisia reittejä solmujen välillä. Kuvasta voidaan erottaa myös vähän yhteyksiä omaavat reunasolmut (engl. *leaf node*) sekä verkon rungon muodostavat, paljon yhteyksiä omistavat ydinsolmut. Verkkotopologioihin syvennytään tarkemmin luvussa kolme.

S. Androutsellis-Theotokis ja D. Spinellis antavat julkaisussaan "A Survey of Peer-to-Peer Content Distribution Technologies" [2] kattavan yleiskuvan tämänhetken vertaisverkkosovelluksista, niiden toimintatavoista sekä käyttötarkoituksista. Siksi tässä tutkimuksessa sivuutetaan perusteellisempi vertaisverkkojen yleistoiminnan kuvaus ja keskitytään enemmän itse hakualgoritmien toimintaan.

2.2 Käyttökohteet

Suurin osa tämänhetkisistä vertaisverkkoa hyödyntävistä sovelluksista on tiedostojen jakamispalveluita. Alan pioneerina toimi Gnutella-verkkosovellus [12], jonka protokolla julkaistiin vuoden 1999 lopulla. Se oli ensimmäinen hajautettu tiedostojen jakoon tarkoitettu ohjelmisto, joka hyödynsi uutta vertaisverkkomallia. Tämän jälkeen aluetta ovat vallanneet lukuisat kilpailevat sovellukset.

Toki vertaisverkoille on olemassa myös muitakin käyttötarkoituksia. Freenet-projektin [9] tarkoituksena on pyrkiä luomaan anonyymi ja jäljitykseltä vapaa verkkokerros tavallisen TCP/IP -pohjaisen verkon päälle käyttäen vertaisverkkoratkaisua. Tällainen verkko tarjoaa käyttäjilleen identiteettisuojan, sillä tiedonsiirtoon ei käytetä mitään jäljitettävissä olevia keinoja, kuten IP-osoitteita tai loogisia verkkotunnuksia.

OceanStore-projekti [16] puolestaan rakentaa globaalia tiedontallennusjärjestelmää pohjautuen niinikään hajautettuun vertaisverkkoon. Sen tärkeimpiä tavoitteita ovat eheys, korkea käytettävyysaste ja kestävyys. Tällöin verkon käyttäjällä olisi lähes ääretön tietovarasto käytettävissään, sillä kaikki verkkoon liitetyt koneet tarjoavat omaa paikallista tallennustilaansa yhteiseen käyttöön. Tällainen lähestymistapa vaatii kuitenkin erittäin monimutkaisia käsittelyalgoritmeja tiedon varastointiin, sillä sovelluksen on pystyttävä takaamaan tallennetun datan säilyvyys.

Oman kategoriansa muodostavat valmiit ohjelmointirajapinnat ja verkkoalustat, joita voidaan käyttää kehitettäessä sovelluksia, jotka tarvitsevat vertaisverkkopalveluja. Ne tarjoavat valmiin verkkoimplementaation, joten sovelluskehittäjän ei tarvitse paneutua matalan tason verkkoliikenteeseen vaan voi keskittyä itse sovelluksen kehittämiseen. Eräs tällainen alusta on JXTA-projektin [13] kehittämä virtuaalisen verkkokerroksen tarjoava ohjelmointikirjasto.

Lisäksi vertaisverkkorakennetta käyttävät muun muassa internet-puhelimet kuten Skype [25] ja monet verkkoradioiden lähetysohjelmistot, esimerkiksi PeerCast [20]. Yksi uusimmista vertaisverkkoa käyttävistä sovelluksista on Joost [14], jonka avulla voi katsella kymmeniä tuhansia videolähetyksiä Internetin kautta.

Vertaisverkkojen käyttökohteet eivät rajoitu pelkästään nykyisiin toteutuksiin, vaan niiden avulla voitaisiin rakentaa hyvinkin monipuolisia palveluita, kuten esimerkiksi

- **Hajautettu laskenta** voisi hyödyntää vertaisverkkoa vapaiden laskentaresurssien (prosessori) etsimiseen.

- **Grafiikan renderöinti** pystyisi niin ikään etsimään verkon solmuja, jotka voivat tarjota prosessoriaikaansa grafiikan tuottamiseen.
- **Erittäin korkeaa vikasietoisuutta vaativat kohteet**, esimerkiksi sotilasverkot.
- **Tiedon kerääminen ja edelleenohjaaminen**. Esimerkiksi Jyväskylän yliopistossa on kehitetty menetelmä, jossa mobiililaitteessa toimiva sovellus poimii bensiinin hintatiedon tankkauksen yhteydessä. Kohdatessaan muita mobiililaitteita tätä tietoa levitetään verkossa eteenpäin [29].

2.3 Ominaisuudet

Vertaisverkkojen rakenne tuo mukanaan tiettyjä teknisiä mahdollisuuksia ja ongelmia. Tässä luvussa paneudutaan syvemmin niiden ominaisuuksiin ja tutustutaan tavanomaisimpien verkkorakenteiden toimintaperiaatteisiin. Lisäksi käsitellään muutamia erikoistilanteita, jotka ovat ominaisia juuri hajautetuille verkkotopologioille.

2.3.1 Tiedonvälitys

Tavalliseen Asiakas-Palvelin -malliin verrattuna vertaisverkkojen tiedonsiirto vaatii paljon enemmän käsittelytehoa, sillä lähettävä ja vastaanottava osapuoli eivät ole suorassa yhteydessä toisiinsa. Vertaisverkossa tieto kulkee useiden välikäsien kautta solmulta toiselle, kunnes se saapuu vastaanottajasolmuun.

Tällainen tiedonsiirtomuoto vaatii verkolta paljon älykkyyttä ja yhteispeliä. Solmujen on osattava ymmärtää vastaanotettu viesti, käsitellä se ja lähettää tarvittaessa eteenpäin kohti määränpäätänsä. Lisäksi verkon dynaamisuus aiheuttaa lisäongelmia, sillä yhteydet voivat katketa kesken tiedonsiirron tai uusia lyhyempiä (ja nopeampia) reittejä voi muodostua toiminnan aikana.

Jotta verkon solmut voivat toimia keskenään, niiden on tiedettävä yhteiset pelisäännöt, eli *protokolla*. Se määrää siirrettävän tiedon formaatin ja käytettävän "kielen", millä verkon solmut keskustelevat toistensa kanssa.

Protokollan ohella toinen merkittävä tekijä solmujen välisessä tiedonsiirrossa on käytetty hakualgoritmi. Tämä tarkoittaa tapaa, jolla verkon solmut prosessoivat ja ohjaavat tiedon kulkua. Hakualgoritmi siis määrittelee säännöstön, minkä avulla kukin verkon solmu päättelee vastaanotetulle viestille tehtävän operaation. Jokaisella solmulla on oltava käytössään jokin hakualgoritmi, ja yksinkertaisin algoritmi

voisi olla esimerkiksi viestin eteenpäinlähettäminen sellaisenaan. Koko verkon solmujen ei siis tarvitse välttämättä toteuttaa samaa algoritmia, pääasia on että viestiä välitetään eteenpäin.

Tämän pro pradu-tutkielman aiheena on pääasiassa tutustua näihin hakualgoritmeihin ja vertailla niiden suorituskykyä ja ominaisuuksia.

2.3.2 Tehokkuus

Tiedonhaun nopeuden kannalta vertaisverkot eivät välttämättä ole optimaalisin ratkaisu. Solmulta solmulle tapahtuva liikenne kuluttaa siirtokapasiteettia ja aiheuttaa ylimääräistä kuormaa. Tarkasti ennaltamääriteltyyn ja staattiseen topologiaan verrattuna kyseinen verkkoratkaisu näyttää kaoottiselta ja resursseja tuhlailevalta rakenteelta.

Verkon satunnainen rakenne tarkoittaa myös sitä, että haettavaa tietoa joudutaan hakemaan umpimähkään, tietämättä, mistä päin sitä kannattaisi etsiä. Tästä syystä hakunopeudet ovat hitaampia kuin tarkkaan määritellyllä verkolla. Käytettävästä hakualgoritmista riippuen hakuviesti saattaa kulkea verkossa pitkiäkin matkoja löytämättä mitään, vaikka etsittävä resurssi olisi etsijän naapurisolmulla.

Kyseisen teknologian hyödyt piilevätkin aivan muualla, muunmuassa verkon vikasietoisuudessa ja skaalautuvuudessa. Tehokkaaksi vertaisverkon tekee sen kyky adaptoitua muuttuvaan verkkorakenteeseen ja toimia itsenäisesti kenenkään ohjaamatta. Yksikään verkon solmu ei voi lamaannuttaa koko verkkoa, eikä verkko puolestaan voi mitenkään sulkea yksittäistä solmua ulkopuolelleen.

2.3.3 Tietoturva

Vertaisverkon perusperiaatteena on avoimuus. Verkkorakenteeseen voi liittyä mikä tahansa protokollaa tukeva solmu, joten verkko on rakennettava erittäin joustavaksi. Verkon muut solmut eivät voi tietää, mitä naapurisolmu vastaanottamillaan viesteillä tekee, joten kovin arkaluontoista tietoa ei vertaisverkossa voi siirtää. Vertaisverkon tiedonvälitystä voidaankin verrata sähköpostin välitysmekanismiin; lähettäjä tai vastaanottaja eivät koskaan tiedä, mitä reittiä pitkin viesti toimitettiin ja onko joku mahdollisesti nähnyt tai muuttanut viestin sisällön.

Verkon käyttämä protokolla voi toki tarjota solmujen autentikointimenetelmiä ja täten parantaa tietoturvaa. Tällöin verkkoon voi liittyä ainoastaan luotettavia solmu-

ja, jotka tietävät verkon käyttämän tunnistautumismenetelmän ja omaavat autentikointiin oikeuttavat tunnukset. Tällä tavalla päästään tarvittaessa eroon mahdollisista vakoojasolmuista, jotka pelkästään tarkkailevat verkon liikennettä ja keräävät tietoa.

Verkkoliikenne voi myös olla salattu vaikkapa julkisen avaimen salausmenetelmällä, jolloin solmut kuljettavat salattuja viestejä kykenemättä ymmärtämään niiden sisältöä. Vain viestien lähettäjä ja vastaanottaja pystyvät tähän. Verkkoprotokolla itsessään ei ole salattu, vaan salattu tieto kulkee viestien sisällä kapseloituna, ns. hyötykuormana (engl. *Payload*).

Emil Sit ja Robert Morris [24] ovat tutkineet hajautettuun tietojärjestelmään kohdistuvia yleisimpiä hyökkäysmetodeja ja esittelevät, kuinka niiltä voidaan puolustautua. Tutkimus lähestyy aihealuetta hajautettujen hajautustaulujen (engl. *Distributed Hash Table*) näkökulmasta, mutta samaa teoriaa voidaan soveltaa vertaisverkkoihin yleisestikin. Esimerkiksi suositukset pyrkiä välttämään yksittäisen solmun vastuun liiallista kasvattamista verkon hallitsevana solmuna ja olla luottamatta sokeasti naapurisolmuihin pätevät yleisesti vertaisverkkojen suunnittelussa.

2.3.4 Vikasietoisuus

Avoimen vertaisverkon on esimerkiksi pystyttävä sietämään tilanne, jossa verkkoon liittyy pahaa aikova solmu, joka alkaa häiritsemään verkkoliikennettä esimerkiksi väärentämällä saamiaan hakuviestejä tai lähettämällä virheellisiä viestipaketteja verkkoon.

Koska verkossa ei ole keskitettyä solmua, joka voisi havaita häiriköijän ja katkaista yhteyden siihen, täytyy tilanne hoitaa muulla tavalla. Esimerkiksi verkon topologiasta vastaava algoritmi voi havaita häirikkösolmun tuottavan liikaa liikennettä verkkoon tuottamatta yhtään vastausviestiä. Tällöin se voi katkaista yhteyden tähän häirikkösolmuun ja avata uuden, laadukkaamman yhteyden johonkin toiseen solmuun, joka toimii luotettavammin.

Toinen haaste, johon vertaisverkon täytyy pystyä vastaamaan, on jatkuvasti muuttuva verkkorakenne. Verkosta saattaa millä hetkellä hyvänsä poistua keskeinen ydinsolmu, joka omaa paljon yhteyksiä. Nämä menetetyt yhteydet on pystyttävä korvaamaan jollakin tapaa. Algoritmista riippuen verkko voi alkaa kierrättämään liikennettä muita reittejä pitkin tai se voi valita uuden keskussolmun.

2.4 Tulevaisuus

Vertaisverkkojen tulevaisuus näyttää lupaavalta. Langattomien tietoliikenneyhteyksien myötä verkkojen dynaamisuus korostuu, jolloin kiinteään topologiaan perustuvat verkkorakenteet eivät enää pelkästään sovellu käytettäväksi.

Viimeiset vuodet ovat näyttäneet, että vertaisverkoille keksitään jatkuvasti uusia käyttökohteita. Esimerkiksi kannettavien PDA-laitteiden välinen suora tiedonsiirto tulee yleistymään lähitulevaisuudessa suuresti.

Myös viihdeteollisuus on havainnut vertaisverkkojen tehokkuuden. Muun muassa seuraavat sovelluskohteet ovat alkaneet käyttämään vertaisverkkoratkaisuja:

- Monet verkkopelit käyttävät nykyään vertaisverkkoa keskitetyn palvelimen sijaan.
- Median reaaliaikainen jakaminen. Esimerkiksi internet-radioiden tai videokanavien sisällön välittäminen asiakkaille voi tapahtua suoraan asiakkaalta toiselle. Tämä vähentää palvelimen kuormaa ja tiedonsiirtokaistaa.
- Hajautetut dokumenttikirjastot, esimerkiksi PlanetP [10] tallentavat ja hakevat tietoa suoraan verkon solmuista.
- Tiedon jakaminen suoraan mobiililaitteiden kesken.

3 Vertaisverkkojen rakenne ja toimintaperiaate

3.1 Verkkotyypit

Vertaisverkoille ei ole olemassa yhtä ja ainoaa "oikeanlaista" verkkorakennetta. Sovelluksen käyttötarkoitus sanelee viime kädessä käytännön toteutuksen. Karkeasti jaoteltuna vertaisverkot voidaan kategorisoida kolmeen eri luokkaan, jotka ovat:

- Aidot P2P-verkot
- Hybridiverkot
- Muut

Niinsanotut *aidot P2P-verkot* käyttävät toimintaansa ainoastaan vertaisverkkoa. Niiden toiminta perustuu täysin hajautettuun rakenteeseen ja tiedonsiirto tapahtuu reitittämällä datapaketteja solmujen välillä. Tämä aiheuttaa verkon solmuille lisäkuormaa, sillä ne joutuvat käsittelemään jokaisen vastaanottamansa datapaketin, vaikkei se olisikaan tarkoitettu kyseiselle solmulle. Verkko itsessäänkin kuormittuu suuresta siirrettävästä tietomäärästä.

Hybridi-P2P tarkoittaa nimensä mukaisesti vertaisverkon ja perinteisen Asiakas-Palvelin -mallisen verkon yhdistelmää. Pääajatuksena tällaisissa ratkaisuissa on ollut hyödyntää kummankin rakenteen parhaat puolet. Esimerkiksi Napster-sovellus [18] käyttää tällaista ratkaisua. Sovellus suorittaa hakuoperaatiot käyttäen keskitettyä palvelinta, jolloin haku on erittäin nopea ja tehokas. Varsinainen tiedostonsiirto tapahtuu puolestaan avaamalla suora yhteys lähettävän ja vastaanottavan solmun välille. Kun siirto on loppunut, yhteys katkaistaan.

Muuntyyppisiin vertaisverkkoratkaisuihin kuuluu muun muassa *BitTorrent*-sovellus [7], joka on tarkoitettu erityisesti tiedostojen jakeluun hajautetusti. Kyseinen ohjelmisto kääntää perinteisen vertaisverkkoajattelun päälaelleen, sillä se ei tarjoa hajautettuja resurssien etsintäpalveluja, vaan hajautus on siirretty itse tiedoston latausvaiheeseen. Tiedoston tarjoaja julkistaa verkossa (vaikkapa HTTP-linkityksen avulla) tiedostokuvauksen kyseisestä resurssista, jonka verkkoasiakasohjelma käy ensin hakemassa. Paikka, josta kuvaus löytyy, pitää olla käyttäjän tiedossa. Tämän kuvauksen perusteella sovellus sitten lähtee etsimään verkosta kyseistä tiedostoa ja alkaa ladata sitä löytämistään lähteistä. Tiedostokuvaus siis sisältää metainformaatiota kyseisestä resurssista, ja hajautettua verkkorakennetta käytetään vasta tämän

tiedon hakemiseen. Tällöin saavutetaan suurempi siirtonopeus, sillä sovellus osaa ladata tiedostoa useasta lähteestä samanaikaisesti. Latauksen aikana asiakassovellus toimii samanaikaisesti itsekin lähteenä muille asiakasohjelmille, jotka haluavat ladata samaa tiedostoa.

