Annemari Auvinen

TOPOLOGIAN HALLINTA -ALGORITMIT CHEDAR-VERTAISVERKKOALUSTASSA

Tietotekniikan pro gradu -tutkielma

Ohjelmistotekniikan linja

1.3.2004

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Annemari Auvinen

Yhteystiedot: Sähköposti annauvi@cc.jyu.fi

Työn nimi: Topologian hallinta -algoritmit Chedar-vertaisverkkoalustassa

Title in English: Topology management algorithms in Chedar peer-to-peer platform

Työ: Pro gradu -tutkielma

Sivumäärä: 81

Linja: Ohjelmistotekniikka

Teettäjä: Jyväskylän yliopiston Agora Center ja tietotekniikan laitos

Avainsanat: topologian hallinta, Chedar, vertainen, vertaisverkko, P2P, alusta, potenssijakauma, ohitus, solmun hyvyys, kuormituksen arviointi

Keywords: topology management, Chedar, peer, peer-to-peer, P2P, platform, power-law, overtaking, peer's goodness, load estimation

Tiivistelmä: Vertaisverkot sopivat hyvin tiedon levittämiseen hajautetusti, mutta niiden ongelmana on, miten solmun yhteyksiä lisätään ja poistetaan dynaamisesti muotoutuvassa verkossa ja miten resurssien haku saadaan mahdollisimman tehokkaaksi. Tämän tutkielman teoriaosuudessa käsitellään vertaisverkkoja ja graafeja sekä aiheeseen liittyviä aikaisempia tutkimuksia. Käytännön osassa tutkitaan Chedar-vertaisverkkoalustaan kehitettyjen topologian hallinta -algoritmien toimintaa keskittyen niiden muodostamiin topologioihin.

Abstract: Peer-to-Peer networks are suitable for diffusing the content in a decentralized manner. The problem is to define how peer should add and drop connections in dynamic environment and how to make searching of resources as effective as possible. The theory part of this thesis consists of basics of Peer-to-Peer networks, graph theory and previous works related to the subject. Chedar Peer-to-Peer platform and the topology algorithms it uses are studied in the practical part especially focusing on the topologies formed by the algorithms.

Termiluettelo

ARPANET	1969 luotu Internetin edeltäjä, jota käytettiin uusien verkkotekniikoiden testausalustana. Verkon muodostivat eri yliopistot ja tutkimuskeskukset.
BFS	Breadth First Search. Leveyshakutyyppinen algoritmi, jossa viesti välitetään kaikille solmun naapureille lukuun ottamatta solmua, jolta viesti saatiin. Algoritmissa syvyyttä rajoitetaan viestin TTL-arvolla.
FTP	File Transfer Protocol. Internetissä käytetty tiedostojen siirtoon tarkoitettu protokolla.
P2P	Peer-to-Peer. Vertaisverkko. Verkko, jossa kommunikointi tapahtuu koneiden välillä ilman palvelimia.
Hybridi vertaisverkko	Vertaisverkko, jossa hakemiseen käytetään välityspalvelinta.
Puhdas vertaisverkko	Vertaisverkko, jossa ei ole keskitettyä pistettä.
Potenssijakautunut verkko	Verkko, jossa suurella osalla solmuja on vähän naapureita ja muutamalla solmulla paljon naapureita sekä verkon halkaisija on lyhyt.
TCP	Transmission Control Protocol. Tarjoaa yhteydellisen ja luotettavan tiedonsiirron laitteiden välillä.
TELNET	Pääteohjelma, jolla voidaan ottaa etäyhteys johonkin TCP/IP-verkon koneeseen.

TTL	Time-to-Live. Resurssikyselyissä käytetty kenttä, joka määrittelee viestin kulkeman reitin maksimaalisen pituuden hyppyjen määrinä.

Sisältö

1 Johdanto

Vertaisverkot ovat saavuttaneet julkisuutta viime vuosien aikana musiikin levitykseen käytetyn Napsterin saavuttaman suuren suosion vuoksi. Vertaisverkot soveltuvatkin erinomaisesti sisällön jakamiseen. Ne ovat vikasietoisia ja skaalautuvat hyvin, koska verkon toiminta ei riipu yksittäisten palvelinten toiminnasta kuten asiakas–palvelin-arkkitehtuurissa, vaan kaikki vertaisverkon solmuina toimivat koneet voivat toimia sekä sisällön tarjoajina ja käyttäjinä. Lisäksi vertaisverkot ovat kustannustehokkaita, koska verkko koostuu yksittäisistä koneista, jotka yleensä ovat muiden kuin yhden ihmisen tai organisaation hallinnassa. Näin saadaan tehokas ja laaja verkko tiedon jakamiseen ilman suuria kustannuksia kuten palvelinten hankintaa. Vertaisverkoilla saavutetaan myös korkea tiedon saatavuus. Mitä enemmän tarjottavaa sisältöä käytetään, sitä suuremmalle alueelle se leviää. Yleisesti esimerkiksi toiselta solmulta ladattu tiedosto tallennetaan omalle koneelle muiden käytettäväksi. Sisällön tarjoamisen lisäksi vertaisverkkoja käytetään esimerkiksi laskentatehon ja levytilan jakamiseen.

Vertaisverkot eivät ole uusi teknologia, mutta tekniikan kehitys on tehnyt mahdolliseksi niiden laajemman käytön ja siten suosion kasvun. Nykyään koneiden käyttäjiä on paljon, koneet ovat verkottuneet ja pääsy niiden tarjoamiin resursseihin on tehty helpoksi. Myös verkkojen kaistanleveys on kasvanut, mikä mahdollistaa suurenkin datamäärän siirtämisen kohtuullisessa ajassa. Lisäksi koneiden suoritusteho ja levykapasiteetti ovat kasvaneet, jolloin tiedostojen ja ylimääräisen prosessoritehon jakaminen myös muille on mahdollista. [5, s.12]

1.1 Tutkimusongelma

Tässä tutkimuksessa selvitetään, miten ilman keskitettyä solmua toimivasta vertaisverkosta ja sen koneista saadaan mahdollisimman tehokas ja vikasietoinen hakuverkko eri tilanteissa. Yksi tutkimusongelmista on, miten onnistutaan säilyttämään yhteydet muihin koneisiin dynaamisesti muotoutuvassa verkossa. Tutkimuksessa määritellään, milloin solmu lisää uusia naapureita tai poistaa olemassa olevia ja millä perusteella nämä naapurit valitaan. Lisäksi etäisyyksien solmujen välillä tulisi olla mahdollisimman lyhyitä eikä yksikään solmu saisi ylikuormittua ja tulla pullonkaulaksi verkossa, jotta resurssien haku voitaisiin tehdä tehokkaaksi.

Tutkimustyökaluna käytetään Jyväskylän yliopiston Agora Centerissä [2] kehitettyä Chedar-vertaisverkkoalustaa, jolla pystytään löytämään hajautetut resurssit vertaisverkosta. Chedarin käyttämät algoritmit on suunniteltu mukautumiskykyisiksi vaihteleviin tilanteisiin, ja niiden käyttäytymistä arvioidaan tutkimalla niiden vaikutusta verkon topologian muodostumiseen useista työasemista koostuvassa vertaisverkossa.

1.2 Cheese Factory -projekti

Toteuttamani Chedar-vertaisverkkoalusta ja topologian hallinta -algoritmit sekä tämä tutkimus ovat osa Cheese Factory -tutkimusprojektia [6]. Cheese Factory -projekti on Jyväskylän yliopiston Agora Centerissä toimivan Innovations in Business, Communication and Technology (InBCT) -tutkimusprojektin osahanke.

Cheese Factory -projektissa tutkimuksen kohteina ovat kommunikointi vertaisverkossa, vertaisverkkojen käyttäytyminen erityisesti hajautettujen resurssien etsimisessä sekä resurssien tehokas käyttö. Vertaisverkkoihin liittyen projektissa tutkitaan myös tapoja liittyä vertaisverkkoon, potenssijakautuneiden verkkojen muodostumista ja hakua näissä verkoissa. Vertaisverkkosovelluksina projektissa tutkitaan hajautettua datafuusiota, hajautettua laskentaa sekä mobiileja vertaisverkkosovelluksia.

1.3 Tutkielman rakenne

Toisessa luvussa käydään läpi tutkimukseen liittyvää teoriaa vertaisverkkojen osalta ja esitellään Gnutella-vertaisverkkoprotokolla sekä käydään läpi aiheeseen liittyviä vertaisverkkotutkimuksia. Luvussa 3 käsitellään graafeja ja luvussa 4 esitellään tutkimuksessa käytetty Chedar-vertaisverkkoalusta. Chedarin käyttämiin topologian hallinta -algoritmeihin perehdytään luvussa 5. Tutkimusympäristö kuvataan kuudennessa luvussa ja tutkimuksessa käytetyt tutkimustapaukset sekä saadut tulokset ja johtopäätökset esitetään luvussa 7. Yhteenveto tutkielmasta sekä tulevaisuuden tutkimussuunnitelmia on esitetty luvussa 8.

2 Vertaisverkot

Luvussa määritellään vertaisverkko (engl. *Peer-to-Peer*, *P2P*) ja käydään läpi lyhyesti sen historiaa sekä Gnutella-vertaisverkkoprotokolla [11]. Lisäksi luvussa esitellään tähän tutkimukseen liittyviä tutkimuksia.

2.1 Määritelmä

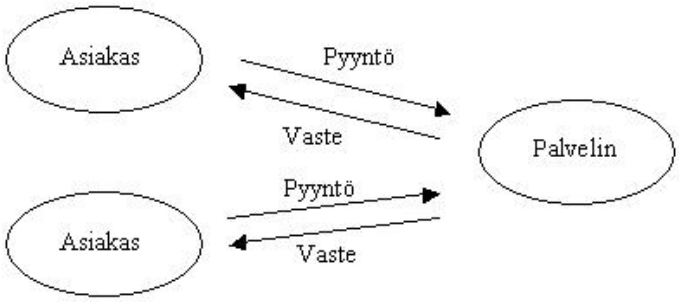
2.1.1 Hajautettu järjestelmä

Hajautettu järjestelmä on kokoelma toisistaan riippumattomia koneita, jotka näkyvät järjestelmän käyttäjälle yhtenä järjestelmänä. Erot koneissa ja niiden tavoissa kommunikoida ovat käyttäjältä piilossa. Käyttäjät ja sovellukset voivat olla vuorovaikutuksessa hajautetun järjestelmän kanssa yhdenmukaisella tavalla huolimatta siitä, missä ja milloin vuorovaikutus tapahtuu. [28, s. 2]

2.1.2 Asiakas–palvelin-arkkitehtuuri

Yleisimmin hajautetuissa järjestelmissä käytössä oleva tiedonhakuarkkitehtuuri on asiakas–palvelin-mallinen (engl. *client-server*). Siinä asiakas ottaa yhteyden tunnettuun palvelimeen ja lähettää palvelimelle pyynnön, joka prosessoidaan. Asiakkaan saatua vastauksen yhteys suljetaan. Arkkitehtuurissa suosittu data on heikommin saatavilla, koska pyynnöt kuormittavat keskitettyä palvelinta. Arkkitehtuuri on myös altis sensuurille ja teknisille vioille, koska palvelimen sulkeminen tai vikaantuminen lamauttaa koko sitä käyttävän järjestelmän. Se sopii kuitenkin hyvin reaaliaikaisiin hakuihin, kuten osakemarkkinatietojen hakuihin sekä järjestelmiin, joissa datan on oltava varmasti luotettavaa tai joissa halutaan hallita datan muutoksia. Datan indeksointi vie kuitenkin aina aikaa, jolloin keskitettyjen hakukoneiden avulla löydettävä tieto on aina vanhempaa kuin vertaisverkosta löydettävä tieto. Jos tieto on hajautettuna useampaan eri paikkaan, tuoreimpana se löydetään vertaisverkosta.

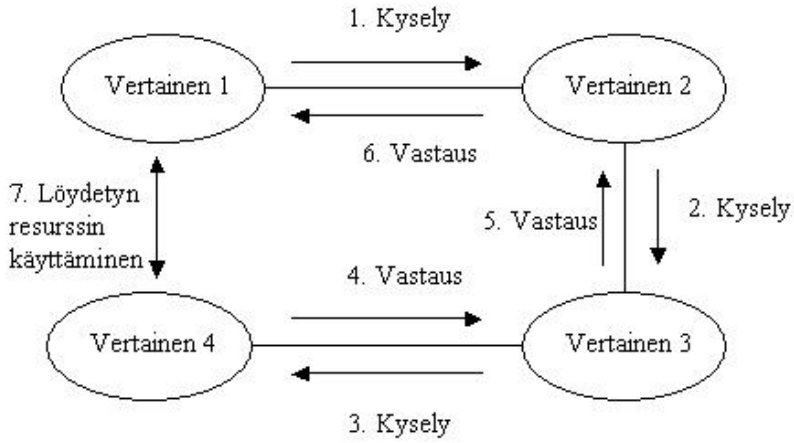
Asiakas–palvelin-malli on esitetty kuvassa 1.



Kuva 1: Kommunikointi asiakas–palvelin-arkkitehtuurissa.

2.1.3 Puhdas vertaisverkko

Puhdas vertaisverkko (engl. *pure Peer-to-Peer*) [26] muodostuu hajautetun verkon koneista eli vertaisista (engl. *peers*), jotka keskustelevat keskenään ilman keskitettyä pistettä. Verkossa ei ole erikseen palvelimia ja asiakkaita, vaan käytännössä verkon jokainen kone toimii sekä palvelimena että asiakkaana (engl. *server+client = servent*). Kone tarjoaa muille verkon koneille palveluita, kuten laskenta-aikaa, tiedonsiirtokapasiteettia, tiedostoja tai levytilaa, ja itse vastaavasti käyttää muiden verkossa olevien koneiden tarjoamia palveluita. Resurssien haku puhtaassa vertaisverkkomallissa on esitetty kuvassa 2.



Kuva 2: Resurssien haku puhtaassa vertaisverkossa.

Vertaisverkon ja asiakas–palvelin-mallin ero voidaan määritellä myös sen avulla, kuka omistaa ja hallinnoi laitteistoa, joka palveluita tarjoaa. Asiakas–palvelin-arkkitehtuurissa omistaja ja hallinnoija on yleensä jokin organisaatio, kun taas vertaisverkkosovelluksissa kukin verkon käyttäjä toimii omistajana ja hallinnoijana. Voidaan siis sanoa, että vertaisverkko on hajautettu myös kustannusten ja hallinnan kannalta. Asiakas–palvelin-malliin verrattuna vertaisverkoissa myös verkon reunoilla olevat koneet voivat tuottaa palveluita kuluttamisen lisäksi. Asiakas–palvelin-arkkitehtuurissa tiedon tuottaminen tapahtuu verkon keskellä palvelimissa. [20, s. 22, 35]

Tiedostojen jakamiseen tarkoitetut Gnutella [11] ja Freenet [9] ovat esimerkkejä puhtaista vertaisverkoista.

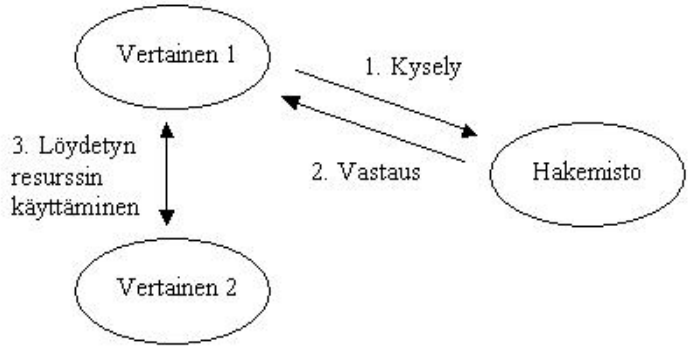
Verkon päällystopologia (engl. *overlay topology*) muodostuu verkon fyysisen topologian päälle. Sitä kutsutaan myös loogiseksi topologiaksi. Päällystopologiassa verkko muodostuu solmuista ja niiden välisistä yhteyksistä ilman, että otetaan kantaa siihen, miten yhteydet sijaitsevat fyysisessä verkossa.

Puhtaat vertaisverkot voidaan jakaa kahteen luokkaan: järjestämättömiin (engl. *unstructured*) ja järjestettyihin (engl. *structured*) vertaisverkkoihin. Järjestämättömässä vertaisverkossa verkon päällystopologiaa ei ole määritelty, vaan solmut voivat liittyä mihin verkon kohtaan tahansa. Tiedonhaku tällaisessa verkossa ei ole kovin tehokasta, mutta toisaalta topologian ylläpito ei vaadi työtä. Järjestämättömässä vertaisverkossa ei voida taata, että verkon yksittäinen resurssi löytyy. Suosittu resurssi löytyy verkosta kuitenkin yleensä useammalta solmulta, jolloin resurssi löydetään helpommin. Haku järjestämättömissä vertaisverkoissa on helppoa ja jokainen solmu voi periaatteessa päättää, miten se saapuvan kyselyn tulkitsee. Gnutella on esimerkki järjestämättömästä vertaisverkosta.

Järjestetyissä vertaisverkoissa kaikille solmuille annetaan identifiointia varten globaali osoiteavain (engl. *identifier*) ja jokainen verkon resurssi saa resurssiavaimen (engl. *key*) verkon avainavaruudesta. Resurssiavain muodostetaan hajautusfunktiolla (engl. *hash function*) joko resurssin nimen tai sisällön perusteella. Verkon päällystopologia muodostetaan siten, että solmu valitsee naapurinsa jokaisesta eri avainavaruuden lohkosta. Järjestetyssä vertaisverkossa tiedonhaku on tehokasta. Mikäli haettava resurssi on verkossa, se myös löytyy. Verkon topologian ylläpito vastaavasti vaatii paljon työtä etenkin, jos verkon solmut poistuvat ja liittyvät verkkoon tiheään. Myös itse haku on rajoitetumpaa kuin järjestämättömissä verkoissa. Esimerkiksi hakusanat ”Madonna.mp3” ja ”Madonna mp3” tarkoittavat järjestetyissä vertaisverkoissa eri asiaa ja johtavat haun eri paikkaan verkossa. Chord [27], Pastry [25] ja CAN [22] käyttävät järjestettyä topologiaa. [14, s. 8-9]

2.1.4 Hybridi vertaisverkko

Puhtaiden vertaisverkkojen lisäksi on olemassa ns. hybridejä vertaisverkkoja (engl. *hybrid Peer-to-Peer*) [26]. Musiikinlevityksestä tutuksi tullut Napster [19] on yksi tällaisista järjestelmistä. Siinä välittäjänä (engl. *broker*) toimiva palvelin ylläpitää hakemistoa käyttäjiltä löytyvistä musiikkitiedostoista. Kukin käyttäjä ilmoittaa välittäjäpalvelimelle tietonsa ja tarjoamansa tiedostot. Käyttäjän halutessa jonkin häneltä löytymättömän tiedoston kysytään välittäjältä, keneltä kyseinen tiedosto löytyy. Välittäjä palauttaa vastauksessa kaikki ne verkon koneet, joilta kyseinen tiedosto löytyy. Käyttäjä valitsee vastauksesta jonkin koneen, ja varsinainen tiedoston lataus tapahtuu suoraan koneelta toiselle. Napsterissa haku tapahtuu asiakas–palvelin-mallin mukaan, mutta tiedoston lataus kuten puhtaassa vertaisverkkomallissa. Kuvassa 3 on esitetty hybridi vertaisverkkomalli.



Kuva 3: Resurssin etsintä hybridissä vertaisverkossa.

2.2 Historia

Vertaisverkko on vanha arkkitehtuuri kommunikoinnissa. Esimerkiksi puhelimet tai IP-reitittimet muodostavat vertaisverkkoja. Alkujaan 1960-luvun lopulla Internet suunniteltiin vertaisverkkojärjestelmäksi. Alkuperäisen ARPANET:in tarkoituksena oli jakaa laskentaresursseja Yhdysvalloissa verkkoarkkitehtuurissa, jossa jokainen isäntä toimii tasavertaisena solmuna. Internet oli myös tuolloin avoimempi ja vapaampi kuin tällä hetkellä eikä palomuurejakaan tunnettu. Jos isäntäkoneesta oli yhteys Internetiin, kuka tahansa Internetissä oleva sai yhteyden isäntäkoneeseen. Vaikka FTP ja TELNET olivatkin asiakas–palvelin-sovelluksia, niin mikä tahansa isäntä saattoi ottaa FTP- tai TELNET-yhteyden toiseen isäntään, eli käytännössä toiminta oli symmetristä. [20, s. ix, 4, 13]

1990-luvun alkupuolella Internetin käyttäjämäärä kasvoi räjähdysmäisesti, mikä muutti Internetin rakennetta merkittävästi. Miljoonat uudet käyttäjät olivat kiinnostuneita sähköpostin lähettämisestä ja WWW-sivujen katselemisesta verkon yksityiskohtien sijaan. Selaimet, kuten monet muutkin Internetin kaupallistumisen aikaan syntyneet sovellukset, perustuivat asiakas–palvelin-protokollaan. Internetistä muokkautui verkko, jossa on suhteellisen pieni määrä erikoistuneita palvelimia ja paljon näitä käyttäviä asiakkaita. [20, s. 8-9]

Nykyiset vertaisverkkosovellukset käyttävät Internetiä niin kuin se alunperin oli suunniteltukin: resursseja keskenään jakavien koneiden kommunikoinnin ilmaisuvälineenä. Ongelmana ovat palomuurit, jotka estävät vapaan kommunikoinnin koneiden välillä, sekä nykyiset asymmetriset yhteydet, kuten ADSL- ja kaapelimodeemiyhteydet, jotka tarjoavat nopeamman yhteyden tiedostojen siirtämiseen omalle koneelle kuin omalta koneelta pois. [20, s. 4, 18-19] Palomuureista aiheutuva ongelma on ratkaistu vertaisverkkojärjestelmissä käyttäen välittäjäsolmuja. Välittäjäsolmut nimensä mukaisesti sitovat yhteen kaksi palomuurien takana sijaitsevaa konetta. Asymmetriset yhteydetkään eivät välttämättä aiheuta ongelmia, koska tiedostoa voidaan ladata useammalta koneelta samaan aikaan, jolloin kukin kone antaa vain osan tiedostosta.

Vuodesta 1979 toiminutta Usenet-uutisryhmää [29] voidaan pitää uusien vertaisverkkojärjestelmien isänä. Usenet on koneiden välillä tapahtuvaan uutisviestien kopiointiin tarkoitettu järjestelmä, jossa ei ole keskitettyä hallintaa. [20, s. 5] Samoin reaaliaikaista keskustelukanavaa, IRC:iä [16], voidaan luonnehtia vertaisverkkojärjestelmien edeltäjäksi.

Olennaisena osana vertaisverkkojen historiaan kuuluu myös hybridi vertaisverkko Napster, joka nosti vertaisverkot yleiseen tietoisuuteen. Napster oli musiikkitiedostojen jakoon tarkoitettu järjestelmä, jonka kehitti opiskelija Shawn Fanning vuoden 1999 alussa. Lyhyessä ajassa Napster saavutti suuren suosion. Vuoden päästä tammikuussa 2000 sillä oli jo noin miljoona käyttäjää ja vuoden 2001 helmikuussa 50 miljoonaa käyttäjää. Musiikkitiedostojen jakamisesta annetun oikeuden päätöksen vuoksi Napsterin välittäjäpalvelin suljettiin vuonna 2001.

2.3 Gnutella

Gnutella on paitsi vertaisverkkojärjestelmä myös puhdas keskittämätön vertaisverkkoprotokolla. Jokainen protokollaa ymmärtävä asiakassovellus on Gnutella-yhteensopiva. Gnutellan asiakasohjelmat käyttävät tiedonsiirtoon TCP-yhteyksiä.

Gnutellan kehitti Nullsoft keväällä 2000 14 päivässä, minkä jälkeen yhtiö julisti sen luvattomaksi projektiksi syytteiden pelossa ja se siirrettiin syrjään. Yksittäiset ohjelmoijat ovat jatkaneet sen jälkeen Gnutellan kehittämistä. [20, s. 95-122]

Koska Gnutellassa ei ole keskitettyä hallintoa tai palvelinta, ketään ei voida osoittaa vastuulliseksi kuten Napsterin tapauksessa, jossa välityspalvelin voitiin sulkea oikeuden päätöksellä. Gnutellassa pitäisi oikeuteen haastaa kaikki sen käyttäjät.

Liittyäkseen Gnutellaan käyttäjän tarvitsee tietää jokin sen solmu eli isäntä. Alussa tieto kulki keskustelupalstojen kautta tai isännän osoitteen pystyi hakemaan joltakin muutamista isäntiä ylläpitävistä WWW-sivuista. Tuota listaa käytiin yksitellen läpi, kunnes yhteys johonkin isäntään onnistui. Koska liittyminen oli hankalaa ja se, mihin verkon osaan liittyminen tapahtui satunnaista, kehitettiin isäntävälimuisti [12]. Muistissa ovat isännät, jotka ovat aina verkossa. Ottamalla yhteyden johonkin isäntään solmu saa listan isännän tietämistä muista isännistä. Muisti helpottaa liittymistä, mutta muodostaa tiukasti klusteroidun verkon. Jokainen verkon solmu ylläpitää tietoa sen tuntemista solmuista, joista ensimmäisen liittymisen jälkeen valitaan solmu, johon yhteys luodaan. Yhteyksiä pyritään pitämään yllä pieni määrä. [20, s .113]

Gnutellassa solmu pitää myös huolta siitä, että sillä on tarpeeksi liikennettä. Solmun pitää päättää, mitkä yhteydet se katkaisee ja mitkä yhteydet se säilyttää, jotta se ei ylikuormitu, muttei myöskään alikuormitu. Ylikuorma kuluttaa liikaa kaistaa, jolloin tiedostojen hakemiseen ei jää kapasiteettia. Alikuorma taas heikentää asiakasohjelman mahdollisuuksia löytää resursseja, koska kyselyt saavuttavat vain pienen osan verkon solmuista.

Gnutellassa viestit yleislähetetään (engl. *broadcast*) ja kukin solmu ylläpitää tietoa välittämiensä viestien tunnisteista (ID) sekä viestin lähettäjästä. Jos sama viesti tulee solmulle uudelleen, viestiä ei enää lähetetä eteenpäin, vaan se poistetaan. Gnutella ei määrittele, miten itse kysely käsitellään tai miten siihen vastataan, vaan sen päättää kukin verkon kone. Jokaisessa viestissä on TTL (*time-to-live*)-arvo, jota vähennetään jokaisessa solmussa. Kun se saavuttaa arvon nolla, viestiä ei enää lähetetä eteenpäin. Näin varmistutaan siitä, ettei viesti tuki verkkoa.

Vastaus reititetään takaisin samaa polkua pitkin kuin mitä kysely saapui. Vastausta vastaavan kyselyn tiedoista katsotaan, keneltä kysely tuli, ja vastaus välitetään sille solmulle, kunnes viesti saapuu kyselyn lähettäjälle.

Gnutella selviytyy hyvin muuttuvasta rakenteesta. Yhteyksien nopeudet verkossa ovat erilaisia, osalla käyttäjistä on 56-kbit modeemi, kun toisilla on nopeat kiinteät yhteydet. Gnutellassa ajan kuluessa nopeat solmut tulevat verkon keskelle, jossa liikenne on suurinta, ja hitaat siirtyvät verkon reunoille. Solmun paikka verkossa ei siis määräydy maantieteellisin perustein.

Gnutella skaalautuu rajattomasti solmujen lukumäärän mukaan, mutta Gnutellan hakualgoritmi ei. Haku ei voi saavuttaa kaikkia verkon solmuja, koska silloin liikennettä tulisi liikaa. Siksi viesteissä joudutaan käyttämään TTL-arvoa. TTL kertoo samalla, mikä on hakukyselyn horisontti eli kuinka paljon verkon solmuja voidaan kyselyllä saavuttaa. Horisonttia kutsutaan myös yhteyden näkymäksi. Yhteyksien poistuessa ja lisääntyessä solmun näkymä suurenee uusien solmujen myötä.

Gnutella noudattaa jossakin määrin potenssijakautuneita verkkoja (luku 3.2), mutta se ei ole puhdas sellainen. Tutkimuksen [23] mukaan marraskuussa 2000 suoritetut mittaukset osoittivat selvästi, että Gnutella-verkot olivat potenssijakautuneita. Myöhemmin samassa tutkimuksessa tehdyt mittaukset osoittavat kuitenkin, että verkko on siirtymässä pois potenssijakautuneen verkon mallista. Uudemmassa verkossa on liian vähän solmuja, joilla on vähän yhteyksiä.

Gnutella on altis ns. vapaamatkustajille (engl. *free riders*), jotka vain käyttävät palveluita eivätkä tarjoa niitä muille. Tutkimuksen [1] mukaan lähes 70% Gnutellan käyttäjistä ei tarjoa muille lainkaan tiedostoja ja melkein 50% kaikista vastauksista tulee 1%:lta isäntiä. Toisen tutkimuksen [23] mukaan 1% eniten yhteyksiä omaavista solmuista tarjoaa 30% saatavilla olevista tiedostoista ja 10% näistä solmuista tarjoaa 71% tiedostoista. Gnutella ei ylläpidä mitään tilatietoja muista solmuista, joten se ei erota vapaamatkustajia palveluita tarjoavista solmuista. Jos verkossa on paljon vapaamatkustajia, kyselyiden pitää vaeltaa pidempi matka, koska välissä on solmuja, jotka eivät ikinä vastaa. Pahimmassa tapauksessa kysely voi epäonnistua vapaamatkustajien vuoksi, koska asetettu viestin horisontti ei riitä. Tuottajien ja kuluttajien välinen tasapaino on ongelma yleisestikin vertaisverkoissa. Aina on olemassa vapaamatkustajia, jotka vain kuluttavat eivätkä tarjoa muille mitään resursseja. [20, s. 238, 292]

2.4 Vertaisverkkojen ominaisuudet

Vertaisverkon etuja ovat sen vikasietoisuus, skaalautuvuus ja tiedon korkea saatavuus. Koska verkossa ei ole keskitettyä pistettä, jonkin koneen poistuminen verkosta ei lamauta koko järjestelmää. Dynaamisen luonteensa vuoksi vertaisverkot asettavat järjestelmille myös haasteita.

Vertaisverkon pitää pystyä selviytymään palvelunestohyökkäyksistä (engl. *denial of service attacks*), joissa järjestelmän kaistanleveys tai suorituskyky ylikuormitetaan, mikä estää palvelun tarjoamisen jossain verkon osassa tai koko verkossa. Kuormituksen ei tarvitse olla tahallinen, vaan se voi olla hetkellinen huippu liikenteessä tai vahingossa tapahtunut väärinkäyttö. [20, s. 273]

Vertaisverkon solmut voivat käyttää protokollaa väärin, esimerkiksi tarjoamalla virheellistä tai laadullisesti heikkoa dataa, perumalla lupauksensa datan säilyttämisestä, kaatumalla ajankohdalla, jolloin solmua tarvittaisiin tai valehtelemalla, että muut solmut käyttävät järjestelmää väärin näillä tavoin. [20, s. 273]

P2P on epäluotettava ympäristö, koska vertaisverkko riippuu yksittäisten käyttäjien henkilökohtaisista resursseista. Nämä resurssit voivat poistua saavuttamattomiin milloin tahansa eri syistä, käyttäjän poistuessa verkosta, sulkiessa koneensa tai satunnaisesta viasta johtuen. Vertaisverkkojärjestelmän pitääkin suhtautua resurssien poistumisiin ja vikoihin normaaleina tapahtumina. [20, s. 203-204]

Järjestelmässä on otettava huomioon sen skaalautuvuus. Napsterin valtava käyttäjämäärä osoitti, kuinka suuri onnistuneen tiedonjakamisjärjestelmän tarve voi olla. Suunnittelussa tulee ottaa huomioon esimerkiksi käytettäessä lokaaleja hakemistoja, että niiden koko riittää eikä tule ylivuotoja, tai jos viestejä yleislähetetään, etteivät ne tuki verkkoa. [20, s. 204]

Vertaiskommunikoinnissa yhteyden nopeus on suurin rajoittava tekijä ja siitä muodostuukin yleensä kommunikoinnin pullonkaula. Ongelma johtuu P2P:n rinnakkaisesta luonteesta: yhteys saattaa olla tarpeeksi nopea kommunikointiin yhden vertaisen kanssa, mutta nopeus ei enää riitä, kun kymmenen vertaista yrittää yhteyttä samanaikaisesti.

Datan etsintä järjestelmästä on vaikeampaa, koska ei ole olemassa keskitettyä hakemistopalvelinta. Viestejä välitetään monta hyppyä solmulta toiselle, ja jokainen hyppy paitsi kasvattaa kokonaiskuormaa myös lisää kyselyn suorittamiseen kuluvaa aikaa. Yhteyden muodostaminen seuraavaan solmuun tai solmun poistumisen havainnointi saattaa myös viedä aikaa. Järjestelmää suunniteltaessa onkin tärkeää vähentää viestien kulkemien hyppyjen määrää leveys- ja rintamahaussa.

2.5 Aiheeseen liittyviä tutkimuksia

Luvussa on rajoituttu tarkastelemaan ainoastaan järjestämättömiin vertaisverkkoihin liittyviä tutkimuksia, koska tutkimuksessa kehitetty Chedar-vertaisverkkoalusta on sellainen.

Khrishna Ramanathan, Kalogeraki ja Pruyne [17] ovat tutkineet, miten vertaisverkossa voi löytää hyviä solmuja. Vertaisverkot muodostuvat ad-hoc-tyylisesti, joten resurssien käyttö on niissä tehotonta. Tutkimuksessa he ehdottavat solmujen intresseihin perustuvaa menetelmää, jossa käyttäen paikallista tietoa parannetaan vertaisverkon suorituskykyä.

Ehdotetussa menetelmässä solmut seuraavat, mitkä verkon solmut vastaavat onnistuneesti niiden lähettämiin resurssikyselyihin. Kun solmu löytää verkosta solmun, joka tarjoaa hyviä tuloksia, se yrittää siirtyä verkossa lähemmäksi sitä luomalla siihen uuden yhteyden. Tämä johtaa solmujen intressien mukaan muotoutuviin klustereihin ja sallii hakujen syvyyksien rajoittamisen, koska samat kiinnostuksen kohteet omaavat solmut ovat lähellä toisiaan. Näin voidaan vähentää viestien määrää verkossa sekä viestien verkossa kulkemien hyppyjen määrää. Menetelmä varaa resursseja tehokkaasti, koska jokainen solmu ylläpitää vähäistä määrää yhteyksiä vain niihin solmuihin, joilla on sama kiinnostuksen kohde. Menetelmä skaalautuu myös hyvin solmujen lukumäärään nähden.

Menetelmässä siirryttäessä otetaan huomioon vain vastauksen lähettäjä, jolloin saatetaan menettää monta hyvää solmua polun välillä. Pahimmassa tapauksessa hyppimistä tapahtuu edestakaisin. Lisäksi jos solmu valitseekin jonkin uuden kiinnostuksen kohteen, sen ympäristössä ei ole muita kuin aikaisemman kiinnostuksen kohteen omaavia solmuja. Menetelmä ei siis välttämättä ole kovin hyvä, kun halutaan hakea monia erilaisia resursseja, eikä sitä käytetty tässä tutkimuksessa.

Pandurangan, Raghavan ja Upfal ehdottavat vertaisverkkojen muodostamiseksi protokollaa [21], joka takaa ilman globaalia tietämystä verkosta, että muodostuneessa topologiassa jokaisen solmun etäisyys toisistaan on pieni. Protokollaa käyttäen uudet solmut päättävät, mihin verkon solmuihin ne ottavat yhteyttä, ja olemassa olevat solmut päättävät, koska ja miten korvaavat kadonneet yhteydet. Tutkimuksessa todistetaan, että tällä menetelmällä saaduissa verkoissa solmujen asteluku on vakio ja verkon halkaisija logaritminen verkon solmuihin nähden. Solmun asteluvulla tarkoitetaan solmun naapureiden lukumäärää.