Tarkat määritelmät eri verkkotyypeille löytyy Rüdiger Schollmeierin julkaisusta "A Definition of *Peer-to-Peer* Networking for the Classification of *Peer-to-Peer* Architectures and Applications" [22].

Tässä tutkielmassa keskitytään pelkästään aitoihin vertaisverkkorakenteisiin.

3.2 Mittakaavaton verkko

Vuosituhannen vaihteessa huomattiin, että monet verkkorakenteet muotoutuvat asteittain tietynlaiseen muotoon. Esimerkiksi WWW-sivujen väliset linkitykset eivät noudattaneetkaan oletettua normaalijakaumaa. Sama jakaumarakenne toistui monissa yllättävissäkin yhteyksissä, kuten ihmisten välisissä sosiaalisissa verkostoissa ja lentokenttien välisissä lentoreiteissä.

Tällaista rakennetta alettiin kutsua nimellä "mittakaavaton verkko" (engl. *Scale-Free network*), joka juontaa juurensa kaavasta, jolla verkon linkkien jakautuminen voidaan ilmaista:

$$P(n) = \frac{1}{n^k}$$

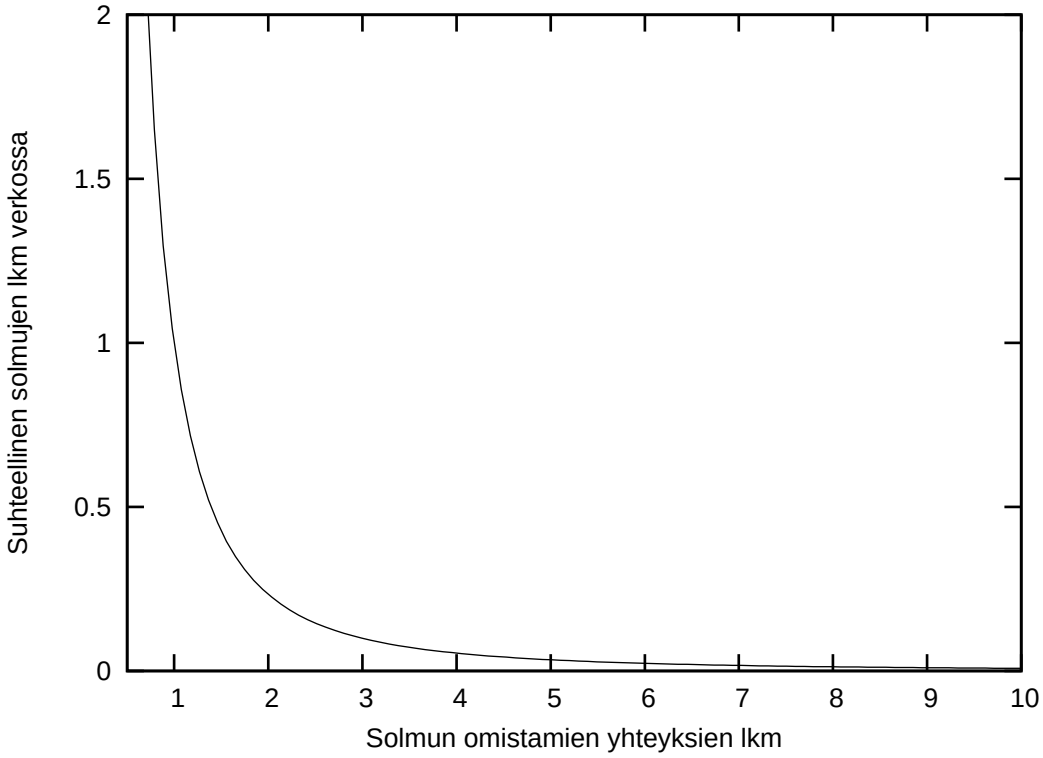
Kaavassa $P(n)$ tarkoittaa sellaisten solmujen suhteellista määrää, joilla on n kappaletta linkkejä toisiin solmuihin. k on verkosta riippuva vakio. Kyseinen kaava ei ota millään tavalla kantaa verkon kokoon. Sen tuottamassa jakaumassa ei myöskään ole huippukohtaa, eikä siitä voida poimia verkkoa edustavaa keskimääräistä solmua kuten normaalijakaumasta. Tästä tulee nimitys "mittakaavaton".

Tällainen verkko rakentuu liittämällä siihen lisää solmuja yksi kerrallaan. Uusi solmu liitetään verkkoon kahden yhteyden kautta tiettyjen valintaperusteiden mukaan. Saatu verkko toteuttaa mittakaavattoman verkon rakenteen [6].

3.3 Potenssijakautuneet verkot

Potenssijakautunut verkko tarkoittaa verkkotopologiaa, jossa pienellä osalla solmuja on suuri määrä yhteyksiä ja suurella määrällä solmuja vain pieni määrä yhteyksiä [3].

Edellä mainittu mittakaavaton verkkorakenne noudattaa potenssijakaumaa. Tästä syystä myös tässä tutkielmassa käytetyt verkkorakenteet on tehty potenssijakauman mukaisiksi. Kuvassa 3.2 on esimerkki potenssijakaumasta, jossa k=2.1. Siitä voi nähdä, että esimerkiksi yhden yhteyden omistavien solmujen määrä suhteessa kaksi yhteyttä omistaviin solmuihin on noin nelinkertainen (suhdeluku on noin 1 : 0.25).



Kuva 3.2: Esimerkki verkon potenssijakaumasta

3.4 Toimintaperiaate

Tässä kappaleessa kerrotaan lyhyesti vertaisverkkojen toimintaperiaate yksittäisen solmun näkökulmasta.

3.4.1 Tiedon kulku verkossa

Kuten kappaleessa 2.3 mainittiin, vertaisverkon protokolla määrittelee kielen, jolla verkon solmut kommunikoivat keskenään. Tämä kieli koostuu tietystä joukosta komentoja, joiden avulla solmut tekevät pyyntöjä naapureilleen. Usein komentoihin

kuuluu yksi tai useampi parametri, jolloin tiedot kapseloidaan jonkinlaisen viestirakenteen sisään.

Tästä eteenpäin *viesti*-termillä tarkoitetaan juuri tätä tietorakennetta, jonka avulla verkon solmut vaihtavat tietoa.

Yksinkertaisimmillaan vertaisverkon protokollan tarvitsee sisältää pelkästään seuraavat operaatiot:

- Yhteyden avaus ja sulkeminen
- Hakuviestin lähetys eteenpäin
- Vastausviestin reititys takaisin haun lähettäjälle

Vertaisverkon toiminnan perusedellytyksenä on viestien välitys. Tästä seuraa, että yksittäiselle verkon solmulle asetettu minimivaatimus on sen kyky lähettää saamansa viesti eteenpäin vähintään yhdelle naapurilleen. Näin verkkoinfrastruktuuri saadaan itseään palvelevaksi ja toimivaksi. Käytännössä tämä vaatimus ei kuitenkaan riitä, vaan solmujen on toimittava älykkäämmin, sillä kuten edellä mainittiinkin, vertaisverkon topologia on erittäin monimutkainen ja vaihtuva. Tämä aiheuttaa solmuille ylimääräistä kuormaa, sillä niiden on jatkuvasti pidettävä kirjaa välittämästään tiedosta. Toisin sanoen jokaisella solmulla täytyy olla eräänlainen reititystaulukko, josta voidaan tarkastaa mistä kukin viesti on vastaanotettu ja minne se on lähetetty. Yleensä yksittäinen reititystaulun merkintä sisältää ainakin seuraavat tiedot:

- Viestin tunniste (viesti-id)
- Solmun tunniste, jolta viesti saatiin alunperin
- Lista solmuista, joilta viesti on saatu myöhemmin
- Lista solmuista, jonne viesti lähetettiin edelleen
- Aikaleima, jolloin viesti viimeksi nähtiin

Solmun tehtävänä on luoda jokaiselle uudelle vastaanotetulle viestille tällainen reititystietue ja päivittää sitä aina, kun se saa kyseisen viestin käsiteltäväkseen. On nimittäin hyvinkin mahdollista, että verkko sisältää silmukoita, jolloin solmu tulee hyvin todennäköisesti saamaan kopioita jo kerran käsittelemästään viestistä. Algoritmista riippuen nämä kopiot voidaan tunnistaa kyseisen reititystietueen avulla ja jättää tarvittaessa käsittelemättä.

Koska yksittäisellä solmulla ei ole käsitystä verkon rakenteesta, kyselyä lähettäessään se ei voi myöskään mitenkään ennalta tietää reittiä, mitä kautta viesti etenee verkossa. Näin ollen yksittäinen solmu ei voi itse paljoa vaikuttaa haun tehokkuuteen. Hakualgoritmin tehokkuus perustuukin solmujen yhteistoimintaan ja haun nopeus on sen käsittelyyn osallistuneiden solmujen nopeuksien summa.

Jatkuvasti muokkautuvassa verkossa on myös mahdollista että viesti katoaa. Yksittäiset solmut voivat poistua verkosta millä hetkellä hyvänsä, jolloin yhteyksiä katkeaa. Tällöin voi käydä niin, että vaikka kyselyn mukaisia resursseja löytyykin, vastausviestit eivät löydä tietä takaisin kyselyn lähettäneelle solmulle.

3.4.2 Resurssien haku

Koska vertaisverkot itsessään toimivat vain työkaluna varsinaiselle sovellukselle, ne eivät ota kantaa haun kohteeseen. Tällainen kohde voi olla vaikkapa tiedosto kiintolevyllä, prosessoriaikaa tarjoava tietokone tai henkilö, joka haluaa keskustella muiden verkossa olevien henkilöiden kanssa. Siksi määrittelemmekin termin *resurssi*, jolla tästedes viitataan tällaiseen vertaisverkkoa käyttävän sovelluksen kohteeseen.

Tietomäärän lisääntyessä ongelmaksi muodostuu olennaisen tiedon löytäminen. Esimerkiksi WWW-palvelun käyttö on tänä päivänä lähes mahdotonta ilman hakurobottien apua. Sama ilmiö pätee myös vertaisverkkojen tapauksessa. Kun verkon koko paisuu satojen tuhansien solmujen kokoiseksi, yksittäisten resurssien löytäminen valtavasta tietomäärästä on epätoivoista.

Vertaisverkoissa resurssien haku tapahtuu kyselyjen avulla. Solmu lähettää verkkoon viestin, jossa se ilmaisee mielenkiintonsa tietynlaiseen resurssiin. Tätä viestiä kuljetetaan eteenpäin ennalta määritellyn hakualgoritmin mukaisesti, jolloin se leviää solmulta toiselle. Jokainen solmu, joka saa kyseisen viestin, on velvollinen tarkastamaan oman resurssikokoelmansa ja tutkimaan, löytyykö siitä kyselyä vastaavaa resurssia. Algoritmikohtaiset viestin välitysmekanismit on esitelty seuraavassa kappaleessa 4.

Voidaan myös olettaa, että osa verkon solmuista ei toteutakaan sovittua hakualgoritmia. Vertaisverkoissa tämäkään ei ole suuri ongelma, kunhan nämä poikkeavat solmut toteuttavat edes jonkun hakustrategian. Esimerkiksi verkon toimintaan dynaamisesti adaptoituva NeuroSearch-algoritmi [28] toimisi luultavasti aivan mainiosti missä tahansa muuta algoritmia käyttävässä verkossa.

4 Hakualgoritmit

Edellä mainittiin hakualgoritmien olevan yksi tärkeimmistä vertaisverkon toimintaan ja suorituskykyyn vaikuttavista tekijöistä. Tässä kappaleessa syvennytään näiden algoritmien toimintaperiaatteisiin ja selvitetään niiden eroavaisuuksia.

Hakualgoritmin tehtävänä on toimia solmun viestiliikenteen ohjaajana ja pyrkiä optimoimaan verkon toimintaa. Verkon solmujen määrän kasvaessa myös niiden välinen viestiliikenne lisääntyy, jolloin ongelmaksi muodostuu riittämätön tiedonsiirtokaista. Samalla resurssien etsiminen vaikeutuu. Algoritmin tulisi toimia kaistaa säästellen ja kyetä silti löytämään hakuun vastaavia resursseja mahdollisimman tehokkaasti. Liiallinen viestiliikenteen vähentäminen kuitenkin alentaa resurssien löytymisprosenttia, joten algoritmin tulisi löytää kompromissi näiden kahden kriteerin välimaastosta.

Hakualgoritmin tehtävänä on myös käsitellä verkon virhetilanteita. Sen tulee muun muassa osata käsitellä tilanteet, jossa naapuri lähettää viallisia viestejä tai muuten käyttäytyy virheellisesti.

4.1 Algoritmien toimintaperiaate

Algoritmien perusideana on löytää paras naapurisolmu (tai solmuja), jolle viestin edelleenlähettäminen todennäköisimmin aiheuttaa haetun resurssin löytymisen. Tämä yksinkertaiselta kuulostava tehtävä kuitenkin jakaa algoritmit useaan alikategoriaan valintastrategiansa perusteella, esimerkiksi seuraavasti:

- Pyritään monistamaan viestiä. Tällöin se saavuttaa lyhyessä ajassa isomman joukon solmuja ja todennäköisyys resurssin löytymiselle kasvaa.
- Pyritään ohjaamaan viestiä kohti verkon ydinalueita. Tällä tavalla viesti löytää vähemmällä askelmäärällä isomman joukon potentiaalisia resurssin omaavia solmuja.
- Viestin vastaanottaja valitaan ennaltamääräteltyjen hyvyyskriteereiden perusteella. Tämä strategia luottaa verkon historiatietoihin ja tilastollisiin todennäköisyyksiin.
- Vastaanottajaksi valitaan solmu, joka todennäköisimmin aiheuttaa vähiten viestiliikennettä. Tässä strategiassa painopiste on verkon liikenteen minimoimisessa.

Algoritmin toimintaan vaikuttaa sen käyttötarkoitus. Mikäli verkon tiedetään olevan todella laaja ja haetun resurssin erittäin harvinainen, algoritmin valintakriteerit ovat erilaiset kuin etsittäessä resurssia, jonka tiedetään olevan melko yleinen kyseisessä verkossa.

Mikäli solmu omistaa resurssin, jota vastaanotetulla viestillä etsitään, sen tulee luoda uusi vastausviesti, joka sisältää resurssin käyttöön tarvittavat tiedot. Tämä viesti lähetetään takaisin samalle solmulle, jolta hakuviesti vastaanotettiin. Edelleen, algoritmin vastuulla on reitittää nämä vastausviestit takaisin haun aloittajalle takaisin.

4.1.1 Viestien reititys

Algoritmeilla on käytössään tietty valikoima informaatiota, jonka pohjalta ne tekevät päätöksiään. Kappaleessa 3.4.1 esitellyn reititystiedon lisäksi yleisimmin käyttyjä parametreja ovat:

- Naapurisolmujen kaiutusaika (kuvaa yhteyden nopeutta)
- Naapurisolmujen asteluku, eli yhteyksien määrä
- Naapurisolmuilta saatujen vastausten lukumäärä

Aiemmin mainitun reititystaulukon avulla algoritmi voi pitää kirjaa solmun kautta kulkeneista viesteistä ja reitittää haun vastausviestit takaisin alkuperäiselle lähettäjälle. Vastausviestejä ei siis käsitellä samalla tavalla kuin hakuviestejä, vaan solmu katsoo reititystaulukostaan viestiä vastaavan hakuviestin alkuperäisen lähettäjän ja ohjaa viestin suoraan tälle solmulle.

Lisäksi jokaisella algoritmilla voi olla joukko omia muuttujiaan, joita se voi tarvittaessa jopa jakaa naapurisolmujen kesken optimoidessaan verkon toimintaa ja suorituskykyä.

Yhteistä useimmille algoritmeille on niiden kyky tuhota useampaan kertaan solmulle saapuvat viestit. Näin voi tapahtua, mikäli verkkotopologiassa on kehä, jolloin edelleenlähetetty viesti voi palata solmulle takaisin eri yhteyden kautta. Viestin tuhoamisen sijaan algoritmi voi myös reagoida kyseiseen tilanteeseen myös valitsemalla uuden vastaanottajan ja lähettämällä viestin edelleen.

Kukin viesti sisältää yleensä ainutlaatuisen satunnaisluvun eli *viesti-id*:n, jolla yksittäinen viesti tunnistetaan kaikesta viestiliikenteestä. Niinikään jokainen verkon

solmu tunnistetaan oman id:nsä avulla. Solmu tallentaa viestiliikenteensä näitä tunnuksia käyttäen. Vastausviestin id:ksi kopioidaan aina hakuviestin id. Tällöin se voidaan palauttaa takaisin samaa reittiä.

4.1.2 Ongelmakohtia

Vastausviestien katoaminen. Vertaisverkon dynaamisuus aiheuttaa ongelmia paluuviestien reitityksen. Verkosta poistuva solmu on saattanut välittää hakuviestin, johon naapurisolmu on saanut vastauksen. Tämän vastausviestin takaisinvälitys kuitenkin epäonnistuu katkenneen paluureitin seurauksena. Kyseisen ongelman voisi poistaa esimerkiksi lisäämällä hakualgoritmiin toiminnon, jolla verkosta poistuva solmu voi julkaista reititystaulukkonsa naapurisolmuilleen ennen yhteyksien katkaisemista. Tällä tavalla naapurisolmut voivat tarvittaessa luoda korvaavat yhteydet poistuvien tilalle.

Tiedonsiirron optimointi. Kuten edellä mainittiin, hakualgoritmin on tehtävä kompromissi viestiliikenteen määrän ja resurssien löytymisen välillä. Paljon viestejä tuottava algoritmi ei skaalaudu verkon koon kasvaessa, vaan tiedonsiirtokanava aiheuttaa pullonkaulan algoritmin toiminnalle. Siksi onkin olennaista, että uusia hakualgoritmeja testataan sekä suurilla että pienillä solmujen määrillä. Algoritmien optimoinnista kerrotaan lisää kappaleessa 8.

4.2 Määritelmiä

Seuraavassa taulukossa määritellään termit, joita käytetään tulevissa esimerkeissä.

$Origin(p, m)$	Solmu, josta p vastaanotti viestin m ensimmäisenä.
$Sender(p, m)$	Solmu, josta p vastaanotti viestin m viimeksi.
$Senders(p, m)$	Solmut, joista solmu p on vastaanottanut viestin m.
$LastSender(Q, p, m)$	Joukon Q solmu, josta solmu p vastaanotti viestin m viimeksi.
$Sent(p, m)$	Solmut, joille solmu p on lähettänyt viestin m.
$Neighbors(p)$	Solmun p naapurisolmut.
$Type(m)$	Viestin m tyyppi. Mahdolliset arvot: $Query$ tai $Reply$.
$Hops(m)$	Viestin m kulkemien askelien lukumäärä.
$TTL(m)$	Viestin m Time-To-Live -arvo.
$Find(p, m)$	Solmussa p hakuviestiin m vastaavien resurssien joukko.
$Overload(p)$	Solmun p kuormitusarvo.
$Random(Q)$	Solmujen joukosta Q satunnaisesti valittu solmu.
$HighestDegree(Q)$	Solmujen joukon Q suurimman asteluvun omaava solmu.
$Needed(m)$	Viestin m sisältämä *needed*-parametrin arvo.

4.3 Leveyshaku (Breadth-First Search (BFS))

BFS [15, s. 3] on kaikkein yksinkertaisin hakualgoritmi, joka monistaa saamansa kyselyn ja lähettää sen edelleen kaikille naapurisolmuilleen. Tämä algoritmi on tullut tutuksi Gnutella-sovelluksen [12] myötä, sillä se käytti ainoastaan BFS-hakua resurssien etsimiseen.

Periaatteessa tällainen menettely on erittäin tehokas, sillä lähetetty hakuviesti saavuttaa maksimaalisen määrän potentiaalisia resurssin haltijoita. Samalla algoritmi kuitenkin aiheuttaa suunnattoman määrän verkkoliikennettä, jota juuri halutaan minimoida. Lähetettyjen viestien määrä kasvaa eksponentiaalisesti suhteessa viestin kulkemaan matkaan, joten yksittäinen viesti monistuu räjähdysmäisesti tukkien hitaimmat verkkoyhteydet. Jordan Ritter on tutkinut Gnutella-verkkojen skaalautuvuusongelmaa julkaisussaan “Why Gnutella Can’t Scale. No, Really.” [21].

Käytännössä tilanne ei kuitenkaan ole niin paha, kuin Ritterin tutkimus antaa ymmärtää. BFS-algoritmi jättää käsittelemättä jo kertaalleen vastaanottamansa viestit, mikä hieman rajoittaa viestiliikenteen määrää. Lisäksi pienehköissä verkoissa viestin monistuminen pysyy hallinnassa melko hyvin. R. ja G. Schollmeier esittävät julkaisussaan ”Why peer-to-peer (P2P) does scale: an analysis of P2P traffic patterns” [23] Ritterin väitteille täysin päinvastaisia tutkimustuloksia. Heidän mukaansa Gnutella-verkon kasvaessa viestiliikenteen määrä ei kasvakaan eksponentiaali-

sesti, ja vertaisverkko skaalautuukin odotettua paremmin.

BFS-algoritmin viesti rakentuu seuraavanlaisesta informaatiosta:

- **Viestin tunniste (viesti-id)** on 32 merkkiä pitkä satunnainen merkkijono, jonka avulla yksittäinen viesti voidaan tunnistaa viestiliikenteen joukosta. Tätä tunnistetta käytetään viestin reitityksessä.
- **Viestin tyyppi** kertoo, onko kyseessä kysely vai vastausviesti. Tätä tietoa tarvitaan viestin reitityksessä eteenpäin.
- **Kuljettu matka (Hops)** on kasvava numeerinen muuttuja. Tämä arvo ilmaisee viestin kulkeman matkan, toisin sanoen kuljettujen solmujen lukumäärän.
- **Elinkaaritieto (Time-To-Live, TTL)** ilmaisee etäisyyden, kuinka kauas viesti saa korkeintaan kulkea. Tämän tiedon avulla viesti ei jää harhailemaan verkkoon loputtomasti, vaan se lopetetaan tietyn matkan kuljettuaan.
- **Kyselyn avainsanat (Keywords)** sisältävät itse kyselytiedon, eli kuvaavat resurssin, jota ollaan etsimässä.

4.3.1 Reititys

Aluksi BFS-algoritmi tarkastaa, löytyykö haettu resurssi kyseisestä solmusta. Mikäli näin on, luodaan vastausviesti ja lähetetään se takaisin hakuviestin lähettäneelle solmulle. Tämän jälkeen tutkitaan viestin elinkaaritieto, ja mikäli viestin kulkema matka on pienempi kuin TTL-arvo, lähetetään se eteenpäin. Muussa tapauksessa viesti on tullut elinkaarensa päähän ja se voidaan jättää käsittelemättä.

Lista viestin vastaanottajista saadaan vähentämällä naapurisolmujen listasta viestin lähettäjä sekä solmut, joille viesti on jo lähetetty aiemmin. Mikäli tämä solmujen joukko on tyhjä, viesti voidaan tuhota.

Jos vastaanotettu viesti oli $Reply$-tyyppinen, se ohjataan edelleen vastaavan kyselyn lähettäneelle naapurille. Tämän solmun tiedot saadaan algoritmin ylläpitämästä reititystaulusta.

Algoritmi 1 BFS-algoritmin viestin reititysmekanismi

BFS-Routing(Peer p, Message m)
  if $Hops(m) + 1 < TTL(m)$ **then**
    $Hops(m) = Hops(m) + 1$

if $Type(m) = Query$ **then**
if $Find(p, m) \neq \emptyset$ **then**
Create new reply message r containing found resource(s)
$TTL(r) = Hops(m)$
$Hops(r) = 0$
Send message r to $Origin(p, m)$
end if
$Q = Neighbors(p) \setminus (Senders(p, m) \cup Sent(p, m))$
if $Q \neq \emptyset$ **then**
Forward message m to all nodes in Q
$Sent(p, m) = Sent(p, m) \cup Q$
end if
else
Send message m to $Origin(p, m)$
end if
end if

4.4 Adaptive Breadth-First Search (ABFS)

Alkuperäisen BFS-algoritmin kaistanleveyden käyttö aiheuttaa ongelmia etenkin hitailla yhteyksillä. Tämän ongelman ratkaisemiseksi kyseistä algoritmia on jatkokehitetty lisäämällä siihen logiikkaa, joka osaa hillitä massiivista tietovirtaa rajoittamalla yhteyksien välistä liikennöintiä. Kun verkon solmu huomaa ylikuormittuvansa, se osaa informoida naapureitansa kyseisestä tilanteesta ja näin naapurisolmut voivat muokata reititystaulujaan ja vähentää kuormittuneelle solmulle ohjattavaa dataliikenteen määrää. Tästä algoritmista ei löytynyt olemassaolevaa toteutusta, eikä kyseistä mekanismia ole juurikaan tutkittu, joten tässä kappaleessa esiteltävää menetelmää voidaan pitää uutena vertaisverkon hakualgoritmitoteutuksena. Se syntyi osana tätä pro gradu -tutkimusta yhteistyössä Cheese Factory [8] -projektin kanssa.

ABFS-algoritmin viesti rakentuu seuraavanlaisesta informaatiosta:

- **Viestin tunniste**
- **Viestin tyyppi**

- **Kuljettu matka**
- **Elinkaaritieto**
- **Kyselyn avainsanat**

4.4.1 Reititys

Reitityksen perusmekanismi on sama kuin BFS-algoritmissakin. Lisäksi käytössä on ylikuormituksen estomekanismi, joka perustuu viestin kulkemaan matkaan. Kuormituksen lisääntyessä solmu vähentää tätä ylikuormitusarvoa (engl. *Overload limit*) ja ilmoittaa uuden arvon naapureilleen, jolloin ne alkavat rajoittamaan viestien välitystä kyseiselle solmulle. Tämä rajoitus tapahtuu rajaamalla viestien edelleenlähetys sellaisiin viesteihin, joiden sillä hetkellä kulkema matka on pienempi kuin tämä ylikuormitusarvo. Tällä tavoin solmu ilmoittaa haluavansa käsitellä vain tietyn etäisyyden päässä olevien solmujen lähettämiä viestejä.

Kun uusi solmu liittyy ABFS-algoritmia käyttävään verkkoon, se ilmoittaa oman ylikuormitusarvonsa naapurisolmuilleen. Nämä puolestaan vastaavat ilmoitukseen kertomalla omat kuormitusarvonsa uudelle solmulle. Tämän jälkeen solmu on valmis käsittelemään verkon liikennettä. Kuormitustilanteen muuttuessa solmu ilmoittaa naapureilleen kuormitusarvon muutoksista lähettämällä niille protokollakohtaisen ohjausviestin, joka sisältää muuttuneen arvon.

Algoritmi 2 ABFS-algoritmin viestin reititysmekanismi

ABFS-Routing(Peer p, Message m)
if $Hops(m) + 1 < TTL(m)$ **then**
$Hops(m) = Hops(m) + 1$
if $Type(m) = Query$ **then**
if $Find(p, m) \neq \emptyset$ **then**
Create new reply message r containing found resource(s)
$TTL(r) = Hops(m)$
$Hops(r) = 0$
Send message r to $Origin(p, m)$
end if
$Q = Neighbors(p) \setminus (Senders(p, m) \cup Sent(p, m))$
if $Q \neq \emptyset$ **then**

```
        for each q_i in Q do
          if Overload(q_i) < Hops(m) then
            Forward message m to q_i
            Sent(p, m) = Sent(p, m) ∪ {q_i}
          end if
        end for
      end if
    else
      Send message m to Origin(p, m)
    end if
  end if
```

Algoritmi itsessään ei ota kantaa siihen, milloin solmu on ylikuormittunut. Kyseisenä tietona voidaan käyttää esimerkiksi solmukoneen prosessorikuormaa tai verkkoliikenteen määrää. Tämä rajoite kuuluukin algoritmin toteuttajan määriteltäviin asioihin.

4.5 Satunnaiskävelijä (Random Walker Search (RWALK))

Random walker -algoritmissa [1] verkkoon lähetetään ainoastaan yksi kyselyviesti. Tämän viestin tarkoituksena on algoritmin nimen mukaisesti harhailla satunnaisesti verkossa solmulta toiselle. Tällainen ratkaisu kuluttaa huomattavasti vähemmän tiedonsiirtokaistaa kuin aiemmin mainitut algoritmit, sillä jos etsittävästä resurssista on useita ilmentymiä verkossa, haku päättyy heti ensimmäisen resurssin löytyessä ja ylimääräisiä viestejä ei enää lähetetä.

Haittapuolena tällaisessa lähestymistavassa on haun pitkäkestoisuus [11]. Hakuviesti voi harhailla verkossa pitkään, ennenkuin se löytää haluamansa resurssin. Tämän algoritmin hyöty tuleekin parhaiten esiin verkoissa, joissa kaistanleveys on erittäin rajoitettu tai joissa hakukriteerinä on yhdenkin resurssin löytyminen.

RWALK-algoritmin viesti rakentuu seuraavanlaisesta informaatiosta:

- **Viestin tunniste**
- **Viestin tyyppi**
- **Kuljettu matka**. Tätä käytetään ainoastaan viestin elinkaaren seurantaan, eli se ei vaikuta hakualgoritmiin.

- **Kyselyn avainsanat**
- **Haettavien resurssien määrä (Needed)** pienenee yhdellä aina, kun viesti löytää haetun resurssin. Tällä tavalla haku saadaan päättymään tietyn resurssimäärän löydyttyä.

4.5.1 Reititys

Random walkerin reititys on hieman älykkäämpi kuin edellisten algoritmien, sillä hakuviestin sallitaan palaavan kulkemaansa hakupolkua takaisinpäin. Tällä tavalla vältetään viestin jääminen umpikujaan. Mikäli viesti kulkeutuu lehtisolmuun, sitä ei tuhota, vaan solmu lähettää viestin takaisinpäin ja edellinen solmu osaa ohjata viestin toiseen suuntaan. Sama mekanismi toimii takautuvasti myös aiemmissa solmuissa, joten käytännössä viestin voidaan ajatella kulkevan rekursiivisesti verkkoa läpi.

Kyseisestä reititysmekanismista johtuen haun vanheneminen täytyy toteuttaa sovelluskohtaisesti verkon käyttötarkoituksen perusteella. Algoritmiin täytyy määritellä ehdot, milloin haku päättyy. Ehtoina voidaan käyttää vaikkapa TTL-arvoa tai aikaleimaa. Tässä tapauksessa viestin lopetusehtona käytetään hakuviestin löytämien resurssien määrää; viesti voidaan tuhota, kun sen *needed*-parametri menee nollaksi.

Algoritmi 3 RWALK-algoritmin viestin reititysmekanismi

Random-Walker-Routing(Peer p, Message m)
if $Type(m) = Query$ **then**
if $Find(p, m) \neq \emptyset$ **then**
$Needed(m) = Needed(m) - 1$
Create new reply message r containing found resource(s)
Send message r to $Origin(p, m)$
end if
if $Needed(m) > 0$ **then**
Q=$Neighbors(p) \setminus \{Sender(p, m)\}$
if $Q = \emptyset$ **then**
$q = Origin(p, m)$
else
$q = Random(Q)$

end if
Forward message m to q
end if
else
Send message m to $Origin(p, m)$
end if

4.6 Korkeimman asteen haku (Highest Degree Search (HDEG))

Tätä algoritmia voitaneen pitää optimoituna versiona Random Walker -algoritmista. Se niinikään lähettää verkkoon yhden kyselyviestin, mutta viestin reititys ja vastaanottajan valinta tapahtuu eri tavalla.

HDEG-algoritmissa [1, s. 3] olennaisena osana toimii solmun naapurien määrä, toisin sanoen solmun *asteluku*. Satunnaisen valinnan sijaan viestin vastaanottajaksi määritellään aina naapurisolmu, jolla on suurin asteluku. Jotta tämä olisi mahdollista, protokollan on osattava välittää tätä tietoa naapurisolmujen kesken. Solmut kyselevät naapuriensa asteluvun omalla *degree*-komennollaan.

Etenkin potenssijakautuneessa verkossa tällaisen algoritmin voidaan olettaa toimivan tehokkaasti, sillä se pyrkii ensin etsimään verkon runkosolmut, joilla on paljon naapureita. Vasta kun nämä ydinalueet on käyty läpi, aletaan viestiä reitittämään vähemmän yhteyksiä omaaville reunasolmuille. Tällöin saavutetaan suurempi resurssien löytymisprosentti.