Verkon solmujen asteluku on välillä [D, C+1], missä D ja C ovat vakioita. Protokollan keskeinen elementti on keskitetty isäntäpalvelin, jonka ylläpitämässä välimuistissa on vakiomäärä solmuja. Uusi solmu ottaa yhteyttä isäntäpalvelimeen, joka antaa välimuistista D satunnaista solmua, joihin solmu ottaa yhteyttä. Tällöin solmusta tulee D-solmu, kunnes se merkitään välimuistiin tai se poistuu verkosta. D-solmun aste on aina D. Mikäli protokolla laittaa solmun välimuistiin, se pysyy siellä, kunnes sen asteluku saavuttaa C:n, jolloin se poistetaan muistista C-solmuna. Tällöin se säilyttää etuoikeutetun yhteyden D-solmuun, joka tulee sen tilalle välimuistiin. Jos C- ja D-solmut eivät ole yhteydessä toisiinsa, lisätään yhteys näiden välille. C-solmu saattaa menettää yhteyksiä muistista poistumisensa jälkeen, mutta sen asteluku pysyy aina vähintään D:n suuruisena.

Protokollan haittana on se, että se käyttää keskitettyä palvelinta, joka tekee siitä alttiimman virheille ja hyökkäyksille, mistä syystä menetelmä ei sovi tähän tutkimukseen. Palvelimesta voi myös muodostua järjestelmän pullonkaula, koska siitä haetaan yhteyksiä jatkuvasti, myös muulloin kuin liityttäessä ensimmäistä kertaa verkkoon.

Alima, Mesaros, Van Roy ja Haridi [3] esittävät tutkimuksessaan verkon hyvyydelle mitan, jolla voidaan määrittää päällysverkon laatu annetulle metriikalle, joka voi olla esimerkiksi kaistanleveys, viive tai hyppyjen lukumäärä. Verkon hyvyyttä arvioitaessa verrataan päällystopologiaa verkon fyysiseen topologiaan. Suurin osa nykyisistä vertaisverkkojärjestelmistä ei takaa, että päällysverkko olisi hyvä. Menetelmä pyrkii siihen, että esimerkiksi käytettäessä hyvyyden mittana hyppyjen lukumäärää fyysisesti toisiaan lähellä olevat solmut olisivat yhteydessä toisiinsa eivätkä Keski-Euroopasta samaan maanosaan lähetettävät viestit kulkisi esimerkiksi Amerikan kautta.

Tutkimuksessa linkin ja koko verkon hyvyyteen vaikuttaa verkon fyysinen topologia. Tutkimuksessamme oletetaan, että fyysistä topologiaa ei välttämättä tunneta, joten algoritmi ei suoraan sovellu käytettäväksi. Lisäksi tutkimuksessa ollaan kiinnostuneita siitä, että päällystopologia on haun kannalta hyvä ja vikasietoinen, eikä fyysistä verkkoa ole otettu huomioon.

Chawathe ym. [7] kehittivät tiedostojen jakamiseen Gia-vertaisverkkojärjestelmän, joka on hajautettu ja järjestämätön kuten Gnutellakin. Järjestelmä saavuttaa kuitenkin kokonaissuorituskyvyn, joka on 3-5 kertaluokkaa parempi kuin Gnutella. Tutkimusryhmän tarkoituksena oli luoda Gnutellan kaltainen järjestelmä, joka kuitenkin kestäisi suuremman kyselynopeuden ja joka toimisi hyvin järjestelmän koon kasvaessa.

Yhtenä järjestelmän tärkeimmistä komponenteista on topologian mukautumisalgoritmi. Sen tarkoituksena on, että solmut, joilla on korkea suorituskyky, olisivat solmuja, joilla on korkea asteluku, ja alhaisemman suorituskyvyn solmut olisivat lähellä niitä. Tätä varten jokainen solmu laskee itselleen tyytyväisyystason, joka kertoo, kuinka tyytyväinen solmu on nykyisiin naapureihinsa. Tyytyväisyystasoon vaikuttavat solmun kapasiteetti ja asteluku. Tyytyväisyystaso saa arvot nollasta ykköseen, ja niin kauan kuin solmu ei ole täysin tyytyväinen naapureihinsa, eli arvo on alle yhden, topologian mukautumisalgoritmi yrittää etsiä solmulle sopivia naapureita, jotta tyytyväisyystaso nousisi. Mikäli solmulle tulee yhteyspyyntö ja sillä on naapureita maksimilukumäärä, sen pitää pudottaa jokin yhteys pois, mikäli haluaa hyväksyä pyynnön. Uuden solmun suorituskyvyn pitää siinä tapauksessa olla enemmän kuin solmun nykyisten naapureiden suorituskyvyt. Suorituskyvyiltään alhaisimmista solmuista pudotettavaksi valitaan solmu, jonka asteluku on suurin, koska sille pudottaminen on vähiten vahingollista. Tosin ehtona pudottamiselle on vielä se, että uudella solmulla on oltava vähemmän naapureita kuin pudotettavalla solmulla.

Tutkimuksessa tarkoituksena on saada solmulle parhaat mahdolliset naapurit, joilla on esimerkiksi paljon prosessoritehoa ja suuri kaistanleveys. Tutkimuksessa ei oteta kantaa siihen, miten paljon solmut verkossa tarjoavat tai kyselevät. Tarkoituksena on vain saada verkon keskelle solmuja, joilta löytyy kapasiteettia suuren liikenteen käsittelyyn. Toisaalta solmulla saattaa olla naapuri, joka ei tutkimuksessa määritellyllä kapasiteetillaan ole mitenkään erinomainen, mutta joka saattaa tarjota solmulle paljon resursseja. Algoritmin mukaan tällainen solmu korvattaisiin helposti jollakin kapasiteetiltaan paljon paremmalla, jolloin saatettaisiin hukata hyvä naapuri.

Lv, Ratnasamy ja Shenker [18] ovat kehittäneet algoritmit, jotka rajoittavat kyselyiden määrää solmuille, jotta nämä eivät ylikuormitu, sekä kehittävät päällysverkkoa dynaamisesti siten, että kyselyt kulkevat niille solmuille, joilla on kapasiteettia käsitellä ne. Algoritmissa jokainen solmu laskee saapuvien ja lähtevien viestien lukumäärää kullekin naapurille erikseen. Tietyin ajanjaksoin solmu tarkistaa, onko se ylikuormittunut. Mikäli on, se valitsee naapurin, jolla on suuri tulevien solmujen aste, ja siirtää sen sellaisen naapurin naapuriksi, jolla on paljon vapaata kapasiteettia. Mikäli vapaata kapasiteettia omaavaa naapuria ei löydy, solmu pyytää kuormittavaa solmua rajoittamaan sen solmulle lähettämien viestien lukumäärää.

Tutkimuksessa pyritään jakamaan liikennettä verkossa tasaisemmin. Myöskään tässä tutkimuksessa ei anneta painoa naapureiden tarjoamille tai välittämille resurssivastauksille. Tässäkin tapauksessa naapuri, joka tarjoaa solmulle paljon resursseja, saattaa joutua useiden siirtojen vuoksi kauaksikin solmusta, jolloin kyselyiden kulkema matka pitenee.

Cooperin ja Garcia-Molinan [8] tutkimuksessa kehitetyssä vertaisverkossa on kahdenlaisia linkkejä: indeksi- ja hakulinkkejä. Indeksilinkkien kautta lähetetään kopio solmun resurssihakemistosta. Näin solmu tietää naapurinsa tarjoamat resurssit ja kyselyiden kulkemista voidaan lyhentää. Hakulinkkien kautta solmut lähettävät resurssikyselyitä. Tutkimuksessa solmu valitsee täysin vapaasti ne solmut, joihin haluaa muodostaa yhteydet. Erilaisille linkkityypeille voidaan antaa todennäköisyydet. Mikäli solmu ylikuormittuu, sen pitää katkaista jokin yhteys. Katkaistava yhteys voi olla eniten kuormaa aiheuttava yhteys, jonka viestien määrä ylittää annetun parametrin rajan, tai voidaan katkaista kaikki ne linkit, joiden kautta tulee enemmän kuin parametrin arvon verran viestejä. Yhteyksiä voidaan katkaista myös kuormituksen tyypistä riippuen. Jos suurin osa kuormasta aiheutuu hakuviesteistä ja hakukuorma ylittää parametrin arvon, katkaistaan kaikki hakulinkit. Vastaavasti voidaan katkaista kaikki indeksilinkit. Kummassakin tapauksessa voidaan tiputtaa myös vain se linkki, joka aiheuttaa eniten kuormaa. Solmu voi monitoroida kuormaa ja katkoa linkkejä jatkuvasti tai tietyin aikavälein.

Tutkimuksessa naapurin pudottamisessa vaikuttaa ainoastaan naapurin aiheuttama kuorma ja määritelty liikenteen määrä. Myöskään tässä ei vaikuta lainkaan se, kuinka hyvä tai huono naapuri on solmulle siltä saatavien vastausten kannalta.

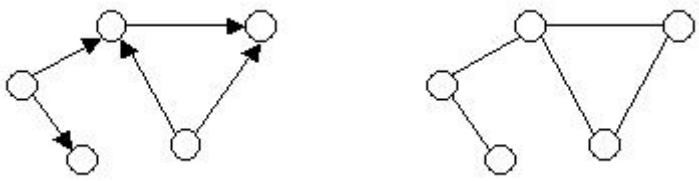
Iles ja Deugo [15] käyttävät geneettistä ohjelmointia kehittääkseen Gnutellan kaltaisen paremman vertaisverkkoprotokollan. Solmun ylläpidettävien yhteyksien lukumäärän määrittelee lauseke CONN, ja jokaiselle olemassa olevalle tai mahdolliselle yhteydelle määritellään sen hyvyyttä kuvaava RANK-lauseke. Gnutella-protokollassa CONN on kiinteä 7 ja RANK = isNeighbor, jossa isNeighbor saa arvon 1, mikäli solmujen välillä on yhteys, ja muuten se saa arvon 0. Tämä osoittaa, että Gnutellassa solmu suosii olemassa olevia yhteyksiä. Tutkimuksessa on geneettisen ohjelmoinnin avulla etsitty lausekkeille arvot, joilla haku verkossa on tehokkainta. CONN-lauseke sisältää mm. solmun naapureiden lukumäärän ja kaistanleveyden. RANK-lauseke käyttää solmujen keräämää statistiikkaa muista solmun tuntemista solmuista. Näitä ovat esimerkiksi muiden solmujen tarjoamien resurssien lukumäärä, lyhyin etäisyys hyppyinä solmuun, solmun lähettämien resurssivastausten lukumäärä ja solmun kautta kulkeneen liikenteen määrä.

3 Graafeista

Luvussa esitellään lyhyesti perusasiat graafeista sekä potenssijakauma.

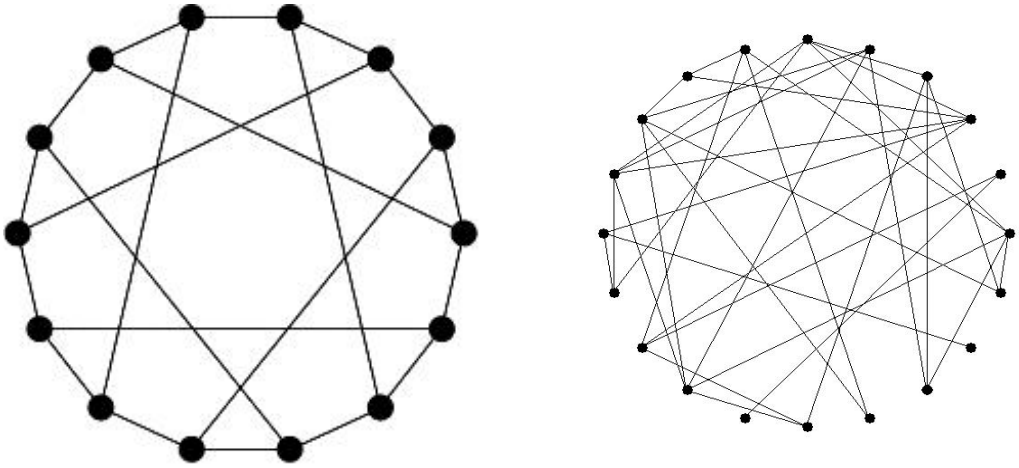
3.1 Graafiteorian perusteet

Graafi muodostuu solmuista (engl. *edges, E*) ja solmujen välillä olevista linkeistä (engl. *vertex, V*). Graafia kutsutaan suunnatuksi (engl. *directed graph*), kun jokaiselle linkille on määriteltynä suunta eli mistä solmusta mihin linkin kautta voi ”kulkea”. Suuntaamattomassa graafissa (engl. *undirected graph*) linkit ovat kaksisuuntaisia. Vertaisverkot voidaan kuvata suuntaamattomina graafeina. Suunnattu ja suuntaamaton graafi on esitetty kuvassa 4.



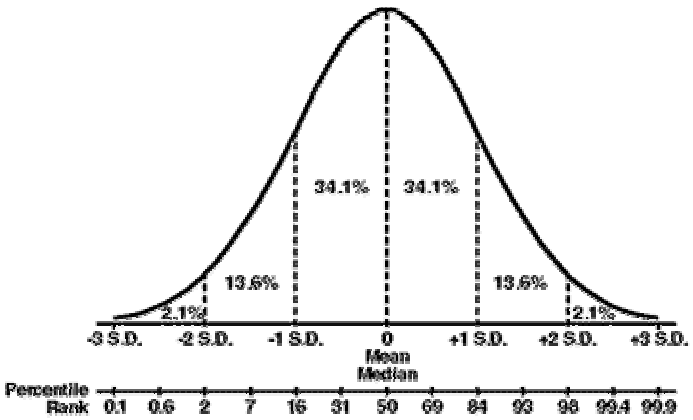
Kuva 4: Vasemmanpuoleisessa kuvassa oleva graafi on suunnattu ja oikeanpuoleisessa graafi on suuntaamaton.

Säännöllisessä graafissa (engl. *regular graph*) jokaisella graafin solmulla on sama määrä linkkejä. Kuvassa 5 on esitetty säännöllinen graafi, jossa linkkien lukumäärä kaikissa solmuissa on kolme. Satunnaisissa graafeissa (engl. *random graph*) linkit sijaitsevat graafissa täysin satunnaisesti, vaikka kaikilla solmuilla on yhtä suuri mahdollisuus saada linkki riippuen todennäköisyyksistä graafia muodostettaessa. Kuitenkin jos linkit sijoitetaan graafiin satunnaisesti, jotkut solmut saavat enemmän linkkejä kuin toiset. Jos verkko on laaja, kaikilla solmuilla on kuitenkin keskimäärin sama määrä linkkejä. Kuvan 5 oikeanpuoleinen graafi on satunnainen. [4, s. 22-23]



Kuva 5: Vasemmanpuoleisessa kuvassa on kuvattu säännöllinen graafi [13], jossa jokaisella solmulla on 3 linkkiä. Oikeanpuoleinen graafi on satunnainen.

Satunnaisen graafin solmujen yhteyksien lukumäärästä piirretty jakauma noudattaa normaalijakaumaa, jota kutsutaan myös Bellin käyräksi (engl. *Bell curve*). Normaalijakauma saadaan kaavalla $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, missä σ on keskihajonta ja μ on odotusarvo. Tällöin suurimmalla osalla solmuja on lähes sama määrä linkkejä kuin keskivertosolmulla. Linkkien lukumäärä voi heitellä odotusarvosta keskihajonnan verran. Kuvassa 6 on normaalijakauman kuvaaja. Kuvaajassa noin 68% datasta, esimerkiksi yhteyksien määrästä, on yhden keskihajonnan päässä odotusarvosta, 95% on kahden keskihajonnan päässä ja kolmen keskihajonnan sisälle mahtuu yli 99% datasta.



Kuva 6: Normaalijakauma [10].

3.2 Potenssijakautuneet verkot

Monet luontoon liittyvät verkot, kuten soluissa olevat molekyylit, ruokaketjuverkostot, ihmisten sosiaaliset ryhmät ja tietoliikennejärjestelmät kuten Internet ja WWW, muodostavat niin kutsuttuja potenssijakautuneita (engl. *power-law*) verkkoja. Näissä verkoissa suurimmalla osalla solmuja on vain muutamia yhteyksiä ja pienellä osalla keskussolmuja on suuri määrä yhteyksiä.

Eräs tapa, jolla potenssijakautunut verkko saadaan aikaiseksi, on Barabásin ja Albertin kehittämä malli, jossa jokaisella ajanhetkellä lisätään yksi uusi solmu verkkoon. Jokainen uusi solmu luo yhteydet verkon kahteen solmuun. Todennäköisyys, jolla yksittäinen solmu valitaan, on verrannollinen sen yhteyksien lukumäärään. Mikäli jollakin solmulla on kaksi kertaa enemmän yhteyksiä kuin toisella, se valitaan kaksi kertaa todennäköisemmin kuin toinen. Malli suosii siis yhteyksiä luodessaan niitä verkon solmuja, joilla on eniten yhteyksiä. Tästä seuraa, että verkkoon muodostuu muutama keskussolmu, jolla on paljon yhteyksiä. [4, s. 86-87]

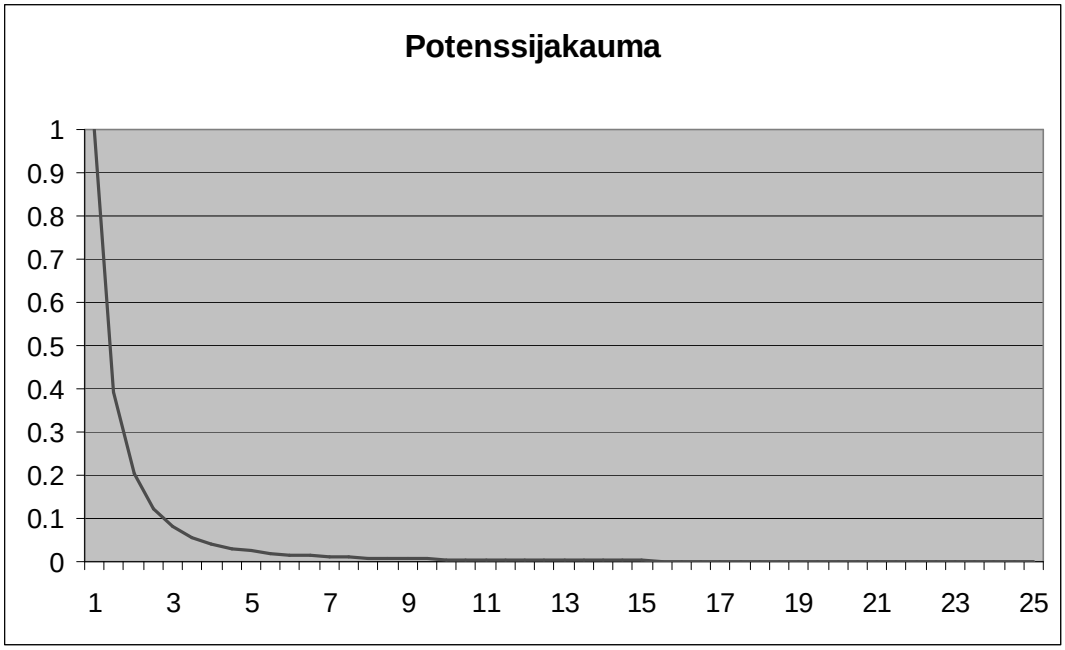
Potenssijakautuneissa verkoissa niiden solmujen osuus, joiden yhteyksien lukumäärä on L, on verrannollinen funktioon

$$f(x) = L^{-k}, \tag{1}$$

missä k on verkosta riippuva vakio.