Highest Degree Search -algoritmista on olemassa myös toinen versio, joka sisältää piirteitä BFS-algoritmista. Tässä versiossa solmu voi halutessaan monistaa saapuneen hakuviestin useammalle vastaanottajalle, mikäli tietyt kriteerit täyttyvät. Tällainen kriteeri voi olla vaikkapa solmun asteluku. Kun viesti löytää tiensä verkon runkorakenteeseen, se monistuu ja näinollen haku tehostuu [19].

HDEG-algoritmin viesti rakentuu seuraavanlaisesta informaatiosta:

- **Viestin tunniste**
- **Viestin tyyppi**
- **Kyselyn avainsanat**
- **Haettavien resurssien määrä**

4.6.1 Reititys

Highest Degree Search -algoritmin viesti kulkee solmulta toiselle hyvin samankaltaisella tavalla kuin Random Walker -algoritmissakin. Verkossa vaeltaa vain yksi hakuviesti, mutta sen kulkua ohjaa verkon topologia. Viesti tuhotaan, kun sen *needed*-parametri menee nollaksi. Koska HDEG-algoritmin määritelmä on epätarkka, siitä on olemassa monta vaihtoehtoista toteutusta. Alla on esitelty eräs HDEG-algoritmin viestinvälitysmekanismin muunnelmista, jonka kappaleessa 5 esiteltävä P2PCore-sovellusalusta toteuttaa:

Algoritmi 4 HDEG-algoritmin viestin reititysmekanismi

Highest-Degree-Routing(Peer p, Message m)
if $Type(m) = Query$ **then**
if $Find(p, m) \neq \emptyset$ **then**
$Needed(m) = Needed(m) - 1$
Create new reply message r containing found resource(s)
Send message r to $Origin(p, m)$
end if
if $Needed(m) > 0$ **then**
$Q = Neighbors(p) \setminus (Senders(p, m) \cup Sent(p, m))$
if $Q \neq \emptyset$ **then**
$q = HighestDegree(Q)$
else
$Q = Senders(p, m) \setminus Sent(p, m)$
if $Q \neq \emptyset$ **then**
$q = LastSender(Q, p, m)$
else
$q = Origin(p, m)$
end if
end if
Forward message m to q
$Sent(p, m) = Sent(p, m) \cup \{q\}$
end if
else
Send message m to $Origin(p, m)$

end if

Ymmärtääksemme paremmin HDEG-algoritmin viestinvälitysmekanismia, määrittelemme seuraavat termit viestin m kontekstissa:

- **Musta solmu** on vastaanottanut viestin m joltakin naapuriltaan ja välittänyt sen edelleen kaikille mahdollisille vastaanottajille. Toisin sanoen sillä ei ole enää vaihtoehtoja, minne viesti voitaisiin lähettää edelleen. Yllä olevaa merkintätapaa käyttäen solmun p mustat naapurisolmut Q saadaan kaavalla

$$Q = Senders(p, m) \cap Sent(p, m)$$

- **Harmaa solmu** on käsitellyt viestin m ja ohjannut sen joillekin naapureilleen, mutta sillä on vielä muita naapureita, joille viestiä ei ole lähetetty. Yllä olevassa algoritmikoodissa solmun p harmaiden naapurisolmujen joukko Q saadaan seuraavasti:

$$Q = Senders(p, m) \setminus Sent(p, m)$$

  Tämän määritelmän mukaan harmaa joukko siis sisältää ne solmut, joilta viesti on jo aiemmin vastaanotettu, poislukien mustat solmut.

- **Valkoinen solmu** ei ole vielä koskaan vastaanottanut viestiä m. Yllä olevassa reititysmekanismissa solmun p valkoisten naapurisolmujen joukko Q saadaan kaavalla

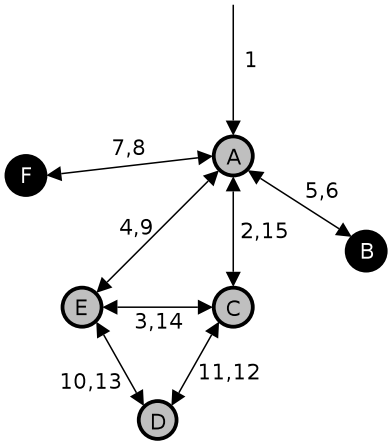
$$Q = Neighbors(p) \setminus (Senders(p, m) \cup Sent(p, m))$$

Näiden termien avulla kussakin solmussa voidaan määritellä naapurisolmujen tiloja. Jokainen solmu pitää omaa erillistä kirjanpitoaan, eivätkä ne tiedä toistensa kirjanpidosta mitään, tai jaa tätä tietoa keskenään.

Viestin vastaanottajan valinnassa puolestaan pätee seuraavat säännöt:

- Viesti lähetetään aina valkoiselle solmulle, mikäli mahdollista.
- Mikäli valkoisia vastaanottajakandidaatteja on useampi, valitaan niistä se solmu, jolla on suurin asteluku.

- Mikäli valkoisia vastaanottajasolmuja ei ole, viesti lähetetään harmaalle naapurisolmulle.
- Jos viesti palaa takaisin solmulta, jolle se aiemmin lähetettiin, kyseinen solmu merkitään mustaksi.
- Viestiä ei koskaan lähetetä mustalle solmulle tai solmulle, jolle se on jo kertaalleen lähetetty, paitsi jos muuta vaihtoehtoa ei ole.
- Solmu pyrkii välttämään joutumista mustalle listalle. Toisin sanoen se edelleenohjaa viestin alkuperäiselle lähettäjälle vasta, kun muita vastaanottajakandidaatteja ei ole.
- Jos solmu ei voi välttää joutumista mustalle listalle, viesti lähetetään sille (harmaalle) solmulle, jolta se vastaanotettiin viimeksi. Tällä tavalla varmistetaan viestin rekursiivinen takaisinpäin palauttaminen ja estetetään mahdollisten reitin varrella olleiden matala-asteisten solmujen löytyminen.



Kuva 4.3: Esimerkki viestin kulkemisesta HDEG-algoritmia käytettäessä.

Kuvassa 4.3 on esitetty vaiheittain viestin eteneminen pienessä, kuuden solmun kokoisessa vertaisverkossa. Viestin eteneminen tapahtuu seuraavalla tavalla:

1) Solmu A vastaanottaa hakuviestin. Tällä hetkellä kaikki A-solmun naapurit ovat valkeita solmuja, poislukien viestin lähettäjä.

2) Solmu A:n naapureista C:llä ja E:llä on suurin asteluku 3. Niinpä se valitsee näistä satunnaisesti toisen (C) ja lähettää vastaanottamansa viestin edelleen tälle. Samalla A-solmu merkitsee solmun C harmaaksi omassa kirjanpidossaan.

3) Solmu C vastaanottaa A:n lähettämän viestin. Tällöin A merkitään harmaaksi solmuksi. Näinollen C:llä on kaksi valkeata solmua D ja E, mutta E:llä on suurempi asteluku (3). Täten vastaanottajaksi valitaan solmu E.

4) E merkitsee C:n harmaaksi solmuksi. Sillä on kaksi valkoista solmua A ja D, joista A:lla on isompi asteluku. Viesti lähetetään sille ja A merkitään harmaaksi.

5) A vastaanottaa viestin E:ltä ja merkitsee E:n harmaaksi. Nyt A:lla on kaksi valkeata solmua B ja F. Vastaanottajaksi valitaan satunnaisesti solmu B.

6) B:llä ei ole naapureita, joten se ei voi muuta, kuin lähettää viesti taksisin A:lle.

7) A huomaa, että solmu B lähetti viestin takaisin. Se merkitsee solmun B mustaksi. Nyt A:lla on enää yksi valkea solmu F, jolle viesti lähetetään.

8) F:llä ei myöskään ole naapureita, joten taas viesti palaa takaisin A:lle ja solmu F merkitään mustaksi.

9) A:lla on ainoastaan harmaita naapureita, joilla on kummallakin sama asteluku. A kuitenkin huomaa, että viesti on jo kertaalleen lähetetty C:lle. Näinollen se palautetaan takaisin solmulle E.

10) E huomaa, että A palautti aiemmin lähetetyn viestin. Näin A merkitään mustaksi solmuksi. E:llä on yksi valkea naapuri D, jonne viesti ohjataan.

11) D:llä on valkea naapuri C. Viesti ohjataan sinne.

12) C:n kaikki naapurit on harmaita. Edellä mainittujen sääntöjen nojalla (C ei voi välttyä joutumasta mustalle listalle) viestin vastaanottajaksi valitaan solmu D.

13) D merkitsee solmun C mustaksi. Sen ainut vaihtoehto on lähettää viesti takaisin E:lle.

14) E merkitsee D:n mustaksi ja lähettää viestin C:lle.

15) C merkitsee E:n mustaksi. Sen ainoa harmaa naapuri on solmu A, joten viesti lähetetään A:lle. Solmu A huomaa, ettei viestiä voida lähettää minnekään ja haku päättyy.

4.7 Muita vertaisverkkoihin liittyviä menetelmiä

Tässä kappaleessa selitetään hieman keinoja, joilla verkon ja hakualgoritmin toimintaa voidaan tehostaa, sekä esitellään menetelmiä, jotka liittyvät vertaisverkkojen hallintaan.

4.7.1 Topologian hallinta-algoritmit

Jotkin algoritmit osallistuvat aktiivisesti naapurisolmujen valintaan. Ne tarkastelevat säännöllisin väliajoin listaa aukiolevista yhteyksistä ja tarvittaessa korvaavat huonoimmat yhteydet laadullisesti paremmilla. Kunkin yhteyden "huonous" ja "hyvyys" riippuu tietenkin algoritmin toteutuksesta. Jollekin algoritmille tärkein kriteeri voi olla yhteyden nopeus, kun taas toinen algoritmi arvostaa enemmän esimerkiksi löytyneiden resurssien lukumäärää. Algoritmi voi myös määritellä solmun enimmäisyhteysmäärän. Karkeasti topologian hallinta voidaan jakaa seuraaviin osa-alueisiin [4]:

- Solmun valinta (engl. *Node selection*)
- Solmun poisto (engl. *Node removal*)
- Solmun ohitus (engl. *Overtaking*)
- Kuormituksen arviointi (engl. *Overload estimation*)

Annemari Auvisen pro gradu -työ [3] keskittyy juuri näihin verkkotopologian hallinta-algoritmeihin.

4.7.2 Adaptiiviset hakualgoritmit

Kappaleessa 4.4 esiteltiin erittäin yksinkertainen adaptiivinen ABFS-hakualgoritmi, jonka toiminta muuttuu verkon solmujen kuormituksen mukaan. Tällainen menetelmä estää verkon ylikuormittumisen, muttei juurikaan vaikuta haun tehokkuuteen. Tehokkuuteen painottuvassa adaptaatiossa algoritmi voi esimerkiksi valita etupäässä solmuja, joilla on pieni kaiutusaika, eli ne ovat nopean yhteyden päässä, tai käyttää muita saatavilla olevia jatkuvasti muuttuvia verkon parametreja suorituskykynsä optimointiin. Muita adaptiivisia hakualgoritmeja ovat esimerkiksi adaptiivinen probabilistinen hakualgoritmi (APS) [26] ja Jyväskylän yliopistossa kehitetty NeuroSearch-hakualgoritmi [28].

4.7.3 Verkon historiatiedon hyödyntäminen

Vertaisverkkoa käytettäessä solmut keräävät tilastotietoa naapureistaan. Kuten kappaleessa 4.1.1 mainittiin, esimerkiksi naapurin palauttamien vastausviestien määrä

on yksi tällainen tilastollisesti kiinnostava tieto. Muun muassa suunnattu leveyshaku (Directed Breadth-First-Search, DBFS) käyttää tätä informaatiota hyödykseen valitessaan viestin vastaanottajasolmuja. Myös edellä mainitut topologian hallintaalgoritmit ja adaptiiviset menetelmät voivat käyttää historiatietoa toimintansa ohjaamiseen.

5 Tutkimusympäristö

Tämän pro gradu -tutkielman osana suunniteltiin ja kehitettiin P2PCore-vertaisverkkoalusta, joka toteutettiin Java-ohjelmointikielellä. Kyseistä alustaa käytettiin seuraavassa kappaleessa esiteltävien tutkimustulosten tuottamiseen. P2PCore ei kuitenkaan ole pelkkä verkkosimulaattori, vaan sen päälle voitaisiin rakentaa hyvin helposti esimerkiksi nykyisiä tiedostonjakosovelluksia vastaava vertaisverkkosovellus.

5.1 P2PCore

P2PCore on sovellusalusta, joka tarjoaa vertaisverkkopalvelut sitä käyttävälle sovellukselle. Se piilottaa verkonhallintaan liittyvät ongelmat ja toteutuksen yksinkertaisen ohjelmointirajapinnan taakse, joten sovelluskehittäjä voi keskittyä itse toteutukseen ja antaa P2PCoren huolehtia verkkoliikenteen hoitamisesta.

Seuraavissa kappaleissa kuvataan P2PCoren ominaisuuksia ja käytettyjä ohjelmistokomponentteja.

5.1.1 Ominaisuudet

P2PCore-sovellus käyttää tehokkaasti hyväkseen säikeistystä. Jokainen etäyhteys käsitellään itsenäisesti omassa säikeessään, jolloin saavutetaan yhteyksien välinen riippumattomuus ja parempi vasteaika viestien käsittelyyn kuin yksisäikeisellä mallilla toteutettuna. Toki säikeiden käyttö aiheuttaa ylimääräistä kuormitusta itse Java-virtuaalikoneelle, mutta tämän asian tutkiminen ja optimointi menee asiayhteyden ulkopuolelle.

Sovellus koostuu erillisestä ytimestä ja dynaamisesti ladattavista algoritmimoduleista. Lataaminen tapahtuu Javan reflection-rajapinnan avulla, ja alustaan voidaan

lisätä uusia hakualgoritmeja ilman mitään muutoksia itse ytimeen. Algoritmimodulit toteuttavat tietyn rajapintamäärityksen, jonka kautta ydin pystyy käyttämään ulkopuolelta tuotuja laajennuskomponentteja (vrt. selainten plugin-rajapinta). P2PCore voi ajaa useita algoritmimoduleja samanaikaisesti, jolloin sama sovellus voi toimia asiakkaana usean tyyppiseen verkkoon samanaikaisesti. Pienillä muutoksilla P2PCoreen olisi myös mahdollista kehittää erityisiä siltausmoduleita, jotka osaisivat yhdistää useita eri hakualgoritmeja käyttävät verkot toisiinsa.

Ydin sisältää tarvittavat toiminnot yhteyksien ylläpitoon, viestien välitykseen, resurssien käsittelyyn ja algoritmimodulien hallintaan. Se myös kerää statistiikkaa verkkoliikenteestä ja tarjoaa tätä informaatiota algoritmimodulien käyttöön. Ydin esimerkiksi huolehtii viestien tunnistetietojen (viesti-id) säilömisestä ja tämän tietovaraston ylläpidosta.

P2PCoren kaikki asetukset on koottu yhteen konfiguraatiotiedostoon, joka ladataan käynnistyksen yhteydessä. Tämä tiedosto sisältää kaikki käyttäjän muutettavissa olevat asetukset, käytettävät hakualgoritmit ja algoritmikohtaiset erikoisasetukset.

Seuraavassa kappaleessa esitellään tarkemmin ytimeen kuuluvat ohjelmistokomponentit ja niiden käyttötarkoitus.

5.1.2 Ohjelmistokomponentit

P2PCore koostuu useista alikomponenteista, jotka ovat erikoistuneet hoitamaan tiettyä toiminnallista osa-aluetta.

P2PServer käynnistää oman säikeen, jonka tehtävänä on vastaanottaa sisääntulevat yhteydenottopyynnöt, muodostaa näistä yhteyksistä *P2PConnection*-olioita ja delegoida jatkotoimenpiteet *Connection Manager*-komponentille.

Command Handler tulkitsee verkosta vastaanotetut viestit ja muodostaa niistä erillisiä komento-olioita. XML-muotoiset viestit muutetaan Javan natiivi-olioiksi ja ohjataan edelleen komennosta vastaavalle komponentille. Komentojen lisäys tälle käsittelijälle tapahtuu dynaamisesti, toisin sanoen mikä tahansa komponentti, joka toteuttaa *P2PCommandReceiver* -rajapinnan voi halutessaan rekisteröidä haluamiaan komentoja ja näin laajentaa perusprotokollaa.