Potenssijakaumassa ei ole huippua, vaan se on jatkuvasti laskeva käyrä, joka osoittaa, että monet pienet tapahtumat esiintyvät yhdessä suurien tapahtumien kanssa. Jokaisella jakaumalla on erityinen eksponentti k, joka kuvaa verkon astelukua. Esimerkiksi web-sivujen toisiin sivuihin osoittavat linkit noudattavat potenssijakaumaa, jossa asteluku on 2.1 ja ulosmenevien linkkien asteluku on 2.5. [4, s. 67]

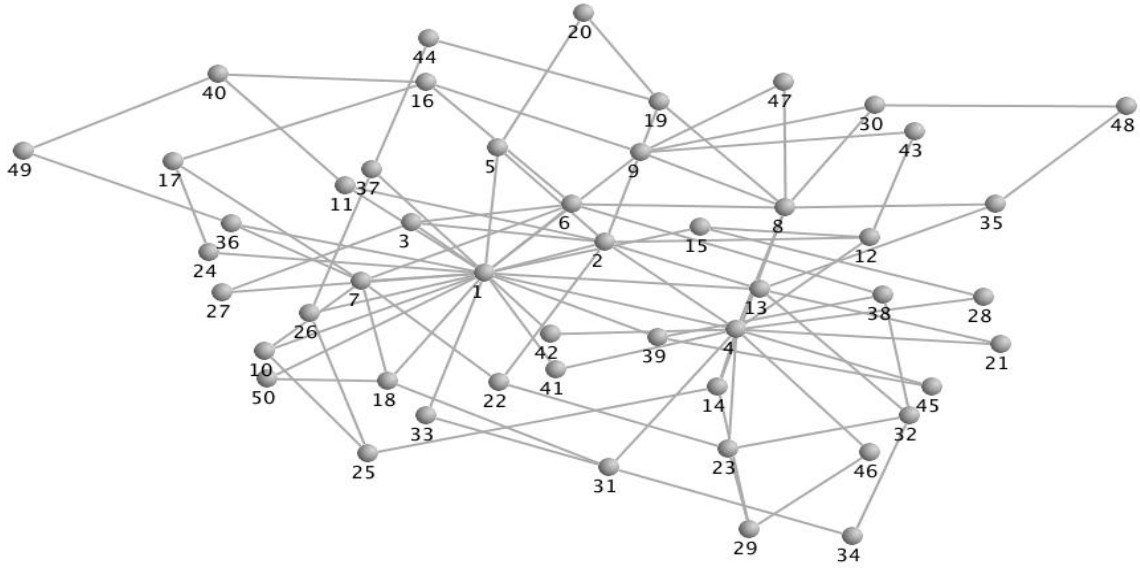
Kuvassa 7 on esitetty funktiolla (1) saatu potenssijakauma, jossa astelukuna on 2.3.



Kuva 7: Potenssijakauma, jossa asteluku *k* on 2.3.

Satunnaisissa verkoissa normaalijakauman huippu (kuva 6) osoittaa, että suurimmalla osalla solmuja on sama määrä yhteyksiä ja että näistä solmuista poikkeavat solmut ovat hyvin harvinaisia. Satunnaisilla verkoilla solmujen yhteydellisyydellä on tunnusomaisuusasteikko (engl. *characteristic scale*), jota edustaa keskivertosolmu ja jonka määrää astejakauman huippu. Sen sijaan potenssijakautuneisuus osoittaa, että todellisissa verkoissa ei ole olemassa tunnusomaista solmua, siksi potenssijakautuneita verkkoja kutsutaankin skaalattomiksi (engl. *scale-free*) verkoiksi. [4]

Kuvassa 8 on esitetty 50 solmun potenssijakautunut vertaisverkko. Siinä solmut 1, 2, 3 ja 4 ovat selvästi verkon keskussolmuja, joilla on eniten yhteyksiä muihin solmuihin.



Kuva 8: Potenssijakautunut vertaisverkko, jossa keskussolmuina ovat solmut 1, 2, 3 ja 4.

Potenssijakautuneet verkot ovat yleisesti vakaita ja vikasietoisia, mutta alttiita satunnaisille romahduksille. Koska suurimmalla osalla solmuja on pieni määrä yhteyksiä, niistä riippuu hyvin vähän. Mistä tahansa scale-free-verkosta voidaan poistaa satunnaisesti 80% solmuista ilman, että verkko jakautuu. Todennäköistä on, että satunnaisesti valittu solmu kuuluu niihin solmuihin, joilla on vähiten yhteyksiä, koska niitä on verkossa lukumäärällisesti eniten. Nämä vähän yhteyksiä omaavat solmut eivät ole kuitenkaan tärkeitä verkon yhtenäisyydelle. Sen sijaan kohdistettu hyökkäys, jossa poistetaan eniten yhteyksiä omaavat solmut on tuhoisa. Tällöin verkko jakautuu toisistaan eristyksissä oleviin klustereihin. Tosin tämäkin edellyttää sitä, että 5-15 % suurimmista solmuista poistetaan käytöstä samanaikaisesti. Potenssijakautuneet verkot selviävät hyvin satunnaisista solmun vioista, mutta ovat alttiita hyvin suunnitelluille hyökkäyksille. [24], [4, s.113-118]

Resurssien haku on nopeampaa ja tehostuu potenssijakautuneessa vertaisverkossa verrattuna satunnaiseen verkkoon, koska solmujen väliset etäisyydet ovat lyhyemmät. Verkon keskussolmut keräävät paljon naapureita, jolloin verkon lävistäjä lyhenee. Näin viestien kulkema matka lyhenee ja kuormitus vähenee. Keskussolmujen ansiosta potenssijakautuneessa verkossa on myös solmujen välillä useita eri reittejä, joita voidaan hyödyntää esimerkiksi yhden keskussolmun kuormittuessa.

4 Chedar-vertaisverkkoalusta

Tässä luvussa esitellään tutkimuksessa käytettyä resurssien etsimiseen tarkoitettua Chedar-vertaisverkkoalustaa.

4.1 Toiminta

Chedar on resurssien etsintään kehitetty Gnutellan kaltainen puhdas vertaisverkkoalusta. Sen avulla verkon koneet eli solmut pystyvät kommunikoimaan keskenään TCP-yhteyksin täysin itsenäisesti ilman keskitettyä pistettä. Chedar on kehitetty dynaamiseen ympäristöön, missä verkon solmut voivat liittyä verkkoon, siirtyä verkossa tai poistua verkosta milloin tahansa ilman, että järjestelmä sen takia lamaantuu. Siinä ei ole keskitettyä pistettä, joten se on vastustuskykyinen kohdistetuille hyökkäyksille ja virheille. Satunnaiset laitteistovirheet näyttävät verkossa siltä, kuin jokin solmu vain poistuisi verkosta. Chedar on toteutettu Java-ohjelmointikielellä.

Gnutellasta poiketen Chedariin on määritelty saatuihin resurssivastauksiin perustuva solmun hyvyys ja resurssivastaus pyritään aina reitittämään kyselyn lähettäjälle. Lisäksi verkon topologiaa ja liikennettä hallinnoidaan Solmun ohitus -algoritmilla ja Kuormituksen arviointi -algoritmilla, joita käsitellään luvussa 5. Chedarissa voidaan hakualgoritmina käyttää mitä tahansa siihen toteutettua hakualgoritmia toisin kuin Gnutellassa, jossa käytetään yleislähettävää (engl. *broadcast*) algoritmia.

Chedar tallentaa tietoja aktiivisista yhteyksistä sekä historiatietoja vanhoista yhteyksistä. Historiassa on tallessa yhteystietoina myös ne verkon solmut, jotka solmu on saanut tietoonsa naapureidensa välityksellä. Solmu pitää yllä aktiivisista yhteyksistä eli naapurisolmuistaan niiden naapureiden lukumäärää sekä tietoa niiden tarjoamista resursseista, kaikista yhteyksistä osumatietoja sekä tietoa, koska yhteys on viimeksi vastannut ja koska sinne on lähetetty pyyntö sekä onko tuo lähetys onnistunut vai epäonnistunut.

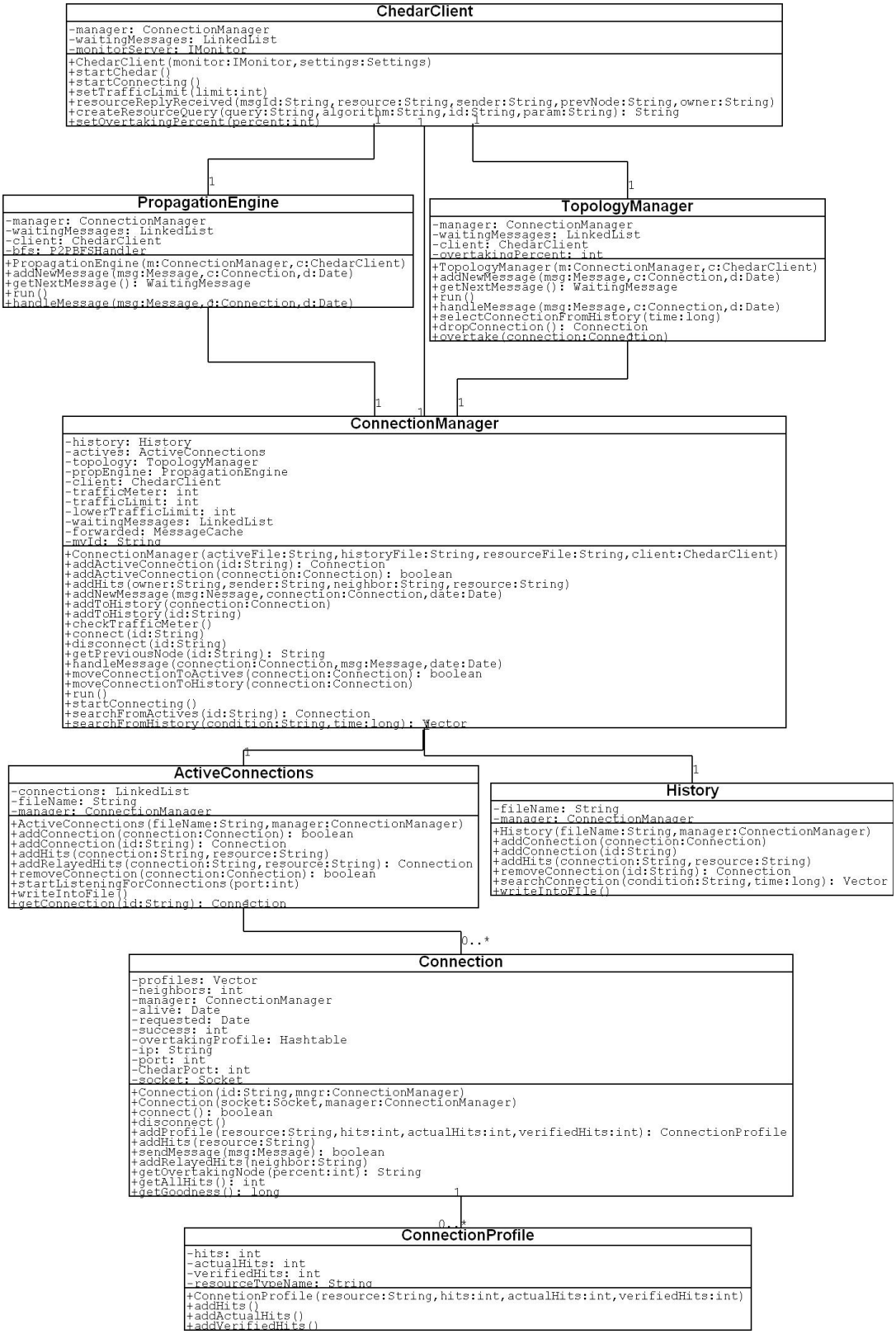
Osumatietoja on kolmenlaisia. Ensimmäistä kasvatetaan joka kerta, kun yhteydeltä saadaan vastaus resurssikyselyyn. Toista kasvatetaan, kun yhteyden resurssia ollaan käytetty. Kolmatta päivitetään, kun halutaan arvioida käytetyn resurssin laatua sisällön perusteella. Lisäksi yhteyksistä pidetään yllä niiden naapureiden välittämiä osumia, jotka kertovat paljonko vastauksia on saapunut koneelle sen yhteyden kautta muilta koneilta. Kun vastausviesti saapuu koneelle, se lisää vastauksen lähettäneen yhteyden osumia. Sen lisäksi kasvatetaan viestin välittäneen yhteyden naapurin välitettyjä osuma-arvoja ja Solmun ohitus -algoritmilla tarkistetaan hypätäänkö yhteys yli.

Chedarissa ei ole kiinnitettyä lukumäärää solmun yhteyksille, vaan yhteyksiä lisätään ja poistetaan solmun kautta kulkevan liikenteen mukaan. Mikäli viestien koko määritellyssä ajassa ylittää käyttäjän määrittelemän arvon, Chedar tiputtaa pois huonoimman yhteyden. Mikäli liikenne on vähäistä, Chedar pyrkii aktiivisesti luomaan uusia yhteyksiä.

Asiakasrajapinnan kautta Chedar-alustaa voivat käyttää erilaiset vertaisverkkosovellukset, kuten esimerkiksi luvussa 4.4 esitelty Peer-to-Peer Studio (P2PStudio).

4.2 Keskeiset luokat

Kuvan 9 luokkakaaviossa on esitettynä algoritmien kannalta Chedarin tärkeimmät luokat ja niiden attribuutit ja metodit. Asetus- ja saantimetodeja ei muutamaa poikkeusta lukuun ottamatta mainita luokkakaaviossa lainkaan.



Kuva 9: Luokkakaavio Chedarin pääluokista.

4.2.1 Connection

`Connection`-luokka sisältää tiedot yhteydestä. Se tallentaa osuma-arvot yhteyden tarjoamien resurssien mukaan eriteltyinä `ConnectionProfile`-olioihin sekä lisäksi yhteyden naapureiden välittämät osuma-arvot. Yhteyden naapureiden osumia säilytetään `overtakingProfile`-taulukossa. `Connection`-luokassa on tieto naapureiden lukumäärästä (`neighbors`) sekä tallessa aika (`requested`), jolloin Chedar yritti lähettää yhteyspyynnön yhteydelle, sekä tieto siitä, onnistuiko tämä pyyntö vai ei (`success`). Solmun saadessa viestejä yhteyden kautta päivitetään yhteyden elossaolosta kertovaa aikaa (`alive`).

4.2.2 ActiveConnections ja History

`ActiveConnections` sisältää solmun aktiiviset yhteydet eli listan `Connection`-olioista. Se huolehtii uusien yhteyksien lisäämisistä ja poistamisista. `History`-luokka ylläpitää XML-puuta kaikista solmun entisistä yhteyksistä sekä sen naapureilta tietoonsa saamistaan yhteyksistä. Yhteys ei voi olla samaan aikaan sekä aktiivisissa että historiassa. Aktiiviset yhteydet sekä historiassa olevat yhteydet tallennetaan XML-tiedostoihin, joista ne voidaan lukea solmun liittyessä verkkoon.

4.2.3 ConnectionManager

`ConnectionManager`-luokka huolehtii aktiivisista sekä historiassa olevista yhteyksistä lisäten ja poistaen niitä `TopologyManager`-luokan pyyntöjen mukaan. Se myös tallentaa tiedot välitetyistä viesteistä (`forwarded`) sekä käsittelee kaikki saapuvat viestit välittäen ne kaikille kyseistä viestiä haluaville luokille. `ConnectionManager`-luokassa on kulkevan liikenteen laskuri (`trafficMeter`), joka laskee solmun kautta annetussa ajassa kulkevien resurssiviestien koot yhteen. Mikäli laskurin arvo ylittää määritellyn liikenteen rajan, poistetaan yhteyksiä. Muuten yhteyksiä lisätään. `ConnectionManager` lisää osumia yhteyksille `ChedarClient`-luokan pyytäessä eli joka kerta, kun solmu saa vastauksen lähettämäänsä resurssikyselyyn.

4.2.4 TopologyManager

`TopologyManager` huolehtii uusien luotavien yhteyksien ja poistettavien yhteyksien valinnasta sekä naapureiden ohituksista `ConnectionManagerin` pyytäessä. Sen vastuulla on valita, mitkä solmut luodaan ja mitkä poistetaan sekä suoritetaanko ohitus ja mihin solmuun. `TopologyManager` hoitaa yhteyspyyntöjen ja -vastauksien käsittelyn sekä naapureiden ja naapureiden resurssien kyselyt ja vastaukset.

4.2.5 PropagationEngine

`PropagationEngine` on vastuussa resurssiviestien käsittelystä. Se välittää saapuvat resurssikyselyt ja -vastaukset käsiteltäväksi viestissä määritellylle algoritmille. Algoritmi päättää, kenelle viesti välitetään, ja tarvittaessa luo kyselyyn vastauksen ja palauttaa `PropagationEngine`-luokalle välitettävän viestin sekä yhteydet, joille viesti lähetetään. Toisin kuin Gnutellassa, Chedarissa voi hakualgoritmina olla muukin kuin yleislähettävä algoritmi.

4.2.6 ChedarClient

`ChedarClient` aloittaa yhteyksien muodostamisen, kun sovellus käynnistetään. `ChedarClient` on Chedarin käyttöliittymä ja tarjoaa API-rajapinnan vertaisverkkosovellusten käyttöön.

4.3 Viestit

Verkon topologian hallintaan liittyviä viestejä ovat yhteyspyynnöt ja -vastaukset, naapureiden kyselyt ja vastaukset sekä koneen tarjoamien resurssien kyselyt ja vastaukset. Näiden viestien avulla luodaan yhteyksiä sekä kerätään niistä tietoa. Varsinainen pääasia on resurssikyselyiden ja vastauksien käsittely. Kun tarvitaan jotakin resurssia, lähetetään resurssikysely solmun naapureille kulloinkin halutun hakualgoritmin avulla. Mikäli resurssi löytyy, solmu lähettää vastauksen, joka kuljetetaan alkuperäisen kyselyn lähettäjälle.

Solmu muodostaa yhteyden lähettämällä toiselle solmulle `ConnectionRequest`-viestin. Yhteyspyyntöihin vastaaminen tapahtuu `ConnectionReply`-viestillä, jossa kerrotaan, hyväksyykö solmu pyydettävän yhteyden vai ei.

Solmun yhteyden naapureiden eli naapurisolmun naapureiden kyselyyn on `NeighborListRequest`-viesti, jonka lähettäjä laittaa viestin kenttään mukaan omien naapureidensa tunnisteet (ID:t). Toinen solmu vastaa `NeighborListReply`-viestillä, jolla välitetään solmun naapurit.

Yhteyden tarjoamia resurssityyppejä kysellään `ServiceListRequest`-viestillä, johon vastataan `ServiceListReply`-viestillä. Kyselyssä on mukana kyselyn lähettäjän tarjoamat resurssit ja vastauksessa vastaajan resurssit.

Resurssien kysely tapahtuu `ResourceRequest`-viestillä. Solmun tarjotessa kyseltävää resurssia se vastaa `ResourceReply`-viestillä, jossa on mukana kaikki kyselyä vastaavat solmun tarjoamat resurssit.

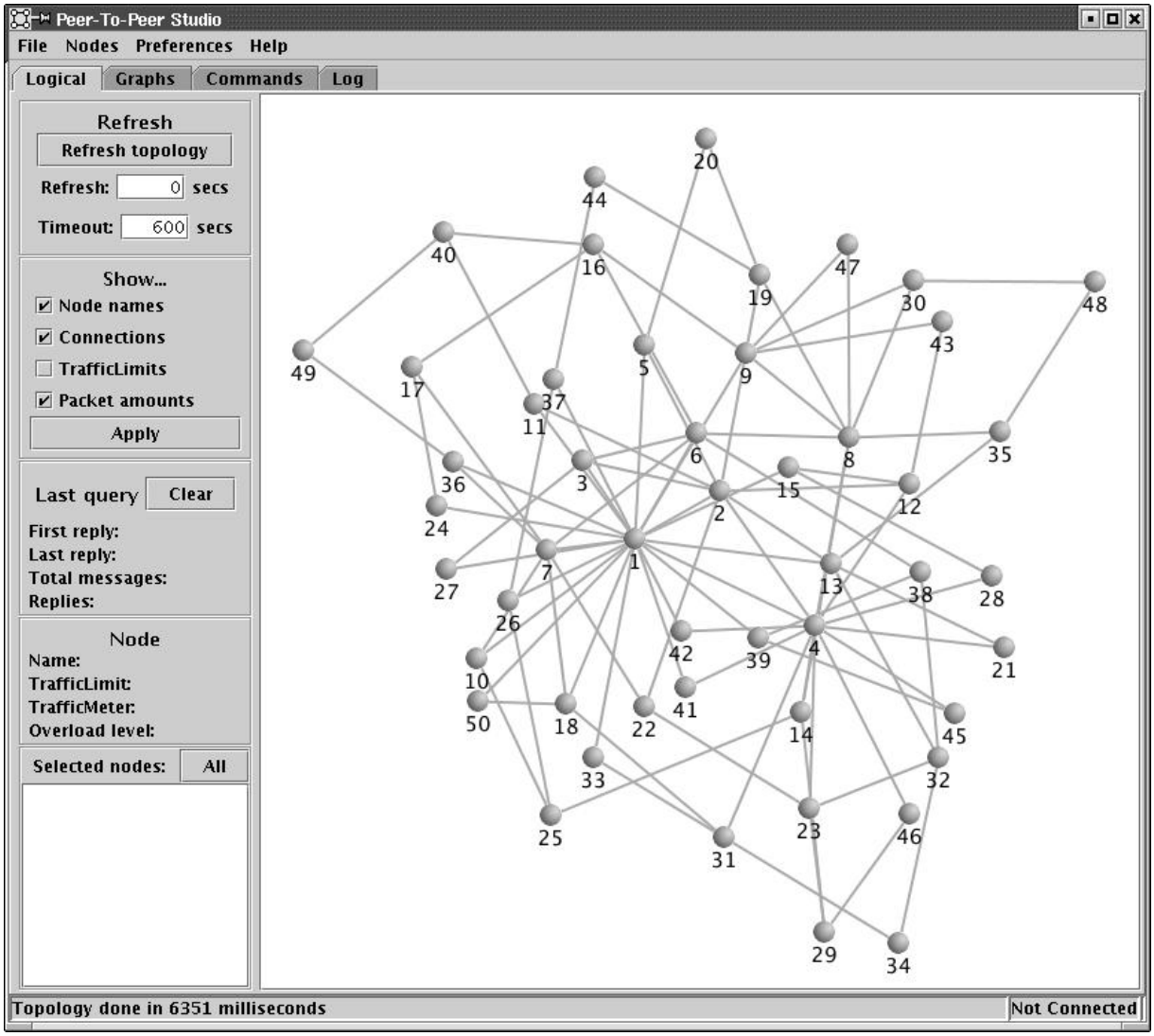
Chedarissa on pyritty takaamaan `ResourceReply`-viestin välittäminen kyselyn lähettäjälle mahdollisimman yksinkertaisella menetelmällä. Jokainen solmu tallentaa välittämistään resurssikyselyviesteistä kyselyn tunnisteen sekä kaksi edellistä solmua, joiden kautta viesti solmulle saapui. Vastausviesti reititetään takaisin kyselyn lähettäjälle samaa reittiä kuin kysely saapui eli lähetetään sille solmulle, jolta kysely tuli. Mikäli yhteys solmuun ei ole enää kunnossa, esimerkiksi naapuri on poistunut verkosta, yritetään luoda yhteys edelliseen solmuun ja lähettää viesti sille. Mikäli tämäkään ei onnistu, viimeisenä keinona yritetään ottaa yhteys kyselyn lähettäjään ja välittää vastausviesti suoraan lähettäjälle.

4.4 Solmun monitorointi ja P2PStudio

Chedar-solmua voidaan monitoroida `iMonitor`-rajapinnan kautta. Tutkimuksessa monitorointityökaluna käytetään Tietotekniikan laitoksen syksyn 2002 Guardian-sovellusprojektin toteuttamaa P2PStudio-ohjelmaa.

Monitorointityökalun avulla Chedariin tehtyjen algoritmien vaikutusta verkon topologiaan voidaan seurata reaaliaikaisesti. Sen avulla välitetään Chedar-solmulle algoritmeissa tarvittavat muunneltavissa olevat muuttujien arvot ja voidaan graafisesti tarkastella vertaisverkon tilaa.

Kuvassa 10 on esitetty P2PStudio-ohjelman käyttöliittymä, jossa on piirrettynä verkon looginen topologia.



Kuva 10: P2PStudion käyttöliittymä, jossa on graafisesti esitettynä 50 vertaisen muodostama verkko.

5 Topologian hallinta -algoritmit

Tässä luvussa esitellään tutkimuksessa topologian hallintaan käytetyt Solmun ohitus, Solmun valinta, Solmun poisto ja Kuormituksen arviointi -algoritmit.

5.1 Solmun ohitus

Solmun ohitus (engl. *overtaking*) -algoritmia käytetään topologian optimoinnissa. Algoritmin tarkoituksena on, että solmu siirtyy lähemmäksi hyviä solmuja eli niitä, joilta saadaan eniten vastauksia, hyppäämällä yli yhden naapurin. Näin kyselyt kuormittavat vähemmän verkkoa, koska solmun etäisyydet sitä kiinnostavia resursseja tarjoavista solmuista lyhenevät. Solmu ei kuitenkaan hyppää ja muodosta yhteyttä suoraan resurssia tarjoavaan solmuun, vaan siirtyy vähitellen lähemmäksi, jolloin se ei menetä polun varrella olevaa mahdollisesti resurssia tarjoavaa parempaa solmua.

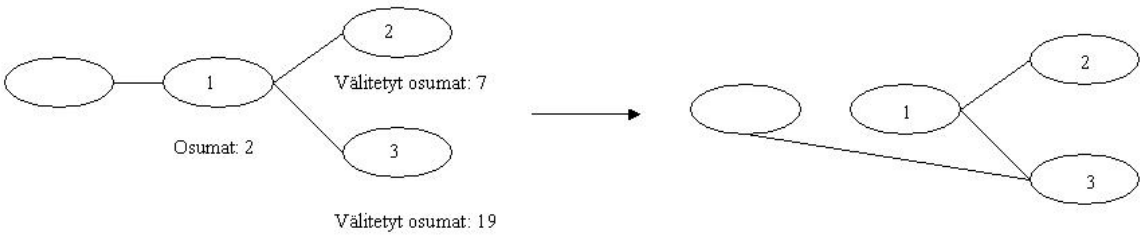
Vastausviestin saapuessa ja osumatietojen lisäyksien jälkeen Solmun ohitus -algoritmilla katsotaan, onko jonkin solmun yhteyden naapurin suhteellinen osuus prosentteina suurempi kuin määritelty ohitusprosentin arvo. Jos on, luodaan yhteys tähän naapuriin ja tiputetaan nykyinen yhteys. Prosentuaalinen osuus saadaan jakamalla yhteyden osumat tai naapurin välittämät osumat yhteyden osumien sekä sen kaikkien naapureiden suhteellisien osumien yhteenlasketulla summalla.

Esimerkiksi jos solmun yhteydellä on taulukon 1 mukaiset tiedot eli yhteydellä itsellään on 2 osumaa ja sen naapureiden välittämät osumat ovat 17 ja 19, niin vastaavasti solmun osumien prosentuaalinen osuus on 7% ja sen naapureiden 25% ja 68%. Mikäli algoritmin ohitusprosentiksi olisi määritelty 60%, niin naapurin 2. naapuriin luotaisiin yhteys ja tiputettaisiin nykyinen yhteys pois. Algoritmia testataan erilaisilla raja-arvoilla ja tutkitaan näiden raja-arvojen vaikutusta verkon topologiaan. Algoritmin pseudokoodi löytyy tutkimuksen liitteestä 1.

Yhteydet	Osumatiedot	Prosentuaalinen osuus
Solmun naapuri	2	2/28 ≈ 0.0714 ≈ 7%
Solmun naapurin 1. naapuri	7	7/28 = 0.25 ≈ 25%
Solmun naapurin 2. naapuri	19	19/28 ≈ 0.679 ≈ 68%
Yhteensä	28	

Taulukko 1: Solmun ohitus -algoritmin käyttämät tiedot yhteydestä.

Kuvassa 11 on havainnollistettu algoritmin toimintaa annetussa esimerkkitapauksessa. Kuvassa solmulla on yhteys solmuun 1, joka on siis solmun naapuri, ja solmut 2 ja 3 ovat naapurin naapureita.



Kuva 11: Solmun ohitus -algoritmin toiminta.

5.2 Solmun valinta

Liittyessään vertaisverkkoon Chedar-solmu etsii ensimmäisenä luotavia yhteyksiä niiden yhteyksien joukosta, joihin se oli ennen poistumistaan verkosta liittyneenä. Parhaassa mahdollisessa tapauksessa solmu saa yhteydet muodostettua kaikkiin niihin solmuihin, joihin se oli liittyneenä ennen poistumistaan verkosta. Nämä yhteydet on tallennettuna tiedostoon, josta `ActiveConnections`-luokka lukee ne ja alkaa muodostaa yhteyksiä. Mikäli yhteydenmuodostus epäonnistuu, yhteys siirretään historiaan.

Mikäli solmun pitää lisätä uusi yhteys, se etsii sopivaa yhteyttä historiatiedoista. Ensin yhteyttä yritetään muodostaa niihin yhteyksiin, joista on osumatietoja tallessa. Koska yhteyttä ei haluta yrittää samaan solmuun, johon yhteys on juuri tiputettu pois, haetaan uutta yhteyttä vain niiden yhteyksien joukosta, joihin ei ole yritetty ottaa yhteyttä määrätyn ajan sisällä. Jollei uuden yhteyden luominen onnistunut, etsitään historiasta seuraavaksi yhteyksiä vain sen perusteella, ettei niihin ole juuri yritetty ottaa yhteyttä tai ettei niistä ole lainkaan tietoja aikaisemmista yrityskerroista. Solmun valinta on pseudokoodina liitteessä 2.

5.3 Solmun poisto

Valittaessa poistettavaa yhteyttä, esimerkiksi solmun ylikuormittuessa, pyritään poistamaan huonoin mahdollinen yhteys eli yhteys, jolla on pienin hyvyys. Yhteyden hyvyys määritellään yhteyden saamien kaikkien osuma-arvojen summana, johon on lisätty yhteyden naapureiden välittämät osumat. Hyvyys voidaan laskea kaavalla

$$Hyvyys = osumat + välitetyt\ osumat \tag{2}$$

Solmun poisto -algoritmi löytyy liitteestä 3.

5.4 Kuormituksen arviointi

Chedarissa ei ole määritelty solmun naapureiden lukumäärää, vaan naapureita lisätään ja poistetaan solmun kautta kulkevan liikenteen mukaan. Kuormituksen arviointi -algoritmi seuraa liikenteen määrää kilotavuina minuutissa (`trafficMeter`) ja sen ylittäessä annetun rajan tiputetaan Solmun poisto -algoritmilla yksi yhteys pois. Mikäli liikenteen määrä jää alle määritellyn rajan, algoritmi pyrkii muodostamaan uusia yhteyksiä käyttäen Solmun valinta -algoritmia.

Algoritmissa on käytössä ylä- ja alaraja liikenteen määrälle. Alaraja (`lowerTrafficLimit`) määritellään ylärajan (`trafficLimit`) avulla niin, että se on 80% ylärajan arvosta. Tutkielmassa testataan algoritmia erilaisilla raja-arvoilla. Algoritmi on kuvattuna pseudokoodina liitteessä 4.

6 Tutkimusympäristö

Vertaisverkkona tutkimuksessa käytettiin neljää tietokonetta, joissa kussakin oli kymmenen Chedar-solmua käynnissä. Tutkimuksessa käytettävässä vertaisverkossa oli siis kokonaisuudessaan neljäkymmentä solmua. Koneiden käyttöjärjestelmänä oli Linux versio 2.2.20 ja Javasta oli käytössä J2SE versio 1.4.2. Koneet olivat yhteydessä toisiinsa 100Mbit/s Ethernet-verkon välityksellä.

Tutkimusympäristössä sijaitsevia Chedar-solmuja monitoroitiin luvussa 4.4 kuvatulla P2PStudiolla.

Tutkittavat solmut jaoteltiin neljään eri ryhmään niiden tarjoamien resurssien määrän sekä niiden lähettämien kyselyiden määrän mukaan. Tarjottavien resurssien perusteella solmut jaettiin kahtia, jolloin vähän tarjoavat tarjosivat 1-5 resurssia ja paljon tarjoavat 15-20 resurssia. Resurssit nimettiin numeroin. Tämän jälkeen solmut jaettiin vielä kahtia kyselyiden perusteella, niin että puolet paljon tarjoavista kyseli vain vähän ja toinen puoli kyseli paljon. Vastaavasti vähän tarjoavat jaettiin samalla tavoin.

Lähetettävien resurssikyselyjen luonnissa kyseltävä resurssi valittiin tasaisesti jakautuneena satunnaislukuna. Näin todennäköisyys, että resurssi löytyy paljon tarjoavalta solmulta, on suurempi kuin, että kyseltävä resurssi löytyy vähän tarjoavalta. Tutkimuksessa tämä riitti, sillä yksittäisten tiedostojen suosittuvuus ei ollut tärkeää. Näin kasvaa todennäköisyys, että resurssi löytyy solmulta, jolla on paljon tiedostoja, koska näillä solmuilla on suhteellisesti paljon mahdollisista resursseista.

Resurssikyselyn lähettäjä valittiin solmuista satunnaisesti niin, että paljon kyselyitä lähettävillä solmuilla oli kymmenkertainen todennäköisyys tulla valituksi kuin vähän kyselyitä lähettävillä.

Resurssikyselyitä lähetettiin verkkoon viisi kyselyä kerrallaan P2PStudiolla, minkä jälkeen odotettiin, että kyselyt oli käsitelty ennen seuraavien lähettämistä. P2PStudio lähetti luodut kyselyt kyselyn lähettäjäksi merkitylle solmulle, joka käsitteli viestin ja laittoi sen verkkoon. Viidenkymmenen kyselyn välein P2PStudio päivitti ja tallensi verkon topologiakuvan sekä naapureiden lukumäärien jakaumasta piirretyn kuvaajan. Kyselyitä lähetettiin yhteensä 1250 käyttäen yleislähettävää BFS-algoritmia, jossa TTL-arvona oli kuusi. Tällöin kyselyt saavuttivat lähtötilanteessa kaikki verkon solmut.