Connection Manager huolehtii yhteyksien ylläpidosta. Se sisältää listan auki olevista yhteyskanavista ja kerää tietoliikenteeseen liittyvää tilastotietoa, kuten esimerkiksi siirrettyjen viestien ja tavujen määrän yhteyskohtaisesti.

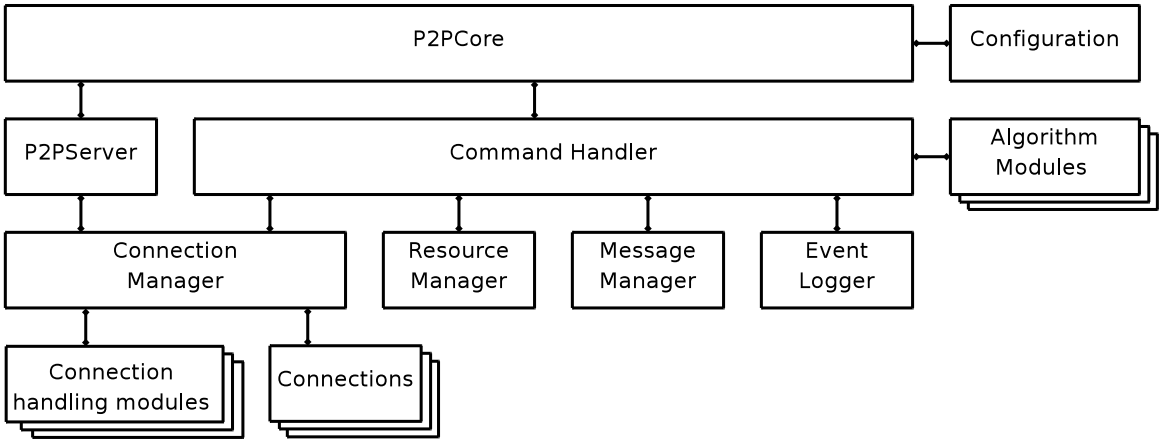
Resource Manager hallinnoi jaettavia resursseja sekä vastaa resurssikyselyihin.

Event Logger taltioi kaikki komennot, joita kyseinen solmu joko vastaanottaa tai lähettää. Tämän komponentin avulla asiakassovellus voi tarkkailla P2PCoren toimintaa ja näyttää tilastotietoa käyttäjälle.

Message Cache Manager pitää kirjaa solmun viestiliikenteestä. Se taltioi kaikki vastaanotetut ja lähetetyt viestit sekä tiedon viestien lähettäjistä sekä vastaanottajista. Kyseinen komponentti myös pitää huolen tämän viestihistorian siivoamisesta. Viestihistoria tarkastetaan tietyin väliajoin ja vanhetuneet viestit poistetaan muistin säästämiseksi.

Algoritmimodulit toteuttavat nimensä mukaisesti varsinaiset hakualgoritmit. Ne käyttävät hyväkseen kaikkia yllä mainittuja komponentteja saadakseen tarvitsemansa tiedot viestien reititystä varten.

Kuvassa 5.4 on esitelty P2PCore-alustan arkkitehtuuri ylimmällä mahdollisella tasolla.



Kuva 5.4: Yksinkertaistettu kaavio P2PCore-alustan arkkitehtuurista.

5.1.3 Työkalut

P2PCore-alustan ohella valmistui joukko aputyökaluja verkon hallintaan ja testaamiseen.

Client ja ClientTester ovat P2PCoren esimerkinomaisia asiakassovellustoteutuksia. Client-komponentti toteuttaa yksittäisen minimaalisen asiakassovelluksen, kun taas ClientTester simuloi useita erillisiä asiakkaita luomalla halutun määrän Client-instansseja toimimaan kukin omalla konfiguraatiollaan. Näiden komponenttien avul-

la voidaan testata useiden satojen solmujen kokoista vertaisverkkoa yksittäisessä tietokoneessa.

P2PMonitor on interaktiivinen statistiikan keräilytyökalu. Se liittyy verkkoon käyttäjän määrittelemän solmun / solmujen kautta ja alkaa seuraamaan verkon liikennettä, eritoten Event Logger -komponentin lähettämiä tapahtumaviestejä. Se sisältää pienehkön graafisen käyttöliittymän, jonka avulla verkkoa hallinnoidaan. P2PMonitor-sovelluksen avulla verkkoon voidaan myös lähettää komentoja ja odottaa niihin vastauksia. P2PCore tunnistaa kyseisen komponentin yhteyspyynnöt ja rekisteröi sen automaattisesti Event Logger komponentille asiakkaaksi.

P2PTool on komentorivipohjainen versio *P2PMonitor*-työkalusta. Se tarjoaa yksinkertaisen komentokielen verkon manipulointiin ja solmujen käsittelyyn. Tämän työkalun avulla on mahdollista automatisoida verkon analysointia ja testaamista kirjoittamalla etukäteen komentojonoja ja ajamalla niitä kyseisen työkalun avulla. Näin esimerkiksi verkon topologian rakentaminen, hakujen lähettäminen ja statistiikan kerääminen verkosta saadaan automatisoitua.

TestCaseGenerator on apuväline, jonka avulla luodaan komentojonotiedostoja *P2PTool*-työkalulle. Se luo potenssijakautuneet verkkotopologiamääritykset ja kirjoittaa testikohtaiset alustustiedostot, jotka P2PTool lukee yksi kerrallaan. Tämän työkalun lähdekoodissa määritellään muun muassa verkkojen koot, halutut algoritmit, resurssien jakautuminen, viestien TTL-arvot ja testiajojen määrä per verkko. ja näistä tiedoista luodaan kaikki mahdolliset kombinaatiot. Esimerkki tämän työkalun luomasta komentojonosta löytyy liitteestä 10.4.

Shell-skriptit ovat tärkeä osa P2PCoren testausautomaatiota. Ohjelmisto sisältää joukon bash-skriptejä, joilla muun muassa ajetaan tiettyjä osajoukkoja testikokonaisuudesta, kerätään testituloksia sekä luodaan tuloksista graafeja ja yhteenvetoja. Esimerkiksi koko testimateriaalin ajaminen tuottaa noin 600 tulostiedostoa, joiden sisältö tiivistetään kuuteentoista Gnuplot-sovelluksen syötetiedostoon.

Syy, miksi testausympäristö käyttää erillisiä komentojonotiedostoja sekä erillistä komentotulkkia niiden ajamiseen on se, että minkä tahansa testin haluttiin olevan toistettavissa täysin samoilla syötteillä ja verkkorakenteella. Tästä syystä kaikki yksittäiseen testiin liittyvä informaatio täytyy olla pysyvästi määritelty. Tällöin myös ongelmatilanteiden selvittely helpottuu huomattavasti, kun sama testi voidaan ajaa uudelleen lisäämällä ohjelmakoodiin sovelluksen toiminnasta kertovia tulosterivejä ja tallentamalla P2PCoren tulostama loki tiedostoon analysointia varten.

5.1.4 Käytetyt algoritmit

P2PCore toteuttaa BFS, ABFS, DBFS, RWALK ja HDEG -algoritmit, joista ABFS ja DBFS ovat vain kokeellisia. Verkkosolmun kuormittumisen määräytymislogiikka ja todellisuuksiin pohjautuva vastaanottajan valinta eivät kuulu tämän tutkielman aihepiiriin, joten kyseiset algoritmit on jätetty testien ulkopuolelle. Uusien algoritmimodulien toteutus olisi melko suoraviivaista, mutta ensin mainitut ovat tutkimuksen kannalta oleellisimmat, sillä ne edustavat kukin omaa algoritmikategoriaansa.

5.1.5 Protokolla

Solmujen väliseksi siirtoprotokollaksi valittiin C2GP-protokolla, joka kehitettiin alunperin Cheese Factory -projektin [8] toimesta Chedar- ja Guardian-sovellusten väliseksi kommunikaatioprotokollaksi. Nimi C2GP tulee sen alkuperäisestä käyttötarkoituksestaan: *Chedar-to-Guardian Protocol*.

Tämä protokolla on XML-pohjainen ja hyvin suoraviivainen. Se sisältää sarjan komentoja, joiden avulla solmut kommunikoivat. Nämä komennot ovat:

- **CONNECT, DISCONNECT** Yhteyden avaus ja sulkeminen.
- **FORCE-CONNECTION, FORCE-DISCONNECTION** Yhteyden pakollinen avaus ja sulkeminen. Ohittaa mahdolliset tarkastukset ja topologianhallinnan.
- **PING** Naapurisolmun kaiuttaminen.
- **NEIGHBOR-LIST** Naapurisolmujen listauspyyntö.
- **RESOURCE** Resurssin etsiminen tai pyyntöön vastaaminen.

Komento voi edustaa kahta tyyppiä, jotka ovat pyyntö (*Request*) ja vastaus (*Reply*).

Edellä mainittujen C2GP-protokollan komentojen lisäksi P2PCore sisältää seuraavat testaukseen ja testaustulosten keräämiseen liittyvät komennot:

- **STATUS** Solmun tilastotietojen hakeminen.
- **DEGREE** Solmun asteluvun hakeminen.
- **NETSETUP** Nollaa solmun laskurit ja käskee sitä hakemaan naapureiden asteluvut DEGREE-komennolla.

Kukin komento sisältää joukon parametrejä, joista osa on algoritmikohtaisia. Yleisimpiä parametrejä ovat:

- **nodeid** Solmun tunniste.
- **algorithm** Käytetty algoritmi.
- **hops** Viestin kulkema matka.
- **ttl** Viestin maksimi elinaika.
- **query** Haettavan resurssin tunnistemerkkijono.

Esimerkki kahden solmun välisestä tiedonsiirrosta:

Pyyntö

```
<request>
    <connect password="salasana"/>
</request>
```

Vastaus

```
<reply>
    <connect success="true"/>
</reply>
```

5.1.6 Tiedonsiirto

P2PCore käyttää tietoliikenteeseen TCP/IP-sokettiyhteyksiä. Verkon solmut tunnistetaan IP-osoite + porttinumero -yhdistelmän avulla. Porttinumero tarkoittaa kyseisen solmun palvelinprosessin porttia, joka vastaanottaa ulkoa tulevia yhteydenmuodostuspyyntöjä.

Yhteys muodostetaan CONNECT -komennolla, jossa annetaan *password* -parametrina salasana. Yhteydenottopyynnön vastaanottanut solmu vastaa pyyntöön *Reply*-tyyppisellä CONNECT-komennolla, johon asetetaan *success*-parametri sen mukaan, hyväksyttiinkö yhteydenmuodostus vai ei.

5.1.7 Suorituskyky

P2PCore kehitettiin osana tutkimusprojektia, joten suorituskykyyn liittyvät tekijät jätettiin toissijaisiksi. Päämääränä oli saada aikaan modulaarinen, helposti ymmärrettävä vertaisverkkoalusta, jonka avulla on mahdollista toteuttaa useita vertailukelpoisia vertaisverkon hakualgoritmeja. Varsinainen toteutus on tehty Java-kielellä,

joka osaltaan vaikuttaa suorituskykyyn. Toisaalta, kyseinen alusta voitaisiin suoraan ottaa osaksi jotain käytännön sovellusta, sillä se ei aseta käytölle mitään ennakkovaatimuksia tai -olettamuksia.

5.2 Tutkimusmenetelmä

Algoritmien suorituskyvyn mittareina käytetään niiden aiheuttamaa verkkoliikenteen määrää ja haetun resurssin löytymiseen kulunutta aikaa.

Tutkimuksessa haluttiin selvittää viestien määrän ja löytymiseen kuluneen ajan välistä korrelointia sekä viestin vastaanottajan valintalogiikan parantamisen vaikutusta näihin. Lisäksi haluttiin tutkia, kuinka verkon koon kasvattaminen vaikuttaa tuloksiin.

Testiajot jaettiin kahteen eri ryhmään resurssijakauman ja hakukriteerien perusteella:

- Verkkoon oli määritelty ainoastaan yksi resurssi: Verkosta valittiin satunnaisesti yksi solmu, johon haettava resurssi asetettiin. Muut solmut eivät omistaneet yhtään resurssia. Testeissä mitattiin, kuinka tehokkaasti algoritmit löysivät tämän resurssin.
- Etsittävä resurssi asetettiin kymmeneen prosenttiin kaikista verkon solmuista, ja algoritmin tuli löytää näistä resursseista puolet. Esimerkiksi 100 solmun verkossa haettava resurssi oli asetettu 10 solmuun, ja algoritmin tuli löytää 5 ilmentymää tästä resurssista.

Kummastakin ryhmästä poimittiin seuraavat mittaustiedot:

- Haun aiheuttama viestiliikenteen määrä. Liikenteeksi lasketaan sekä hakuviestien edelleenohjaukset sekä paluuviestin takaisinreititys.
- Hakuviestien kulkema matka. Tämä kuvaa hakuun kulunutta aikaa (*Time to Satisfaction*). Matka koostuu hakuviestin kulkemien askelten sekä paluuviestin kulkemien askelten summasta.

Testiasettelussa pyrittiin minimoimaan kaikki algoritmin toimintaan vaikuttavat ulkopuoliset seikat, muun muassa siirtokanavan laadun vaihtelu, siirtohäiriöt ja muu rinnakkainen liikenne, jotka kaikki on otettava huomioon toteutettaessa reaalimaailman vertaisverkkosovellusta. Näin tulosten luotettavuutta saatiin parannettua.

5.2.1 Testausprosessi

Algoritmien suorituskykytestaus tapahtuu viidellä eri verkon koolla, jotka ovat 50, 100, 250, 375 ja 500 solmua. Tällä tavalla voimme havaita testitulosten käyttäytymismallin, eli kasvaako esimerkiksi viestiliikenteen määrä lineaarisesti, eksponentiaalisesti vai jollakin muulla tavalla.

Kutakin hakualgoritmia ajetaan jokaisessa verkon kokoluokassa kahdellakymmenellä satunnaisesti rakennetulla potenssijakautuneella verkkotopologialla. Myös resurssijakauma on satunnaisesti valittu kullekin verkolle. Suuri testiajojen määrä takaa sen, että tuloksista tehty keskiarvo antaa todellisuutta paremmin vastaavan kuvan.

Testimateriaali luodaan automaattisesti *TestCaseGenerator* -ohjelmalla. Sille määritellään halutut testiparametrit, joiden perusteella ohjelma luo testitapauskohtaiset syötetiedostot ja verkkotopologiat. Näitä syötteitä käytetään *P2PTool*-sovelluksen ohjaamiseen, joka suorittaa varsinaisen verkon rakentamisen, resurssien jakamisen verkkoon, lähettää haun verkkoon, odottaa vastauksia ja kirjoittaa testiajon tulokset omaan tiedostoonsa. Tätä tutkimusta varten testigeneraattori luo yhteensä 600 testiajoa.

Koska BFS-algoritmi rajoittaa haun TTL:n määrittämän säteen sisälle, sen kanssa jouduttiin käyttämään laajenevan kehän (*Expanding Ring*) strategiaa, joka tunnetaan myös nimellä *Iterative Deepening* [30]. Mikäli algoritmi ei tuota tarvittavaa määrää vastausviestejä, haun aloittanut solmu osaa lähettää verkkoon uuden haun edellistä hakua isommalla TTL-arvolla. Tällainen menettely aiheuttaa verkkoon hieman ylimääräistä liikennettä, mutta sen oletetaan kuuluvan algoritmin heikkouksiin. Lisäksi aloitus-TTL:n valinnalla on suuri merkitys hakua suoritettaessa.

Alla olevassa taulukossa on esitelty BFS-algoritmin aloitus-TTL-arvot kullekin verkon koolle.

Verkon koko	**Aloitus-TTL arvo**
50	2
100	3
250	3
375	3
500	3

Taulukko 5.1: Viestien TTL-arvot eri verkon kokoluokille

Testauksen vaiheet menevät seuraavasti:

Vaihe 1 Verkon solmujen käynnistäminen. *P2PClientTester*-sovelluksen avulla käynnistetään haluttu määrä P2PCore-solmuja. Tässä vaiheessa solmut ovat itsenäisiä eikä niillä ole yhteyksiä naapureihin eikä verkon topologiaa ole määritelty. Kuhunkin solmuun on alustettu vain testauksen kohteena oleva hakualgoritmi, ja viestien mitätöitymisaika on määritelty äärettömän pitkäksi.

Vaihe 2 Resurssien määrittely. Kun kaikki solmut on saatu alustettua, niiden resurssit määritellään käyttäen *P2PTool*-työkalua. Resurssijakauma on ennalta määrätty kullekin topologialle ja haettavan resurssin esiintymisprosentti pidetään samana.