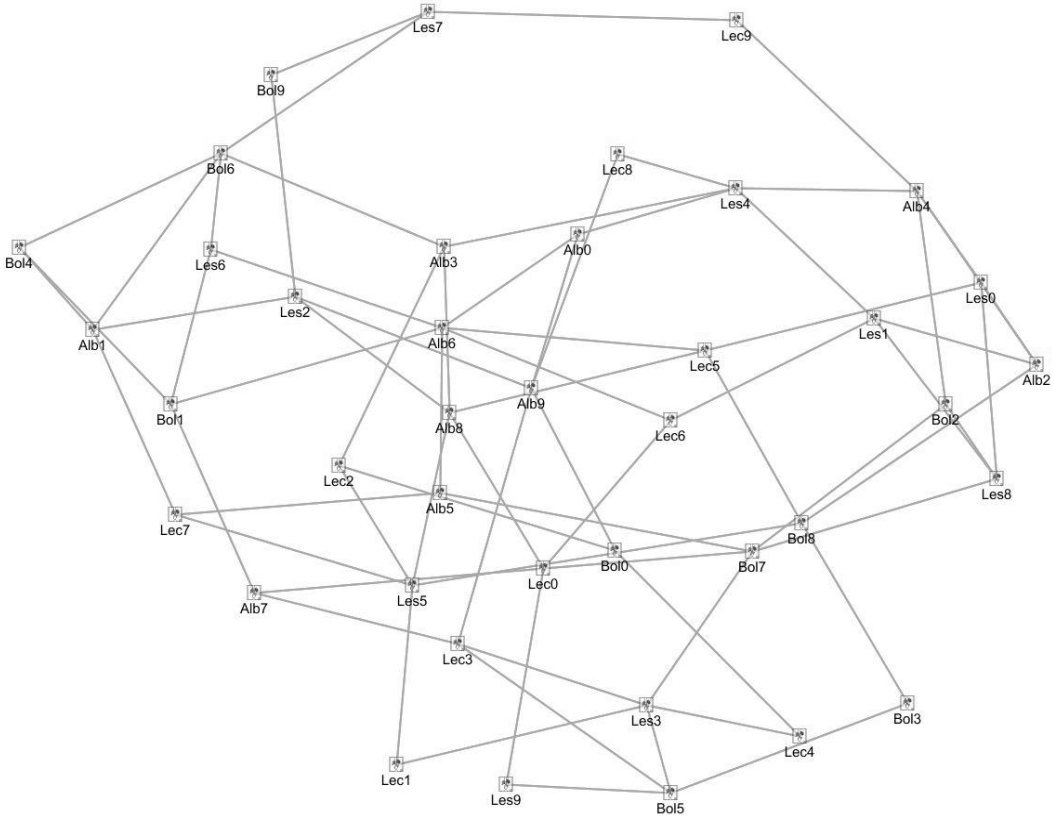
Tutkimuksessa keskityttiin arvioimaan algoritmeja sen perusteella, miten ne vaikuttavat verkon topologiaan. Tarkastelun kohteina olivat verkon konvergoituminen ja naapureiden jakaumista saadut kuvaajat. Jakaumaa verrattiin potenssijakaumaan, koska potenssijakautunut verkko on vikasietoinen. Lisäksi potenssijakaumassa solmujen etäisyydet lyhenevät, mistä johtuen resurssit löytyvät lähempää kuin normaalijakautuneessa verkossa. Jakauman lisäksi tarkasteltiin konvergoitumista. Jos verkko ei konvergoidu vaan sen topologia muuttuu jatkuvasti, yhteyksien muodostuksesta ja poistosta aiheutuva liikenne kuormittaa verkkoa. Paras topologia olisi siis potenssijakautunut verkko, joka konvergoituu.

Saatuihin naapureiden jakaumiin sovitettiin normaali- tai potenssijakaumaa, jonka poikkeama data-aineistosta on laskettu virhefunktiolla

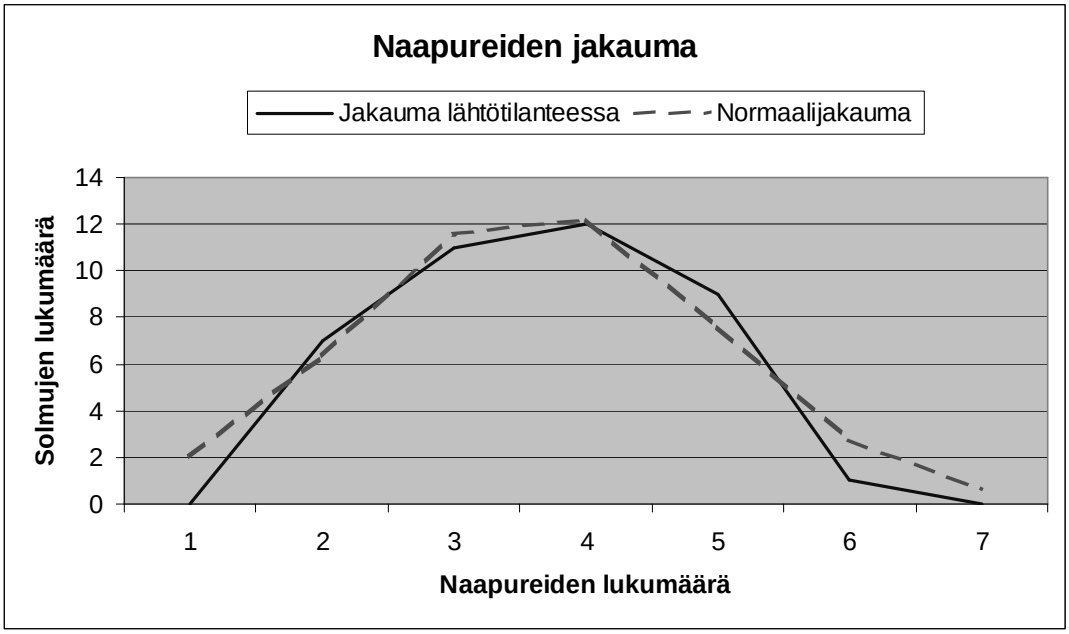
$$S = \sqrt{\frac{\sum_{i=1}^{n_p} \left(y_i - f(x_i)\right)^2}{n_p - n_{param}}}. \qquad (3)$$

Funktiossa n_p on datapisteiden lukumäärä, n_{param} on sovituksessa käytetyn mallin parametrien lukumäärä, y_i on datapiste ja $f(x_i)$ on jakaumafunktion arvo. Mitä lähempänä nollaa virhefunktion arvo on, sitä paremmin jakauma sopii data-aineistoon.

Testeissä käytettiin aina samaa lähtötopologiaa, joka on esitetty kuvassa 12. Verkko on muodostettu satunnaisesti ja sen solmujen naapureiden lukumäärää kuvaava graafi on kuvassa 13. Topologiassa suurimmalla osalla on kolme tai neljä naapuria ja suurimmalla solmulla on kuusi naapuria, kuten graafista käy ilmi. Lähtötilanteen jakaumagraafi noudattaa normaalijakaumaa virhefunktion (3) arvolla 1.7.



Kuva 12: Verkon lähtötilanne.



Kuva 13: Verkon solmujen naapureiden jakauma lähtötilanteessa.

7 Tutkimustapaukset ja tulosten arviointi

Tässä luvussa esitellään käytetyt tutkimustapaukset sekä analysoidaan niissä saadut tulokset.

7.1 Solmun ohitus

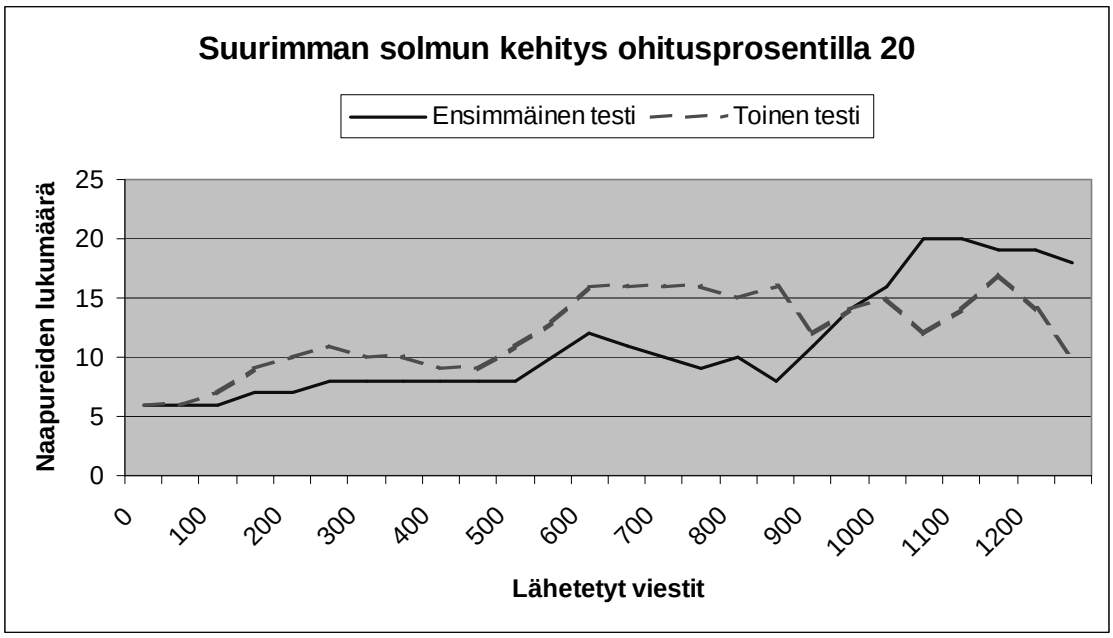
Solmun ohitus -algoritmia tarkasteltiin neljällä eri ohitusprosenttiarvolla: 20%, 40%, 60% sekä 80%.

Tutkimuksessa kaikki eri prosenttiarvot testattiin lähtötopologialla vähintään kahteen kertaan. Saatuja topologioita verrattiin keskenään sekä saman prosenttiluvun sisällä että eri prosenttien välillä. Topologioista etsittiin myös mahdollista konvergoitumista johonkin verkon tilaan.

Kaikilla eri prosenttiluvuilla samastakin lähtötilanteesta lähtevät testaukset saattoivat antaa eri lopputuloksen. Tämä johtui siitä, että koska resurssikyselyviestejä lähetettiin verkkoon viisi kerrallaan, saattoivat vastausviestit ehtiä lähettäjille eri järjestyksessä, mikä tietysti vaikutti osumiin ja siten mahdollisiin ohituksiin. Verkon keskussolmuna saattoi olla muukin kuin eniten tarjoava solmu, jos sillä vain oli ympärillään riittävästi solmuja, jotka tarjosivat tai kyselivät paljon. Tästä syystä tuloksissa seurattiin keskussolmujen lisäksi myös niiden naapureita, topologiaa kuvaavaa graafia sekä verkon konvergoitumista.

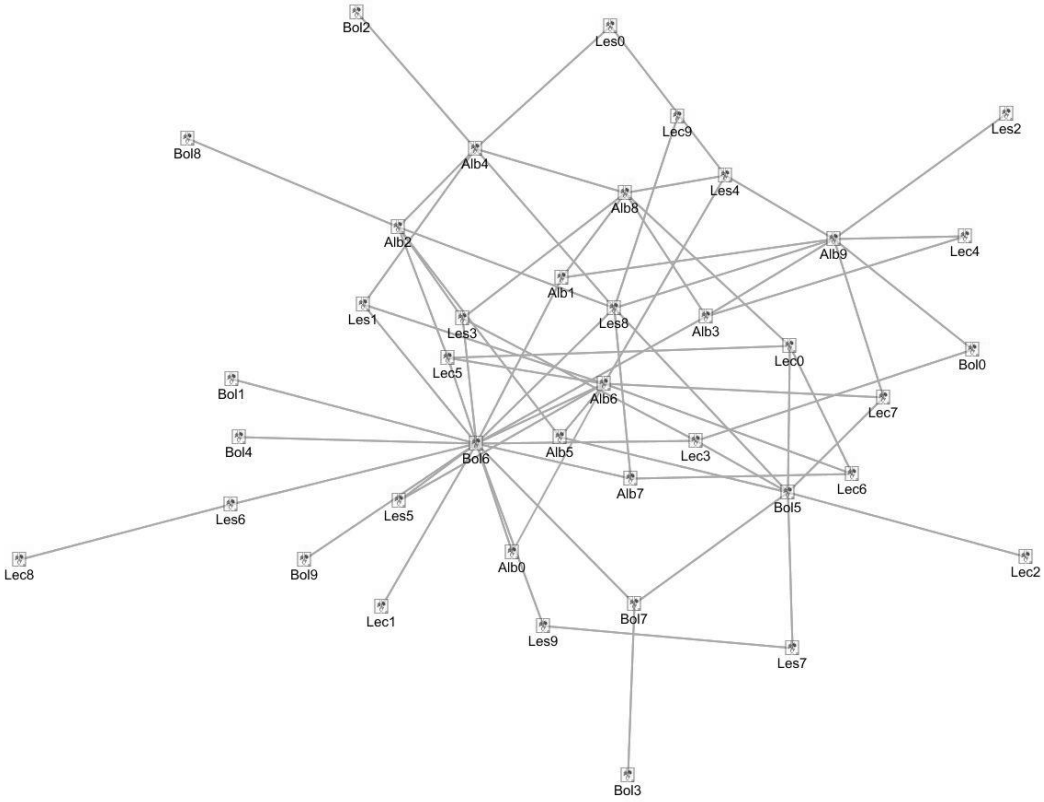
7.1.1 Ohitus 20%

Valittaessa ohitusprosentiksi kaksikymmentä, 1250 kyselyllä saadut graafit noudattelivat potenssijakautunutta graafia. Ensimmäisessä testissä saatiin aikaiseksi verkko, jossa suurimmalla keskussolmulla oli kymmenen naapuria ja toisessa testissä saadussa verkossa oli yksi huomattavasti muita solmuja suurempi keskussolmu, jolla oli 18 naapuria. Suurimman solmun naapureiden lukumäärän kehitys molemmissa testeissä käy ilmi kuvasta 14.

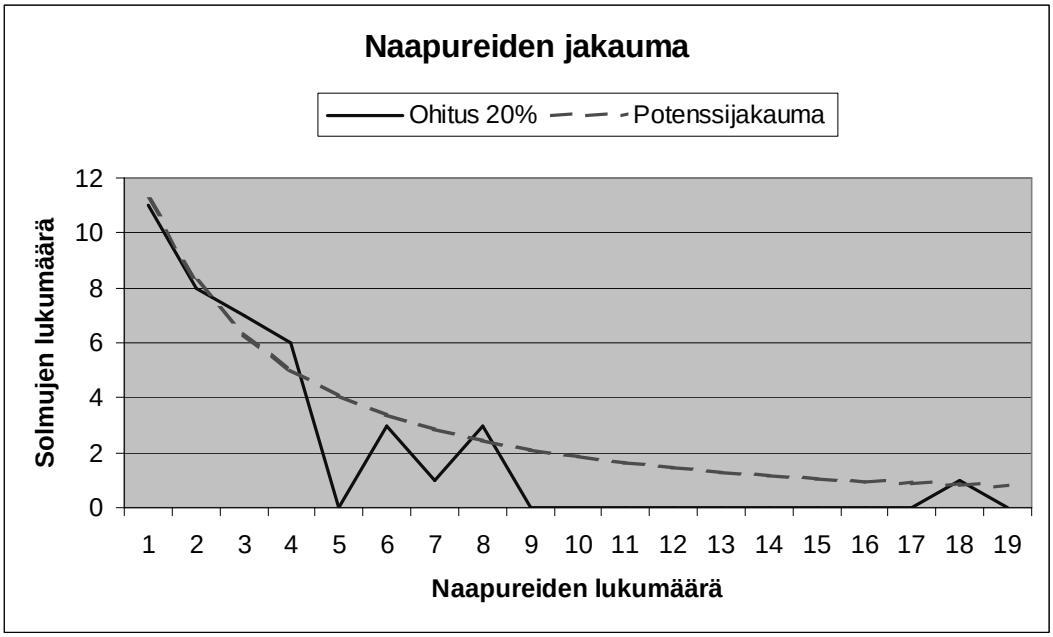


Kuva 14: Suurimman solmun naapureiden lukumäärän kehitys ohitusprosentilla 20.

Ensimmäisessä testissä noin 900 viestin kohdalla alkoi yksi solmuista keräämään naapureita huomattavasti enemmän kuin muut saavuttaen lopputilanteessa 18 naapurin lukumäärän, kun seuraavaksi suurimmilla solmuilla oli kahdeksan naapuria. Suurimmalla osalla solmuja naapureita oli yksi tai kaksi. Ensimmäisen testin lopputuloksena saatu topologia ja sen naapureiden lukumääristä piirretty graafi on esitetty kuvissa 15 ja 16. Naapureiden jakauma noudatti potenssijakaumaa virhefunktion arvolla 1.04.