Vaihe 3 Yhteyksien luominen. Tässä vaiheessa solmut voidaan liittää toisiinsa edellä määritellyn topologian mukaisesti. Kun yhteydet on muodostettu, verkko on valmis testattavaksi.

Vaihe 4 Haun lähettäminen. *P2PTool* lähettää resurssikyselyn aiemmin valitulle satunnaiselle solmulle ja alkaa odottamaan vastausviestejä. Vastausten kulkema matka verkossa kirjataan ylös. Haku lopetetaan, kun vastauksia on saatu tietty määrä tai viimeisimmän vastausviestin saapumisesta on kulunut 30 sekuntia. Haun edetessä sovellus näyttää tilannetietoa verkon löytämistä resursseista ajan kuluessa.

Vaihe 5 Tulosten kerääminen. *P2PTool* yhdistyy jokaiseen verkon solmuun ja lähettää niille status-kyselyn. Vastaukset saatuaan työkalu näyttää yhteenvedon haun tuloksista.

Kustakin testiajosta tehdään oma tulostiedostonsa, joka sisältää seuraavat tiedot:

- Haun aiheuttama viestien kokonaismäärä verkossa.
- Vastausviestin suurin kulkema matka verkossa (hops-arvo). Mikäli haettiin useita resursseja, täksi arvoksi valitaan suurin vastausviesteistä löytynyt hops-arvo.
- BFS-algoritmin tapauksessa TTL-arvo, jolla haku saatiin onnistuneesti suoritettua.

5.2.2 Testaamisen ongelmakohtia

Eräs käytännönläheinen ongelma on suuren verkon simulointi yhdellä tietokoneella. Todellisuudessa vertaisverkot voivat sisältää jopa kymmeniä tuhansia solmuja, ja tällaisen tilanteen mallintaminen vaatisi todella paljon resursseja. Lisäksi yhdellä koneella tapahtuva verkon simulointi ei ota huomioon verkon tiedonsiirrossa ta-

pahtuvaa latenssia. Tämä voitaisiin tottakai sisällyttää osaksi testiympäristöä, mutta tämän tutkielman kannalta asia on merkityksetön.

Myös algoritmien erilaisuus toi mukanaan ongelmia. BFS-algoritmi käyttää viestin elinkaaren määrittelemiseen TTL-arvoa, eli viesti tuhotaan tietyn askelmäärän jälkeen. Tässä tutkimuksessa käytetyissä HDEG- ja RWALK-algoritmeissa sen sijaan ei viestin elinkaarta ollut määritelty, vaan sen annettiin kulkea verkossa niin kauan, että haettu resurssi löytyi. Näin ollen testitulokset eivät suoraan olleet vertailukelpoisia keskenään, sillä BFS-viestien liian iso TTL-arvo vaikuttaa suoraan viestiliikenteen määrään. Liian pieni arvo puolestaan aiheuttaa sen, ettei resursseja välttämättä löydy tarpeeksi (esimerkiksi haettaessa tiettyä määrää kaikista määritellyistä resursseista). BFS-algoritmia testatessa haettiinkin kullekin verkolle sopiva, tarpeeksi pieni aloitus-TTL-arvo. Lisäksi haun uusiminen isommalla TTL-arvolla varmisti sen, että haku saatiin kuitenkin onnistuneesti suoritettua. Näin BFS-algoritmin viestiliikenne pyrittiin saamaan rajoitettua ja tulokset vertailukelpoisemmiksi keskenään. Tämä TTL-arvon optimointi onkin yksi BFS-tyyppisten algoritmien yleisimmistä ongelmista [17, s. 6-7]

Testiajojen tulosten analysointi osoitti myös erään seikan, joka vaikutti vääristävästi vertailuun. Alunperin yhtenä testauksen metriikkana oli tarkoitus käyttää aikaa, jonka haun suorittaminen kestää. Algoritmien implementoinnin eroavuus aiheutti kuitenkin sen, että viestien kulkunopeus verkossa vaihteli eri algoritmien välillä. Esimerkiksi HDEG-algoritmilla solmun yksittäisen viestin käsittelyyn kulunut aika oli huomattavasti suurempi kuin RWALK- ja BFS-algoritmeilla. Tästä syystä testimetriikkaa täytyi muuttaa siten, ettei kyseinen asia vaikuttanut tuloksiin. Ajan sijaan metriikkana käytettiinkin viestin kulkemaa matkaa, jolla oli edelleen yhteys aikamuuttujan kanssa (esimerkiksi 1 askel = n millisekuntia), mutta se ei sisältänyt toteutuksen erojen tuomaa virhettä.

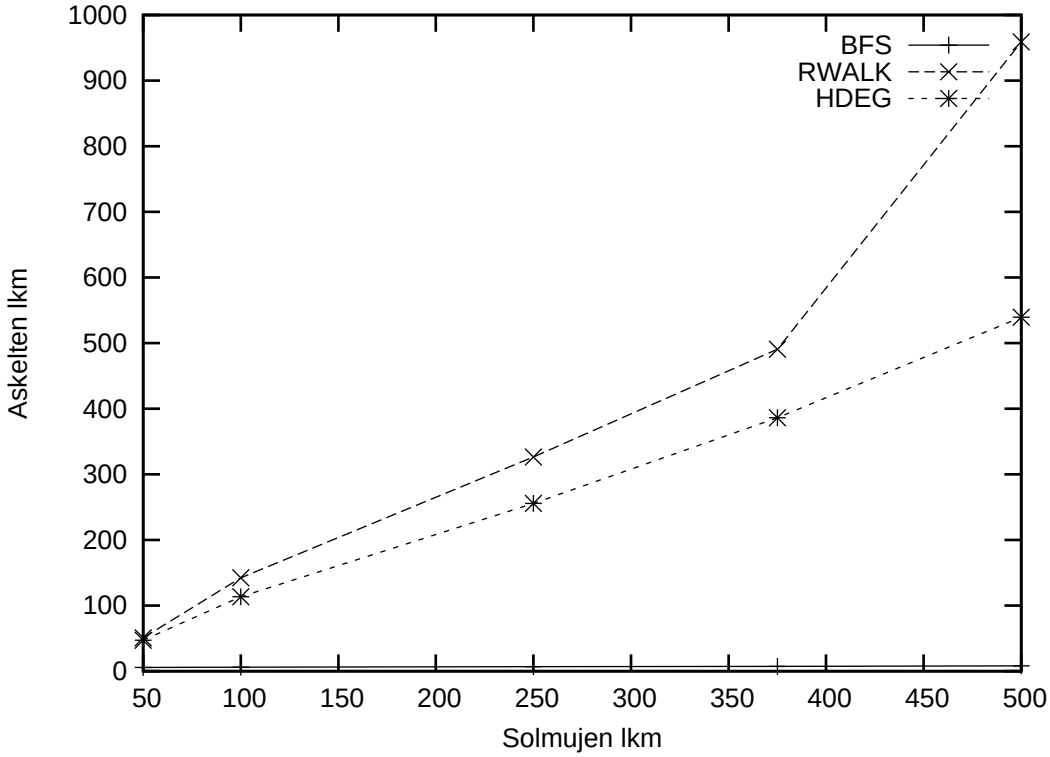
6 Hakualgoritmien suorituskykyvertailu

Testitulokset osoittavat, että algoritmin valinnalla todellakin on merkitystä. Alla olevat graafit kuvaavat kunkin algoritmin toimintaa eri verkkojen koolla. Tulosten kesken on selvästi havaittavissa käänteinen yhteys resurssien löytymiseen kuluneen ajan ja algoritmin aiheuttaman verkkoliikenteen määrän välillä.

6.1 Nopeus

6.1.1 Yhden resurssin etsintä

Kuten edellä mainittiin, vastausviestin kulkema matka voidaan rinnastaa suoraan haun kestoon.



Kuva 6.5: Viestien kulkema matka haettaessa yhtä resurssia.

Kuvassa 6.5 on esitetty vertailu eri algoritmien viestien kulkemasta matkasta etsittäessä yksittäistä verkon resurssia. Kuten kuvasta nähdään, BFS-algoritmi on selvästi ylivoimainen nopeutta vaativissa hauissa.

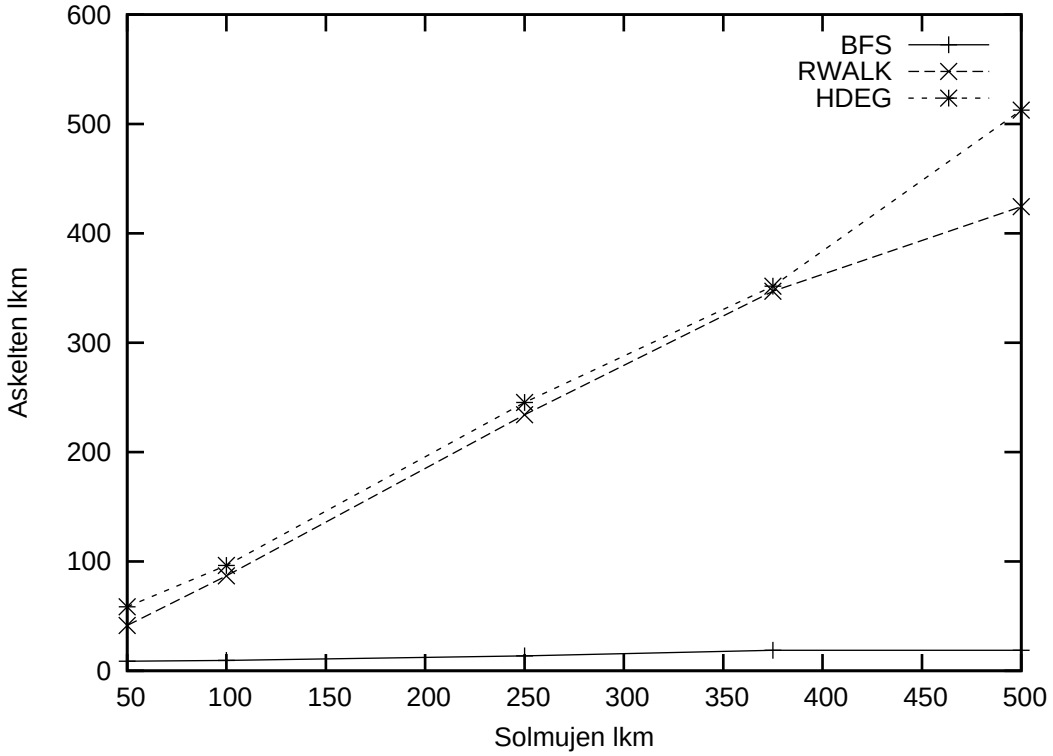
HDEG- ja RWALK-algoritmien tuloksista huomaa, että ne selvästikin keskittyvät kulkemaan verkossa pitkiä matkoja kaistaa säästellen. Verkon koon kasvaessa RWALK-algoritmin suorituskyky alkaa heikkenemään rajusti. Tämä on täysin odotettavissa, sillä yhä isommasta joukosta solmuja yksittäisen tai muutaman resurssin löytäminen vaikeutuu eksponentiaalisesti.

HDEG suoriutuu RWALK-algoritmia paremmin siitä syystä, että se pyrkii välttämään viestin lähettämistä solmuille, joilla viesti on jo käynyt. Tällöin resurssi löy-

tyy vähemmällä askelmäärällä. Lisäksi se hakeutuu ensimmäisenä verkon runkosolmuihin ja näin ollen se löytää pienemmällä vaivalla suuremman määrän solmuja, joita tutkia. Liitteessä 1 taulukossa 10.2 löytyy algoritmikohtaiset tulosdiagrammit.

6.1.2 Etsittäessä 50% resursseista

Haettaessa useampaa resurssia HDEG- ja RWALK-algoritmien erot pienenevät. Kuvassa 6.6 esitetään hakuviestien kulkema matka etsittäessä puolta verkossa olevista resursseista.



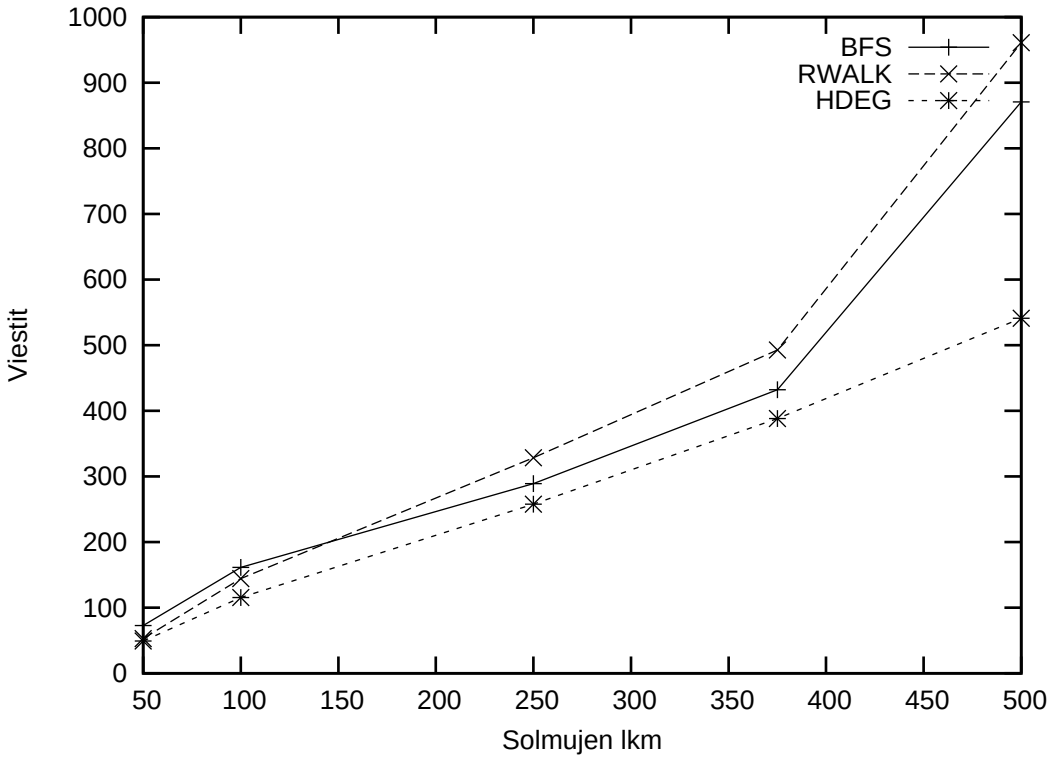
Kuva 6.6: Viestien kulkema matka haettaessa 50% resursseista.

Edelleen BFS vie reilusti voiton muista, mutta RWALK- ja HDEG-algoritmien tulokset pysyttelevät lähes samoina 375 solmun kokoiseen verkkoon saakka. Mielenkiintoista on, että tällä kertaa RWALK toimii HDEG-algoritmia tehokkaammin. Tämän ilmiön selittää se, että todennäköisesti HDEG-algoritmissa viesti jää kiertelemään verkon keskussolmujen läheisyyteen eikä tällöin heti löydä verkon ulkolaidoilla olevia resursseja. RWALK sen sijaan löytää nämä reunasolmujen resurssit sattumalta, aiheuttaen paremman suorituskyvyn.

6.2 Verkkoliikenteen määrä

Toinen hakualgoritmien suorituskyvyn mittareista on niiden aiheuttaman verkkoliikenteen määrä. Tässä kappaleessa tarkastellaan, miten tutkittavat algoritmit tuottavat viestiliikennettä eri hakutavoilla.

6.2.1 Yhden resurssin etsintä



Kuva 6.7: Haun tuottama viestiliikenne haettaessa yhtä resurssia.

Viestiliikenteen määrän suhteen BFS-algoritmi antoi yllätyksen. Kuten kuvasta 6.7 nähdään, BFS tuotti yhtä resurssia etsittäessä paremman tuloksen kuin RWALK-algoritmi. Syinä tähän on epärealistisen pienet testausverkkojen koot sekä verkkojen potenssijakauma. BFS-algoritmissa verkon solmu osaa tuhota jo kertaalleen käsittelemänsä hakuviestin, joten vastoin yleisiä ennakkoluuloja viestien määrä ei kasvakaan räjähdysmäisesti. Pienessä verkossa solmut ovat melko lähekkäin, toisin sanoen vain muutaman askelen päässä toisistaan, joten algoritmi hillitsee tehokkaasti viestiliikenteen kasvua.

Yksi BFS-algoritmin tuloksiin positiivisesti vaikuttanut seikka piilee testausympäristön toteutuksessa. Kun hakuviestin lähettäjäsolmu saa tarvittavan määrän vastausviestejä, se ilmoittaa testaussovellukselle, että haku on valmis. Tällöin testaussovellus lähettää verkon solmuille *status*-viestin ja kerää niiden senhetkiset viestien välitystiedot talteen välittämättä siitä tosiseikasta, että hakuviestien reititys on edelleen kesken. Sovelluksen tulisikin ensin odottaa viestien tuhoutumista TTL-arvon mentyä nollaan. Todellisuudessa viestiliikenteen määrä on siis jonkin verran suurempi kuin mitä tulokset antavat ymmärtää.

RWALK-algoritmin tuloksiin vaikuttaa sen täydellinen satunnaisuus; hakuviesti vaeltelee päämäärättömänä solmulta toiselle ja voi pahimmassa tapauksessa kiertää kehää tai vierailla samoissa solmuissa useita kertoja ennen kuin haettu resurssi löytyy.

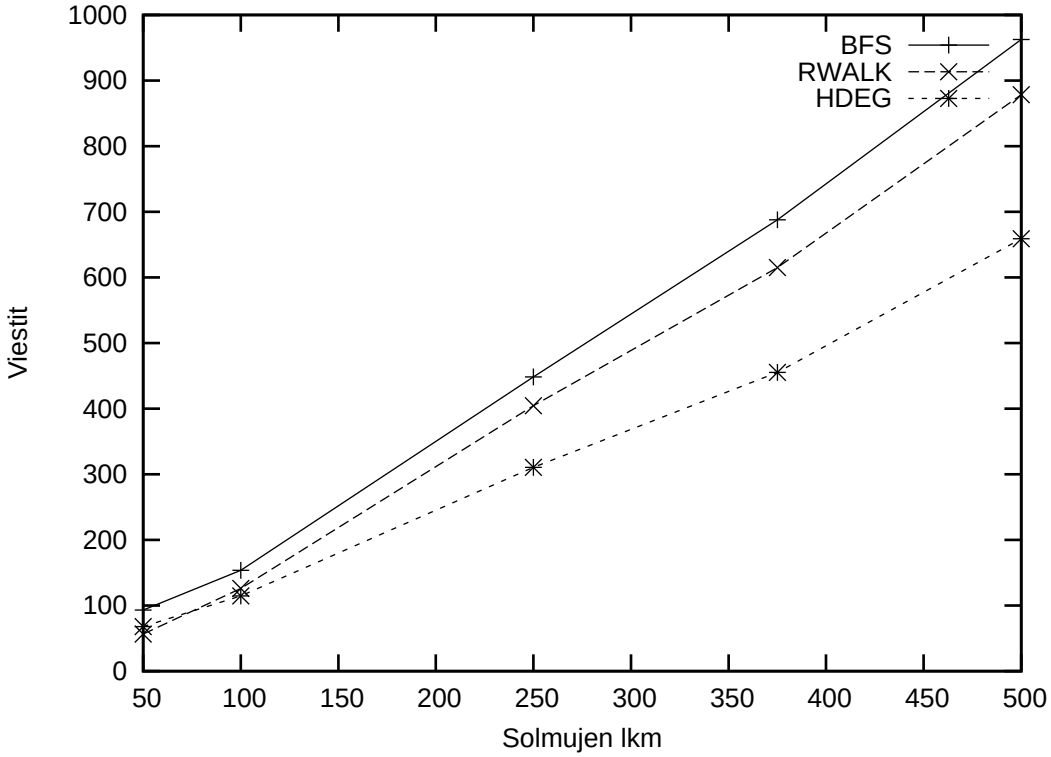
Odotetusti HDEG-algoritmi suoriutuu tässä vertailussa parhaiten. Solmujen kategorisointi mustiin ja harmaisiin solmuihin aiheuttaa viestien ohjaamisen ensisijaisesti solmuille, jotka eivät vielä viestiä ole saaneet. Tällöin turhan liikenteen määrä vähenee huomattavasti. Lisäksi huomioitavaa on, että HDEG-algoritmin tulosgraafi näyttää lähes lineaariselta verrattuna BFS:n ja RWALK:in lopussa jyrkästi nouseviin tuloksiin. Oletettavaa on, että sama trendi jatkuisi myös isommillakin verkoilla.

Tässä tutkimuksessa käytetyillä pienillä verkoilla algoritmien suorituskykyerot ovat vielä melko vähäisiä, joten olisikin ollut mielenkiintoista testata viestiliikenteen määrän kehitystä vieläkin suuremmilla, esimerkiksi 2000 ja 5000 solmua kattavilla verkoilla.

6.2.2 Etsittäessä 50% resursseista

Etsittäessä puolia kaikista verkkoon asetetuista resursseista HDEG- ja RWALK-algoritmien hyödyt tulevat paremmin esiin. Kuvassa 6.8 on esitelty viestiliikenteen määrä haettaessa useampaa resurssia. Etenkin HDEG-algoritmin suorituskyky erottuu selvästi edukseen muihin verrattuna.

Tällä kertaa BFS-algoritmin heikkoudet tulevat esiin. Syynä tähän on osittain edellisessä kappaleessa liian hyviä tuloksia antanut virhe testaussovelluksessa. Tässä testiasetelmassa reititystuloksia ei haetakaan kesken viestien reititysoperaatiota, vaan viestit ehtivät reitittyä eteenpäin ja lisätä verkkoliikennettä ennen tulosten keräämistä.



Kuva 6.8: Haun tuottama viestiliikenne haettaessa 50% resursseista.

7 Johtopäätökset

Mistä RWALK- ja HDEG-algoritmien hitaus johtuu? Kun verrataan BFS:n viestimääriä ja viestien kulkemia matkoja sekä verrataan niitä HDEG- ja RWALK-algoritmien vastaaviin, voidaan tehdä seuraavat johtopäätökset:

- Viestin monistaminen on "halpa" operaatio. Viestiä monistamalla algoritmin hakunopeus lisääntyy verkkoliikenteen kasvun kustannuksella. Sama viesti voidaan välittää useammalle naapurille samassa ajassa kuin se lähetettäisiin vain yhdelle.
- Viestin edelleenlähettäminen naapurisolmulle on "kallis" operaatio. Liiallinen viestin edelleenohjaaminen hidastaa algoritmia.

Yllä mainittujen huomioiden perusteella on loogista, että yksittäistä hakuviestiä käyttävät algoritmit ovat hitaampia: mikäli käytössä olisi BFS-verkko, jossa jokaisella solmulla n on naapureiden määrä $D(n)$, niin tällöin kyseiseen verkkoon lähetetty

hakuviesti saavuttaa $(D(n))^t$ solmua ajassa t, jossa t = askelten lukumäärä. RWALK- ja HDEG-algoritmit puolestaan saavuttavat vain t naapuria.

Testaustulokset välillä 50 - 375 solmua eivät olleet kovin mielenkiintoisia. Oleellisia muutoksia alkoi tapahtumaan vasta, kun verkon koko oli välillä 375 - 500 solmua. Etenkin yhtä resurssia etsivät haut alkoivat käyttäytymään tuhlailevammin.

On hyvin todennäköistä, että yhdistelemällä testattujen algoritmien parhaita puolia saataisiin erittäin suorituskykyinen ja verkkoliikennettä tehokkaasti käyttävä algoritmi. Jos esimerkiksi HDEG-algoritmiin lisättäisiin mekanismi, joka monistaisi hakuviestin useammaksi verkon runkosolmun löydettyään, hakunopeutta saataisiin luultavasti parannettua. Myös RWALK-algoritmi tehostuisi, jos verkon solmut käyttäisivät reititystaulukkoaan hyväksi vastaanottajasolmua valitessaan ja ensisijaisesti yrittäisivät valita "valkean" solmun.

Valittaessa parasta hakualgoritmia pienelle vertaisverkolle BFS-tyyppinen leveyshaku on tulosten perusteella tehokkain ratkaisu. Sen hakuajat ovat huomattavasti paremmat kuin yksittäisen hakuviestin lähettävillä algoritmeilla ja verkkoliikenteen määräkin pysyy kohtuullisena.

8 Aiheeseen liittyviä tutkimuksia sekä jatkotutkimusideoita

Olisi hyvä tutkia, miten algoritmit reagoivat erilaisiin resurssien esiintymisprosentteihin. Onko joku algoritmi parempi tietynlaisen resurssijakauman kanssa kuin muut? Voiko hakua tehostaa valitsemalla oikea algoritmi, kun tiedämme jotain resurssien esiintymisestä? Lähes kaikki vertaisverkkoihin liittyvät julkaisut ja algoritmivertailut sivuuttavat tämän näkökulman.

Useiden algoritmien hyviä puolia yhdistelemällä luultavasti päästäisiin vieläkin parempiin tuloksiin. Yksi tällaisista yhdistelmistä on Firework- algoritmi [19], joka sisältää piirteitä Random Walk -algoritmista ja BFS:stä lisättynä solmujen ryvästysmekanismilla. Kyseinen algoritmi on tulosten perusteella BFS-algoritmia suorituskykyisempi sekä haun suhteellisen tehokkuuden että viestiliikenteen määrän suhteen. Tämä *suhteellinen tehokkuus* tarkoittaa hakutulosten määrää suhteessa haun aiheuttaman viestiliikenteen määrään.

C. Avin ja B. Krishnamachari [5] puolestaan ovat kehittäneet Random Walk -algoritmista version, joka HDEG-algoritmin tapaan käyttää naapurisolmujen astelukua

hyväkseen. Heidän valikoiva satunnaiskävelijä -algoritminsa (Random Walker with Choice, RWC) valitsee viestin vastaanottajan seuraavasti:

1. Luodaan joukko N valitsemalla satunnaisesti d kappaletta naapurisolmuja.
2. Lasketaan tämän joukon kullekin solmulle n_i viestiä m vastaava soveltuvuusarvo $G(m, n_i)$.
3. Näistä solmuista vastaanottajaksi valitaan se solmu n_i, jolla saadaan pienin $G(m, n_i)$.

Yllä mainittu soveltuvuusarvo lasketaan seuraavanlaisella kaavalla:

$$G(m, n_i) = \frac{Visits(m, n_i)}{Degree(n_i)}$$

jossa $Visits(m, n_i)$ on viestin m solmussa n_i vierailujen määrä ja $Degree(n_i)$ on solmun n_i asteluku. Kyseinen algoritmi on antanut testeissä erittäin lupaavia tuloksia.

V. Kalogeraki, D. Gunopulos ja D. Zeinalipour-Yazti [15, s. 3] esittelevät tutkielmassaan hieman optimoidun BFS-algoritmin, joka pyrkii vähentämään viestiliikennettä rajoittamalla monistettavien viestien määrää valitsemalla vain osan naapurisolmuista. Tämän algoritmin suorituskyä olisi mielenkiintoista verrata Random Walker -algoritmiin ja muihin viestiliikennettä tehokkaasti rajoittaviin menetelmiin.

Yksi mielenkiintoinen tutkimuskohde olisi myös K-Random Walker-algoritmi [11, s. 6] [17, s. 9], joka lähettää useamman hakuviestin samanaikaisesti. Kyseistä menetelmää käytettäessä voidaan olettaa, että Random Walker -algoritmille ominainen pitkä hakuaika lyhenee suhteessa lähetettävien hakuviestien määrään.

D. Tsoumakos, ja N. Roussopoulos ovat vertailleet yhdeksää eri vertaisverkkojen hakualgoritmia keskenään julkaisussaan "Analysis and Comparison of P2P Search Methods" [27]. Heidän tutkimuksessaan käytetään useaa eri algoritmityyppiä sekä muovautuvia verkkorakenteita. Tutkimuksen tuloksista käy ilmi, että käytettävä algoritmi tulee valita käyttötarkoituksen mukaan. Tutkimuksessa myös todetaan, että useimmissa tapauksissa yksinkertaisimmat algoritmit ovat kaikkein soveliaimpia; niiden toimintamekanismi on tehokas ja ne ovat helppoja toteuttaa.

Adaptiivisten menetelmien tärkeys korostuu verkon koon kasvaessa. Solmumäärän lisääntyessä myös viestiliikenteen määrä kasvaa. Kun tiedetään algoritmin viestin reititysmekanismi ja verkon koko, voidaan laskea suuntaa antava arvio tietoliikennekaistan tarpeesta. Esimerkiksi B. Yang ja H. Garcia-Molina esittävät arvion

Gnutella-verkon siirtokaistan tarpeesta julkaisussaan “Improving search in Peer-to-Peer Networks” [30, s. 5]. Tätä menetelmää voitaisiin käyttää algoritmien optimointiin. Esimerkiksi, jos algoritmin tärkeimpänä suorituskykyä mittaavana kriteerinä on sen nopeus, mutta algoritmi käyttää vain 10% käytettävissä olevasta kaistanleveydestä hyödykseen, se ei ole optimaalinen. Kun algoritmiin lisättäisiin logiikkaa, jonka avulla tietoliikennekaistan hyötysuhde saataisiin lähelle 100 prosenttia, algoritmi todennäköisesti suorittaisi saman haun nopeammin, mutta pysyisi silti vaadittujen rajoitteiden sisällä. Samalla tällainen adaptiivinen lisäominaisuus pitäisi myös huolen, että tiedonsiirtokaista ei pääsisi koskaan ylikuormittumaan.

9 Yhteenveto

Mobiilit päätelaitteet ja uudet verkkoteknologiat ovat tuoneet mukanaan aivan uudenlaisia haasteita verkko-ohjelmoinnille. Nopeasti topologiaansa muuttavat verkot ovat hiljalleen syrjäyttämässä perinteisiä Asiakas-Palvelin -mallin ratkaisuja. Vertaisverkot ovat osa tätä muutosta.

Vertaisverkot tarjoavat dynaamisen ja hierarkia-vapaan verkkomallin. Ne soveltuvat erinomaisesti vapaasti muuttuvan topologian käsittelyyn ja tiedon hakemiseen. Vertaisverkkojen tarjoamat edut eivät kuitenkaan ole ilmaisia, vaan tuovat mukanaan joukon uusia ongelmia.

Vertaisverkon tärkein osa on hakualgoritmi, joka ohjaa verkkoliikennettä ja huolehtii hakuihin vastaamisesta sekä tiedon kulkemisesta. Yksinkertaiselta kuulostava tehtävä pitää kuitenkin sisällään useita haasteellisia tehtäviä. Algoritmien tulee kyetä suoriutumaan tehtävästään siedettävässä ajassa ja silti pyrkiä minimoimaan muut haittavaikutukset, kuten yletön verkkoliikenteen aiheuttaminen. Verkon koon kasvaessa yksinkertainen leveyshaku ei enää olekaan optimaalisin tapa hakea tietoa. Tarvitaan siis kehittyneempiä menetelmiä.

Tässä tutkimuksessa selvitettiin eri vertaisverkkojen hakualgoritmien toimintatapoja ja niiden suorituskykyä. Vertailussa käytettiin leveyshakua (BFS), satunnainen kävelijä (RWALK) -algoritmia ja korkeimman asteen (HDEG) hakualgoritmia. Tutkimuksessa mitattiin algoritmien hakunopeutta sekä niiden aiheuttamaa verkkoliikenteen määrää.

Tutkimus toteutettiin simuloimalla eri kokoisia potenssijakautuneita vertaisverkkoja yksittäisellä työasemalla ja ajamalla niissä eri algoritmeja. Verkon suorituskykyä testattiin lähettämällä siihen hakuja ja keräämällä tarvittavat mittaustulokset haun päätyttyä. Tätä varten kehitettiin erillinen P2PCore-vertaisverkkoalusta, johon algoritmikomponentteja pystyi lisäämään modulaarisesti. Sovellusta kehitettäessä suurimmat haasteet olivat massiivisen rinnakkaisuuden käsittely lukitusten avulla sekä toimivan testausympäristön rakentaminen varsinaisen vertaisverkkoalustan rinnalle. Lisätietoja P2PCore-alustasta löytyy liitteestä 10.3.

Testaustulokset olivat odotettavia, joskin BFS-algoritmi tuotti yllätyksen tehokkuudellaan. Vastoin yleistä käsitystä sen tuottama verkkoliikenteen määrä pysyi hillittynä. Joissakin tilanteissa se jopa toimi säästeliäämmin kuin RWALK-algoritmi. Toisaalta tähän on osittain syynä testiympäristön asettama testausverkon kokorajoitus 500 solmuun.

Testeissä näkyi selvästi myös BFS-algoritmin ero hakunopeudessa. Yhden viestin lähettävät RWALK- ja HDEG-algoritmit suoriutuivat hakuoperaatiosta jopa useita kymmeniä kertoja hitaammin kuin BFS.