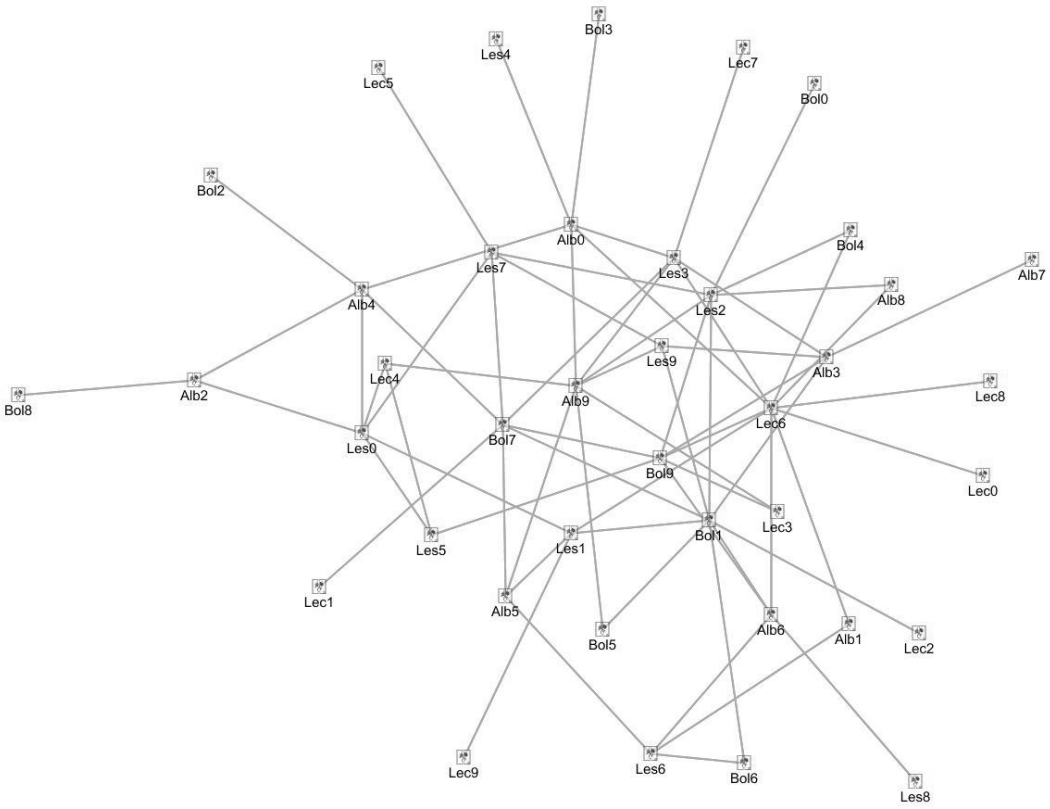
Kuva 15: Verkon topologia ensimmäisen testi jälkeen.



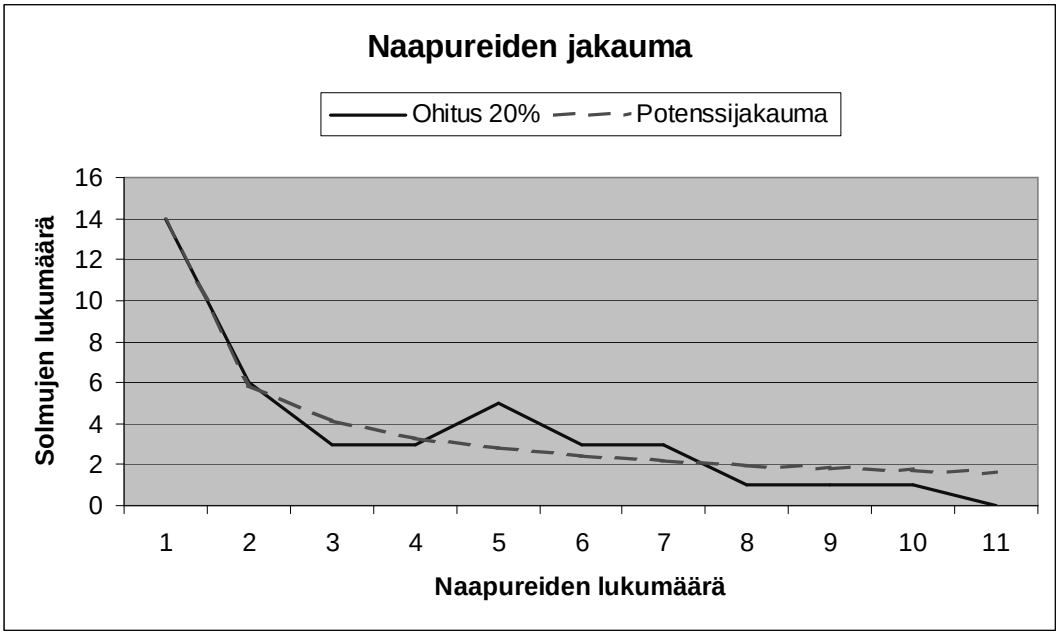
Kuva 16: Naapureiden jakauma ensimmäisen testi jälkeen.

Toisen testin lopputilanteessa mikään solmuista ei kasvanut yhtä selkeästi muita suuremmaksi kuin mitä ensimmäisessä testissä. 500 viestin jälkeen verkossa yksi solmu alkoi muodostua selkeäksi keskussolmuksi saaden suurimmillaan 16 naapuria, kun seuraavalla naapureita oli kahdeksan. 800 viestin jälkeen suurimman solmun naapureiden lukumäärä alkoi kuitenkin hiljalleen pudota ja seuraavaksi suurimman kasvoi hieman ja 1250 viestin kohdalla verkossa suurimmalla solmulla oli kymmenen naapuria ja seuraavaksi suurimmalla yhdeksän. Suurimmalla osalla solmuja naapureiden lukumäärä oli yksi.

Toisessa testissä saatu topologia ja jakauma ovat kuvissa 17 ja 18. Saatu naapureiden jakauma noudatti potenssijakaumaa hieman ensimmäistä testiä huonommin. Virhefunktion arvo oli toisessa testissä 1.16.



Kuva 17: Verkon topologia toisen testin jälkeen.



Kuva 18: Naapureiden jakauma toisen testin jälkeen.

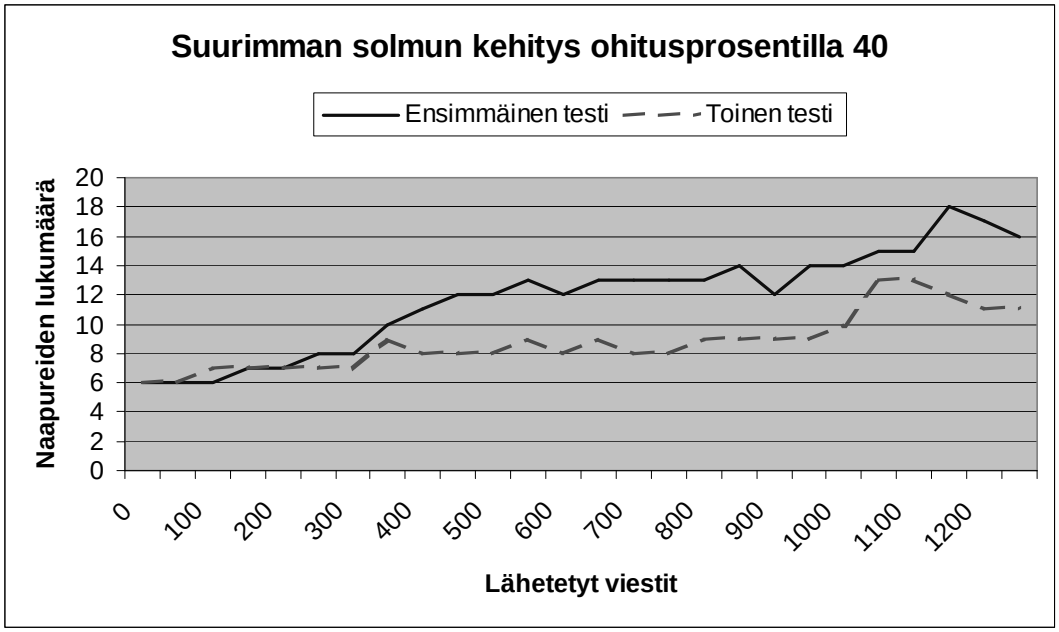
Ensimmäisessä testissä suurimpana solmuna oli paljon tarjoava solmu. Toisessa testissä kahdeksan naapuria saanut solmu oli myös paljon tarjoava. Sen sijaan solmu, jolla oli yhdeksän naapuria, oli vähän tarjoava, mutta paljon kyselevä. Myös suurin osa kaikkien näiden solmujen naapureista oli paljon kyseleviä tai paljon tarjoavia. Ensimmäisen testin keskussolmusta vain kaksi naapureista oli vähän tarjoavia ja kyseleviä. Toisessa testissä vastaava luku oli kolme, tosin näiden naapureiden toisena naapurina oli paljon tarjoava solmu, joka ei ollut keskussolmun naapuri.

Ensimmäisen ja toisen ajon eroihin on monta mahdollista syytä, kuten resurssiviestien erilaisen saapumisen ja ohituksen tarkastamisen sekä pienen ohitusprosentin yhteisvaikutus. Solmu lähtee herkästi liikkeelle ja jo muutama erilainen viestijärjestys saattaa siirtää solmun eri testeissä hyvin eri paikkaan verkossa, jolloin lopputuloksena saatavat verkot poikkeavat toisistaan melkoisesti solmujen paikkojen suhteen.

Kahdellakymmenellä prosentilla ei konvergoitumista tapahdu, koska solmu lähtee herkästi liikkeelle. Tästä syystä 1250 viestin jälkeen ei pystytä sanomaan, muodostuisiko ensimmäisen ja toisen testin verkoista lopulta samankaltaiset, jos vain viestejä lähetettäisiin tarpeeksi. Vaikka muodostuva jakauma onkin potenssijakautunut ja sinänsä vikasietoinen topologia, se ei kuitenkaan ole paras, koska se ei konvergoidu. Tarkoituksena on, että Chedarin solmut liikkuvat kohti hyviä solmuja ilman, että menetetään muita hyviä naapureita, joten kaksikymmentä prosenttia on liian pieni ohitusarvo. Liiallisesta ohituksesta koituu haittaa myös vastausviestien reititykseen, jos naapurit vaihtuvat tiheään. Lisäksi ohitukset lisäävät turhaan verkkoliikennettä yhteyden muodostukseen sekä naapureiden ja palveluiden kyselemiseen tarvittavien viestien takia.

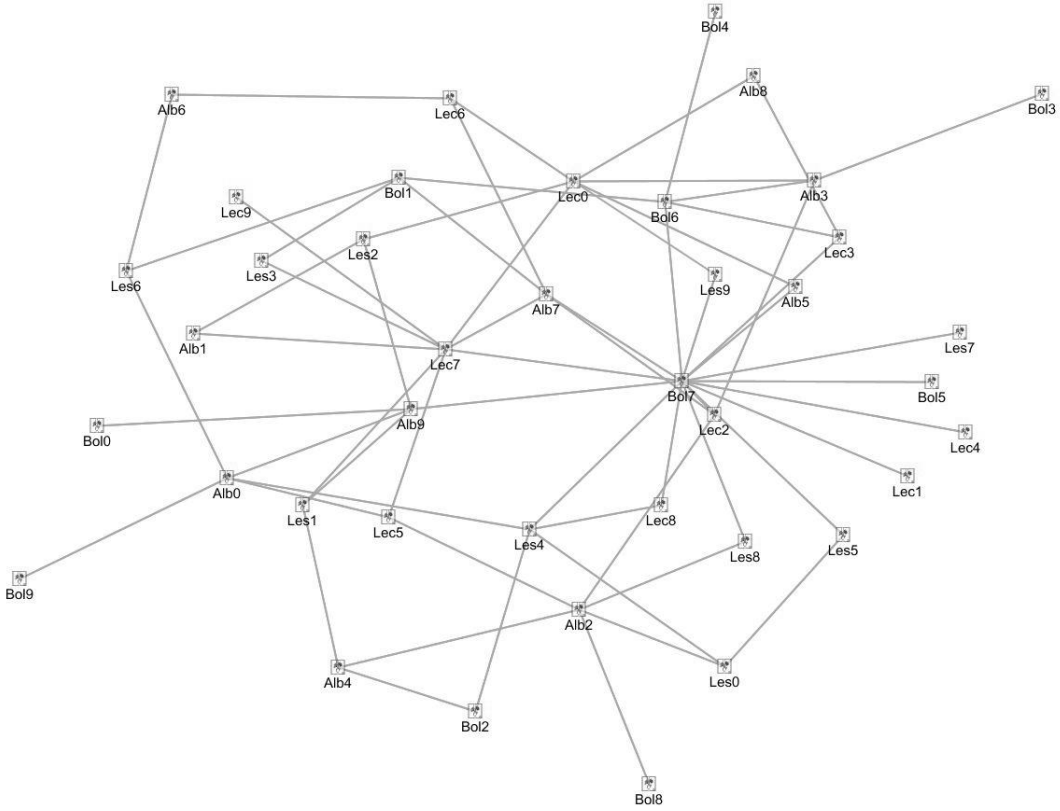
7.1.2 Ohitus 40%

Myös neljälläkymmenellä ohitusprosentilla oli lopputuloksissa havaittavissa merkkejä potenssijakautuneista graafeista. Molemmissa testeissä suurimmalla osalla solmuja oli jälleen vain yksi tai kaksi naapuria. Ensimmäisessä testissä suurimmalla solmulla oli 16 naapuria, kun taas toisessa testissä suurimmat naapurit olivat yhteydessä 9-11 muuhun solmuun. Kuvassa 19 on esitetty suurimman solmun naapureiden lukumäärän kehitys testien edetessä.

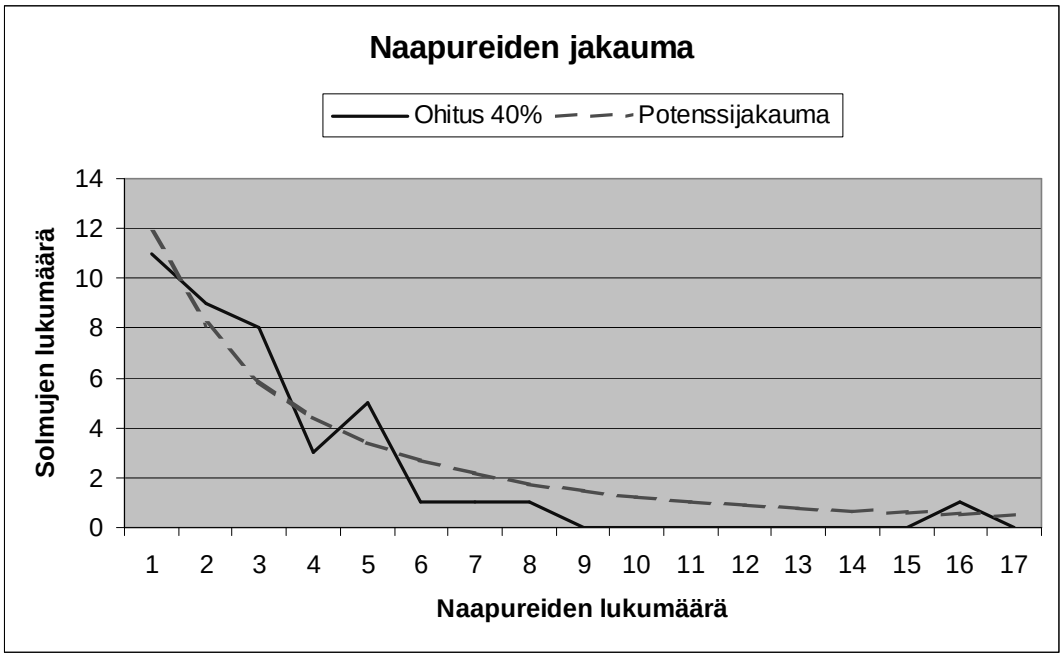


Kuva 19: Suurimman solmun naapureiden lukumäärän kehitys ohitusprosentilla 40.

Ensimmäisessä testissä keskussolmu alkoi hiljalleen erottautua muista solmuista 400 viestin jälkeen. Suurimmillaan solmu oli 1150 viestin kohdalla, jolloin sillä oli 18 naapuria ja seuraavaksi suurimmalla seitsemän. Viimeisten sadan viestin aikana erot hieman pienenivät ja viimeisten viestien jälkeen keskussolmulla oli 16 naapuria ja seuraavaksi suurimmalla kahdeksan, kun suurimmalla osalla oli yksi tai kaksi naapuria. Naapurijakauma erosi siihen sovitetusta potenssijakaumasta virhefunktion arvolla 1.60. Ensimmäisessä testissä saatu topologia on kuvassa 20 ja jakauma kuvassa 21.

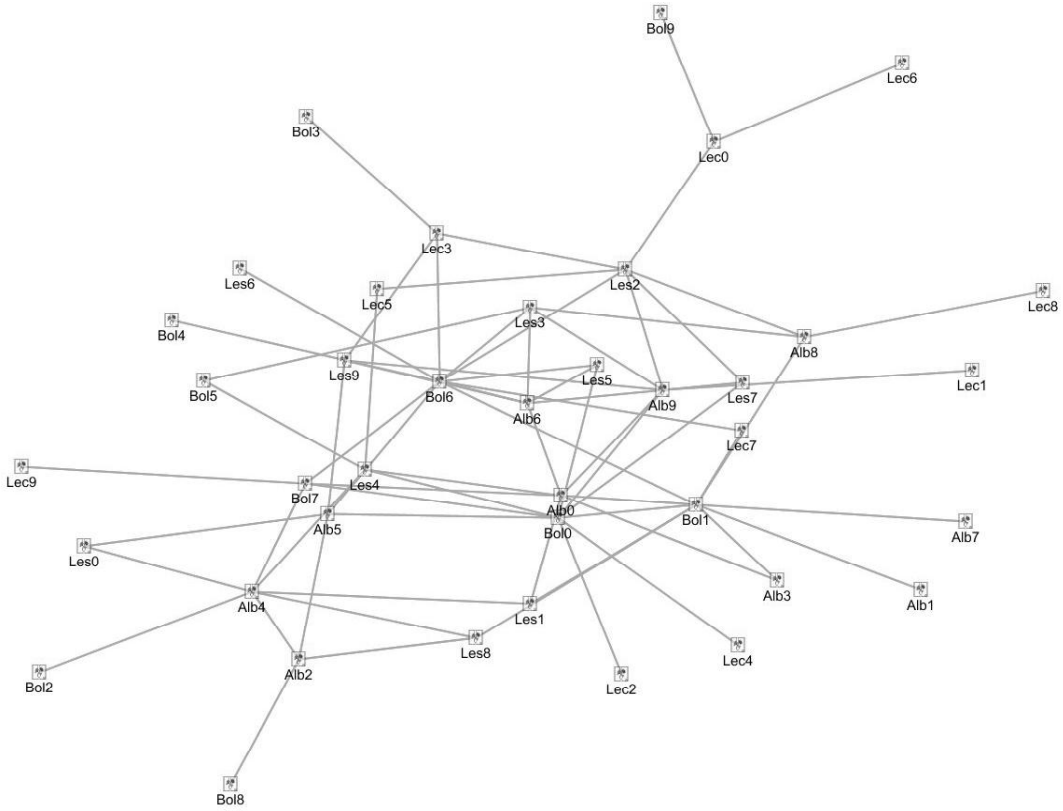


Kuva 20: Kuvassa on ensimmäisessä testissä saatu topologia 1250 viestin jälkeen.