Highest Degree Search -algoritmi toimi oletetulla tavalla; verrattaessa BFS-algoritmiin sen tiedettiin olevan nopeudessa hitaampi, mutta tehokkaampi vertailtaessa verkkoliikenteen määrää. Tulokset osoittivat odotukset todeksi. HDEG-algoritmin solmujen kirjanpito auttoi löytämään haetut resurssit nopeammin ja vähemmän viestiliikennettä tuottaen kuin satunnaisuuteen perustuva RWALK-algoritmi.

Vertaisverkkojen hakualgoritmeista on tehty paljon tutkimusta ja odotettavissa on entistä tehokkaampia ja toimivampia toteutuksia. Yksi varteenotettavimmista tutkimuskohteista onkin tarkastella eri algoritmien toteutuksia ja poimia niistä parhaimmat puolet. Useasta algoritmista koostuva hybridi voisi olla erittäin toimiva ratkaisu monenlaisissa vertaisverkkosovelluksissa.

Viitteet

[1] Adamic, L. A., Lukose, R. M., Puniyani, A. R., Huberman, B. A., ”Search in power-law networks”, Physical Review E, Vol. 64, Issue 4, 2001.

[2] Androutsellis-Theotokis, S., Spinellis, D., ”A Survey of Peer-to-Peer Content Distribution Technologies”, ACM Computing Surveys, Vol. 36, No. 4, 2004, s. 335 - 371

[3] Auvinen, A., ”Topologian hallinta-algoritmit Chedar-vertaisverkkoalustassa”, Pro gradu -tutkielma, Jyväskylän yliopisto, 2004.

[4] Auvinen, A., Vapa, M., Weber, M., Kotilainen, N., Vuori, J., ”New Topology Management Algorithms for Unstructured P2P Networks”, Second International Conference on Internet and Web Applications and Services (ICIW ’07), 2007.

[5] Avin, C., Krishnamachari, B., ”The Power of Choice in Random Walks: An Empirical Study”, Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems, 2006, s. 219 - 228

[6] Barabási, A., Bonabeau, E., ”Scale-Free Networks”, Scientific American, toukokuu 2003, s. 50 - 59.

[7] BitTorrent, http://sourceforge.net/projects/bittorrent/.

[8] Cheese Factory Project, http://www.mit.jyu.fi/cheesefactory/.

[9] Clarke, I., Miller, S.G., Hong, T.W., Sandberg, O., Wiley, B., ”Protecting free expression online with Freenet”, IEEE Computing, Vol. 6, Issue 1, 2002, s. 40 - 49.

[10] Cuenca-Acuna, F. M., Peery, C., Martin, R. P., Nguyen, T. D., ”PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities”, Proceeding of the International Symposium on High Performance Distributed Computing, 2003, s. 236 - 246

[11] Fletcher, G. H. L., Sheth, H. A., Börner, K., ”Unstructured Peer-to-Peer Networks: Topological Properties and Search Performance”, International Workshop on Agents and Peer-to-Peer Computing, 2004.

[12] Gnutella, http://gnutella.wego.com/.

[13] Gong, L., "JXTA: a network programming environment", IEEE Computing, Vol. 5, Issue 3, 2001, s. 88 - 95.

[14] Joost, http://www.joost.com/.

[15] Kalogeraki, V., Gunopulos, D., Zeinalipour-Yazti, D., "A Local Search Mechanism for Peer-to-Peer Networks", Proceedings of the eleventh international conference on Information and knowledge management, 2002.

[16] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B., "OceanStore: an architecture for global-scale persistent storage", Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ACM ASPLOS), 2000.

[17] Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S., "Search and Replication in Unstructured Peer-to-Peer networks", Proceedings of the 16th international conference on Supercomputing, 2002.

[18] Napster, http://free.napster.com/.

[19] Ng, C., Sia, K., King, I., "A Novel Strategy for Information Retrieval in the Peer-to-Peer Network", The Chinese University of Hong Kong, 2002.

[20] PeerCast, http://www.peercast.org/.

[21] Ritter, J., "Why Gnutella Can't Scale. No, Really.", http://www.darkridge.com/ jpr5/doc/gnutella.html.

[22] Schollmeier, R., "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", Proceedings of the First International Conference on Peer-to-Peer Computing, 2001, s. 101-102.

[23] Schollmeier, R., Schollmeier, G., "Why peer-to-peer (P2P) does scale: an analysis of P2P traffic patterns", Proceedings of the Second International Conference on Peer-to-Peer Computing, 2002, s. 112-119.

[24] Sit, E., Morris, R., "Security Considerations for Peer-to-Peer Distributed Hash Tables", Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems, 2002.

[25] Skype, http://www.skype.com/.

[26] Tsoumakos, D., Roussopoulos, N., ”Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks”, Proceedings of the Third International Conference on Peer-to-Peer Computing, 2003, s. 102 - 109.

[27] Tsoumakos, D., Roussopoulos, N., ”Analysis and Comparison of P2P Search Methods”, Proceedings of the First International Conference on Scalable Information Systems, 2006.

[28] Vapa, M., Kotilainen, N., Auvinen, A., Kainulainen, H., Vuori, J., ”Resource Discovery in P2P Networks Using Evolutionary Neural Networks”, International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA 2004), 2004.

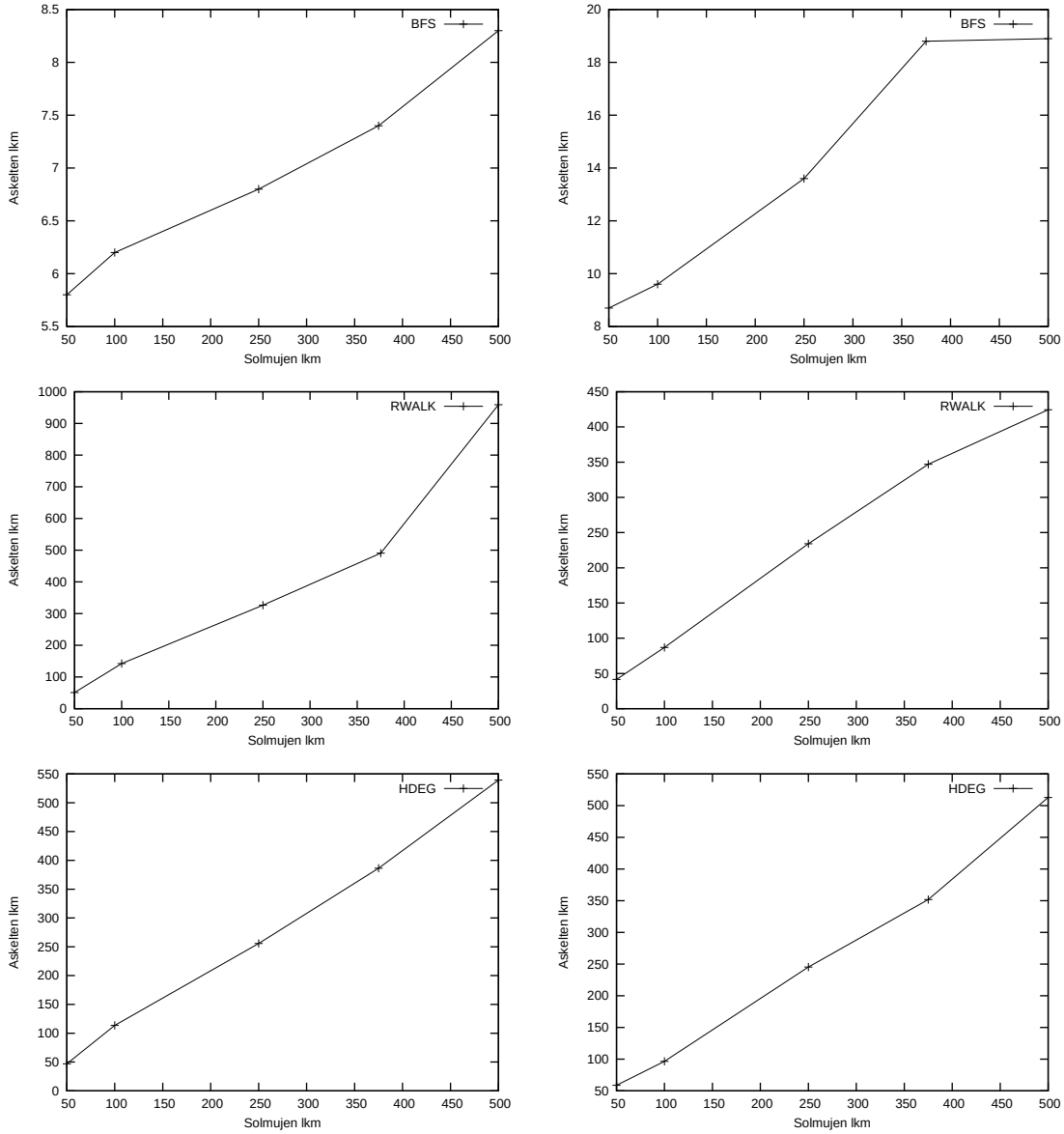
[29] Volovikov, O., Kotilainen, N., Juonoja, T., Vapa, M., Weber, M., Vuori, J., ”Mobile Encounter Networks and Their Applications”, Consumer Communications and Networking Conference (CCNC 2008), 2008, s. 1176 - 1180.

[30] Yang, B., Garcia-Molina, H., ”Improving search in Peer-to-Peer Networks”, Proceedings of 22nd International Conference on Distributed Computing Systems, 2002.

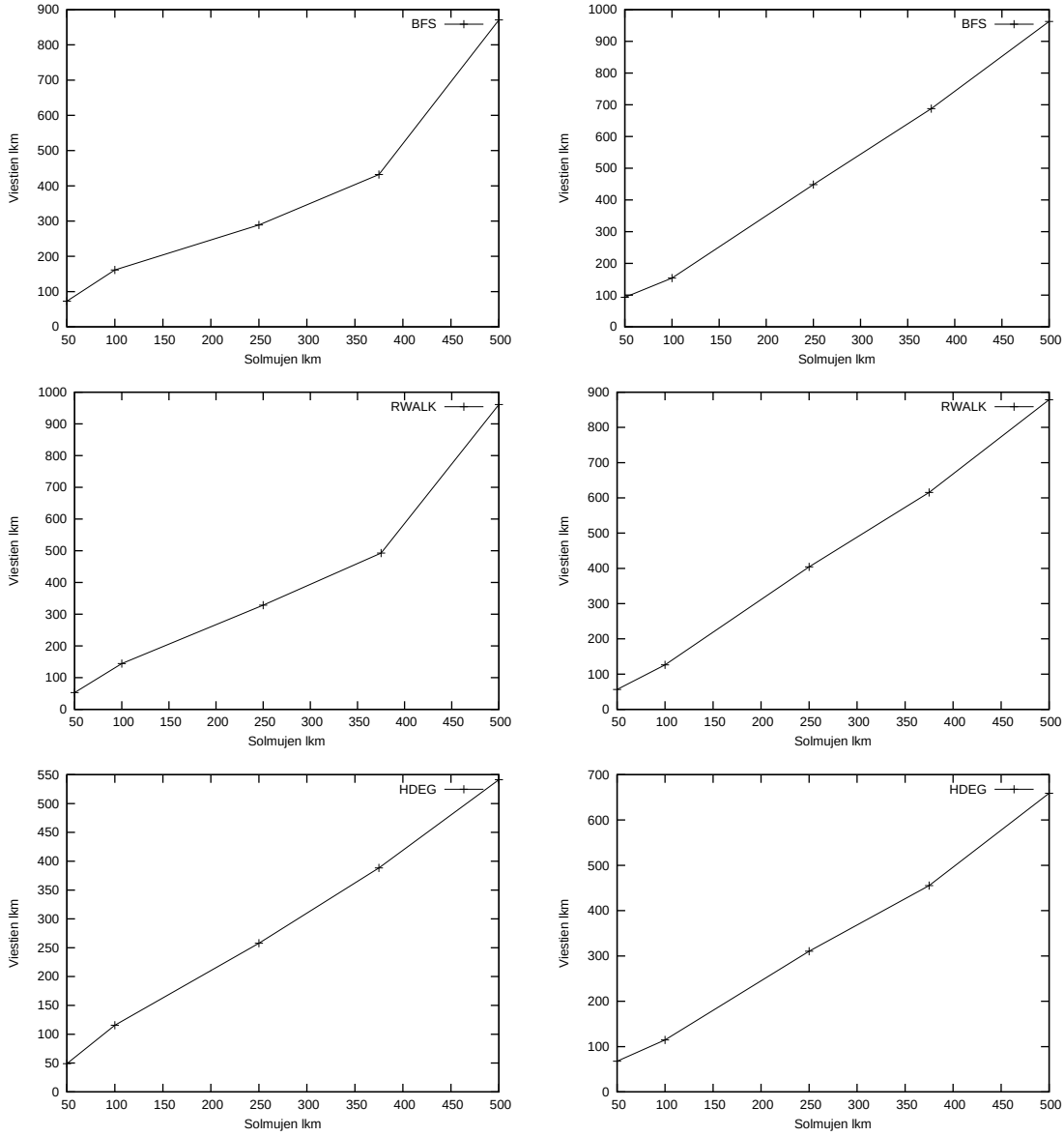
10 Liitteet

10.1 Liite 1: Algoritmikohtaiset tulosdiagrammit

Alla olevissa taulukoissa on esitelty tulosgraafit algoritmeittäin. Vasemmanpuolimmaiset kuvat esittävät tuloksia etsittäessä yksittäistä resurssia ja oikeanpuoleisissa on tulokset etsittäessä 50 prosenttia resursseista.



Taulukko 10.2: Algoritmikohtaiset tulokset mitattaessa viestien kulkemaa matkaa



Taulukko 10.3: Algoritmikohtaiset tulokset mitattaessa viestien määrää

10.2 Liite 2: Numeeriset testitulokset

Seuraavissa taulukoissa on esitelty testaustulokset numeerisessa muodossa. Luvut ovat usean testiajon keskiarvoja.

Testiajon tyyppi	**50 solmua**	**100**	**250**	**375**	**500**
BFS: 1 resurssi	5.80	6.20	6.80	7.40	8.30
BFS: 50% resursseista	8.70	9.60	13.60	18.80	18.90
RWALK: 1 resurssi	50.95	142.15	326.55	490.75	958.80
RWALK: 50% resursseista	41.50	86.95	234.20	347.00	424.45
HDEG: 1 resurssi	47.00	113.50	255.75	386.35	539.25
HDEG: 50% resursseista	58.65	96.45	245.25	351.80	512.80

Taulukko 10.4: Viestien kulkema matka eri algoritmeilla.

Testiajon tyyppi	**50 solmua**	**100**	**250**	**375**	**500**
BFS: 1 resurssi	72.95	161.50	289.05	432.05	870.90
BFS: 50% resursseista	72.95	161.50	289.05	432.05	870.90
RWALK: 1 resurssi	52.95	144.15	328.55	492.75	960.80
RWALK: 50% resursseista	56.35	126.05	404.35	615.20	878.30
HDEG: 1 resurssi	49.00	115.50	257.75	388.35	541.25
HDEG: 50% resursseista	67.80	114.75	310.55	455.20	658.95

Taulukko 10.5: Viestien määrä eri algoritmeilla.

10.3 Liite 3: P2PCore-vertaisverkkoalustan tietoja

Käyttöjärjestelmä: Debian Linux

Ohjelmointikieli: Java 1.6.0

Koodirivien määrä: 10861

Luokkien määrä: 48

Rajapintaluokkien määrä: 8

Konfiguraationhallinta: Ant

Graafiset työkalut: Gnuplot, Dia

Apukirjastot: TinyXML

Muut työkalut: 23 kappaletta bash-shell skriptejä

10.4 Liite 4: Esimerkki *P2PTool*-työkalun ohjaustiedostosta

```
loadnodes net/nodes_100.txt
waitconnections
topology net/tc_182_topology.ini
waittopology
netsetup hdeg
set_ttl 3
setresource FindMe localhost:2079
setresource FindMe localhost:2071
setresource FindMe localhost:2029
setresource FindMe localhost:2035
setresource FindMe localhost:2001
setresource FindMe localhost:2081
setresource FindMe localhost:2085
setresource FindMe localhost:2012
setresource FindMe localhost:2067
setresource FindMe localhost:2095
setneededresources 5
waitresources
query kissa localhost:2050
waitreplies 0
status
waitstatus
writestats res/tc_182_hdeg_pl100_10res.txt
disconnect
quit
```