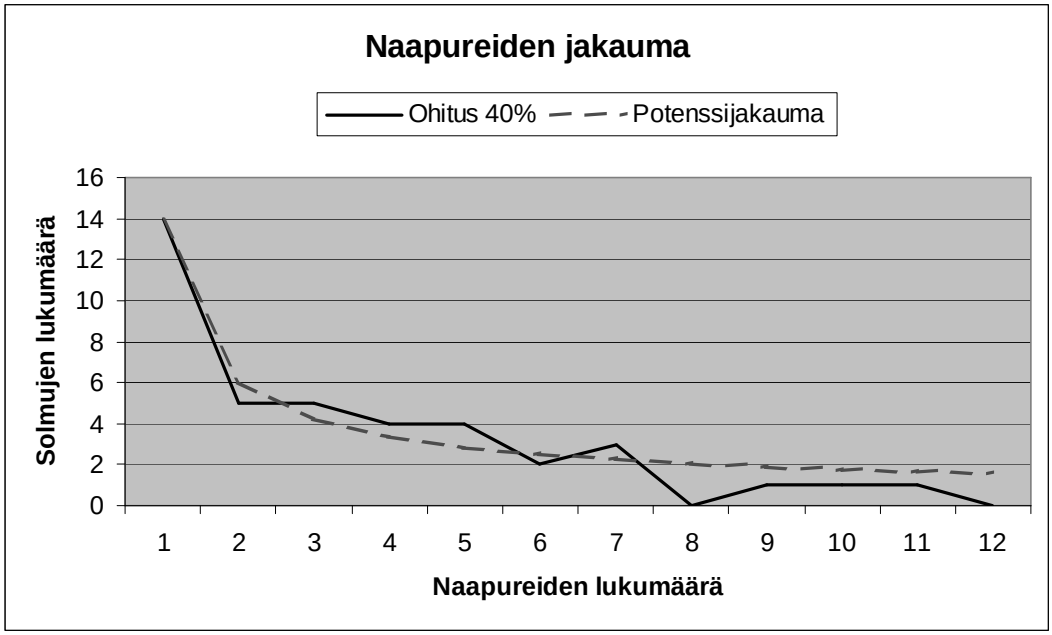


Kuva 21: Solmujen naapureiden jakauma ensimmäisen testin jälkeen.

Toisessa testissä ei yksikään solmu muodostunut yhtä suureksi keskussolmuksi kuin ensimmäisessä testissä, vaikka jakauma noudattelikin potenssijakaumaa virhefunktion arvolla 0.92. Verkon suurimmalla solmulla oli 11 naapuria, mutta seuraavaksi suurimmalla oli vain yksi vähemmän eli kymmenen. Saatu topologia on esitetty kuvassa 22. Naapureiden lukumäärästä kertova jakauma muodostui kuvan 23 kaltaiseksi.



Kuva 22: 40 prosentilla toisessa testissä saatu topologia.

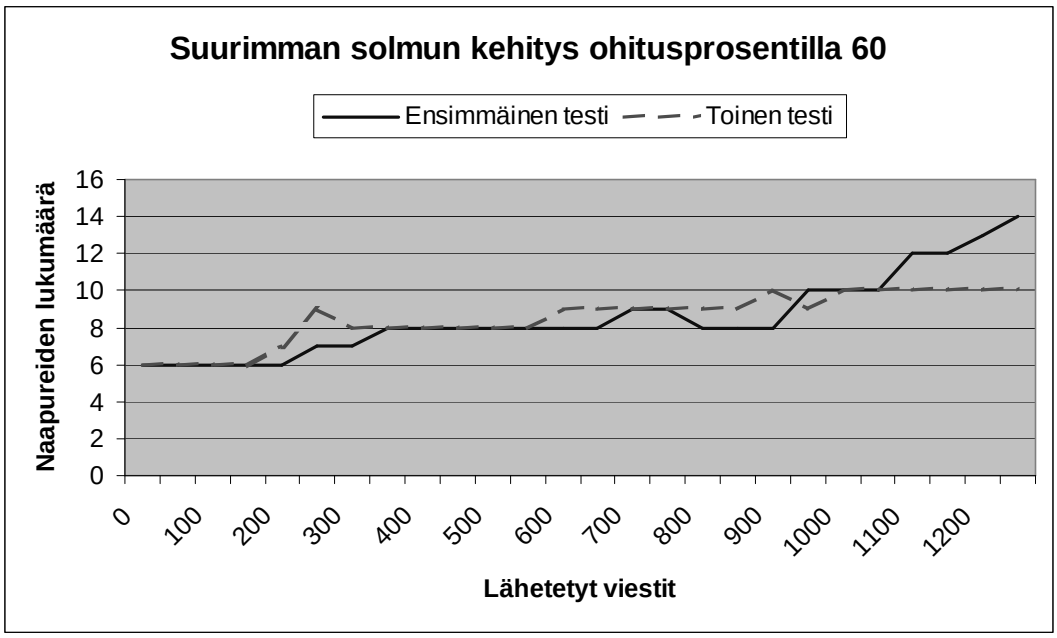


Kuva 23: Naapureiden jakauma toisessa testissä.

Kummassakaan testissä verkko ei konvergoitunut 1250 viestin aikana, vaikka suuria muutoksia topologiassa ei viimeisten viestien aikana tapahtunutkaan. Verkot olivat topologioiltaan erilaisia kuten kahdellakymmenellä prosentillakin. Keskussolmujen ympärillä olevat solmut olivat kuitenkin kummassakin verkossa osittain samoja, joten neljälläkymmenellä prosentilla saatiin kuitenkin hieman vähemmän ailahtelevainen verkko. Lisäksi kummassakin verkossa suurimpina keskussolmuina oli paljon resursseja tarjoavia solmuja.

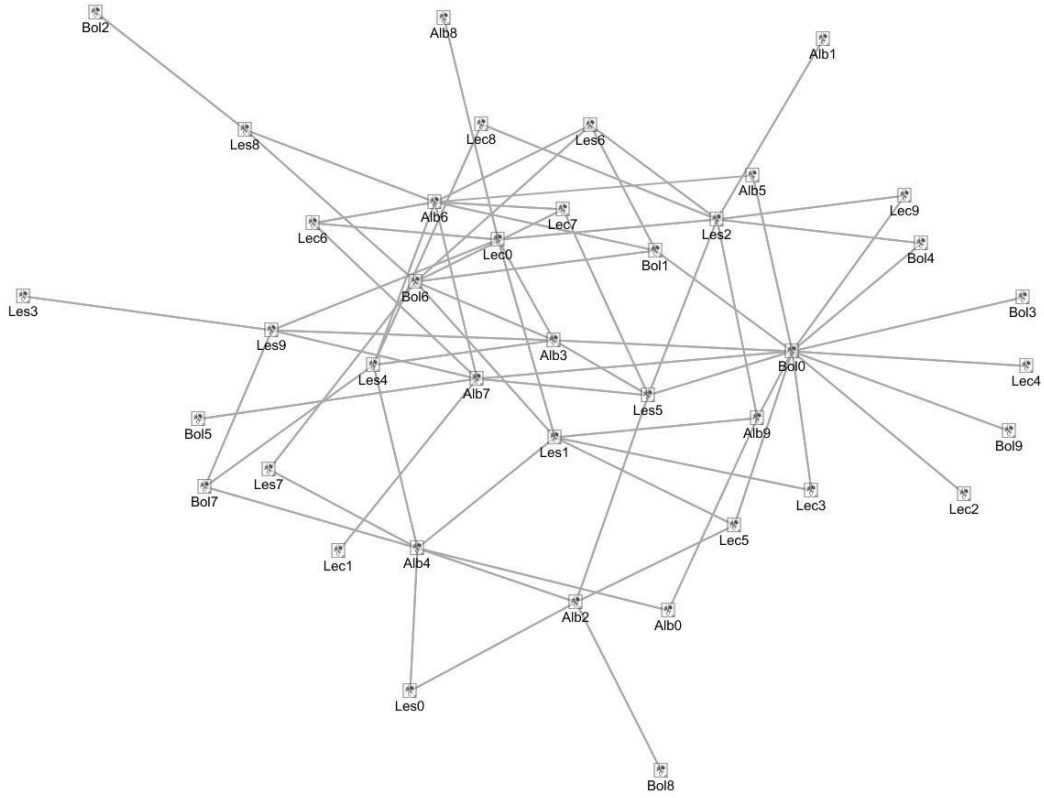
7.1.3 Ohitus 60%

Myöskään kuudellakymmenellä prosentilla saadut verkot eivät konvergoituneet täysin 1250 viestin aikana. Naapureiden jakaumat noudattivat edelleen potenssijakaumaa. Kuvasta 24 käy ilmi suurimman solmun keräämien naapureiden lukumäärän kehitys ensimmäisessä ja toisessa testissä.

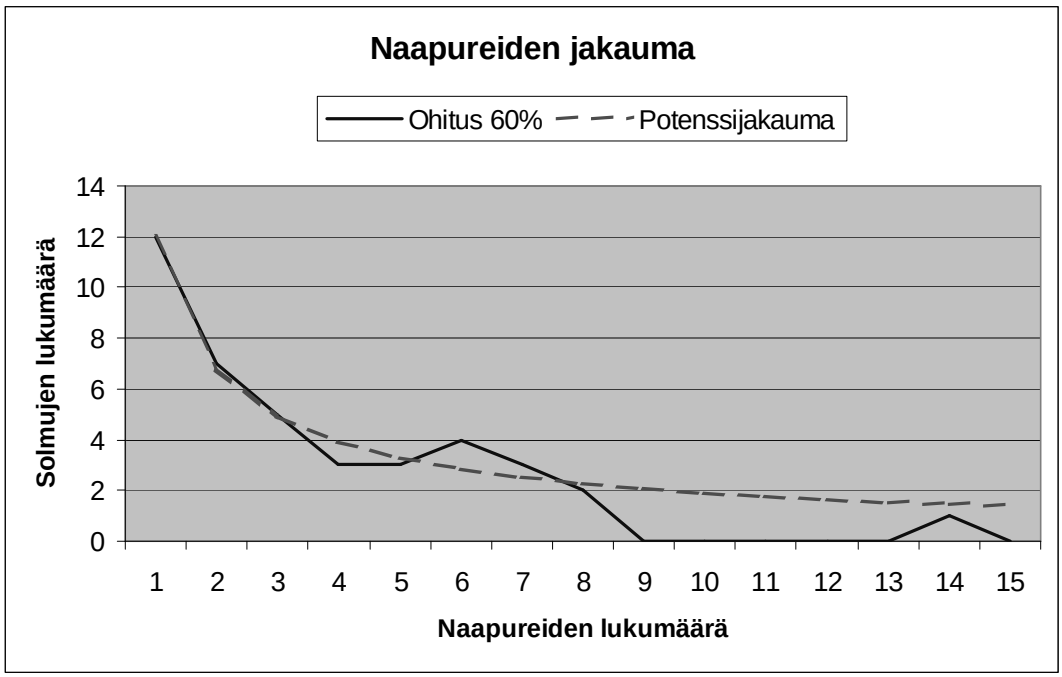


Kuva 24 Suurimman solmun naapureiden lukumäärän kehitys ohitusprosentilla 60.

Ensimmäisessä testissä oli 1000 viestin jälkeen graafissa havaittavissa keskussolmun erkaantumista ja graafi alkoi enemmän muodostua potenssijakauman mukaiseksi. Viimeisten viestien aikana topologia muokkaantui tilaan, jossa verkossa oli yksi suuri keskussolmu, jolla oli 14 naapuria, kun seuraavaksi suurimmat olivat kahdeksan naapurin ympäröimiä. Suurimmalla osalla solmuja oli yksi naapuri. Saatu jakauma noudatti potenssijakaumaa virhefunktion arvolla 0.69. Kuvassa 25 on ensimmäisessä testissä saatu topologia. Siitä piirretty naapureiden jakauma on kuvassa 26.

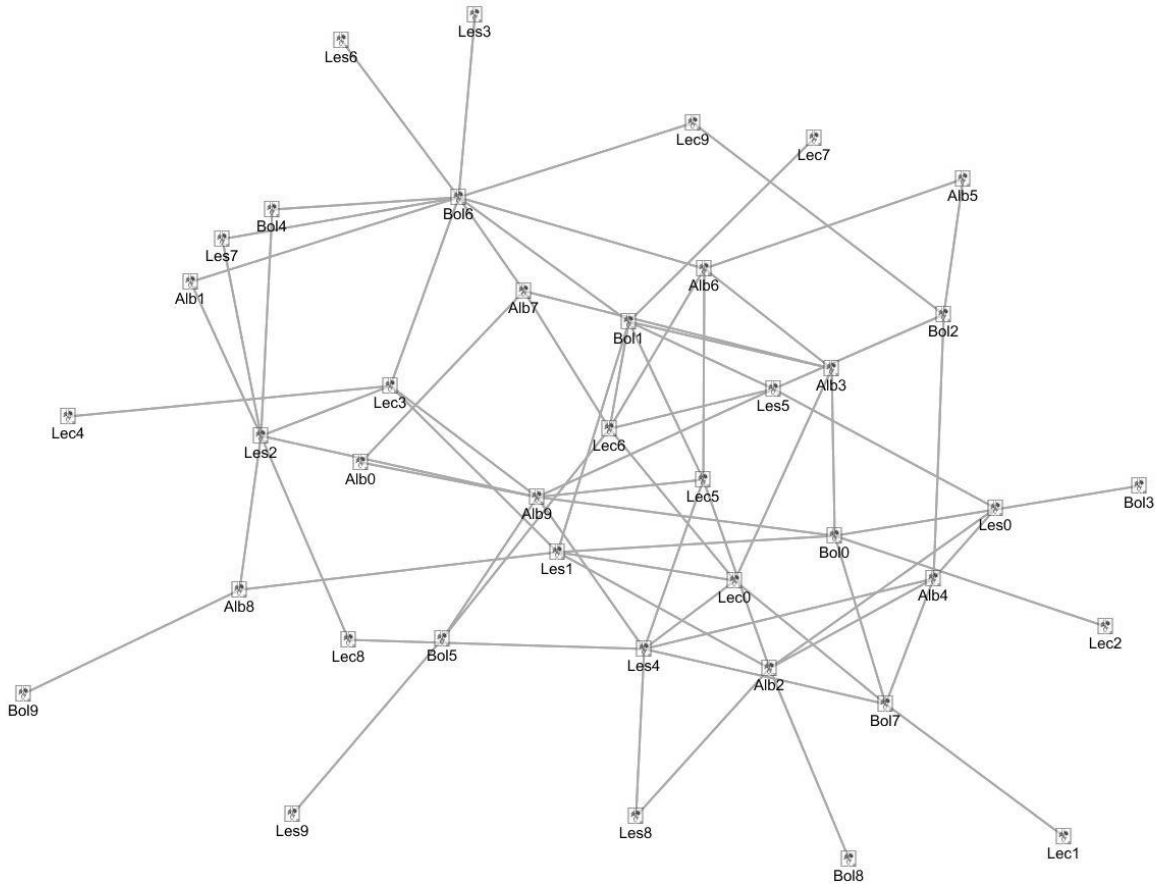


Kuva 25: Verkon topologia ensimmäisen testin jälkeen.

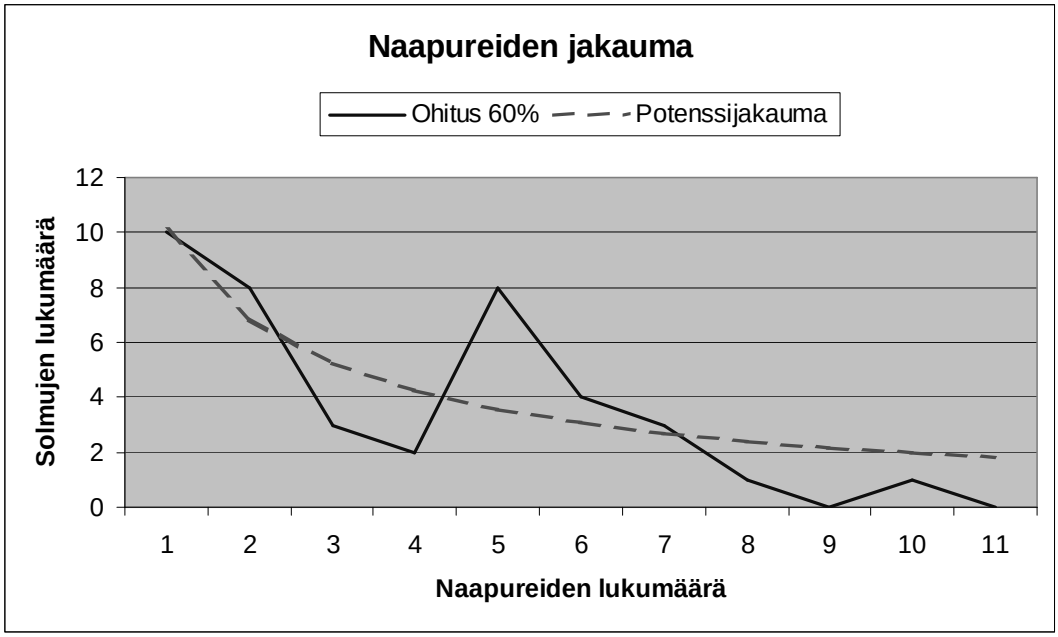


Kuva 26: Solmujen naapureiden jakauma ensimmäisen testin jälkeen.

Toisessa testissä saatu verkko poikkesi jo enemmän potenssijakaumasta. Virhefunktion arvoksi saatiin 2.41. Myöskään yhtä selkeää keskussolmua ei muodostunut, vaan naapurit olivat tasaisemmin jakautuneet. Suurimmalla solmulla oli testin lopussa kymmenen naapuria ja seuraavaksi suurimmalla kahdeksan. 600 viestin jälkeen verkko alkoi muodostua lopputilanteessa olevaan muotoon ja suurin solmu piti paikkansa 800 viestistä lähtien. 45%:lla solmuista oli naapureita yksi tai kaksi.



Kuva 27: Verkon topologia toisen testin jälkeen.



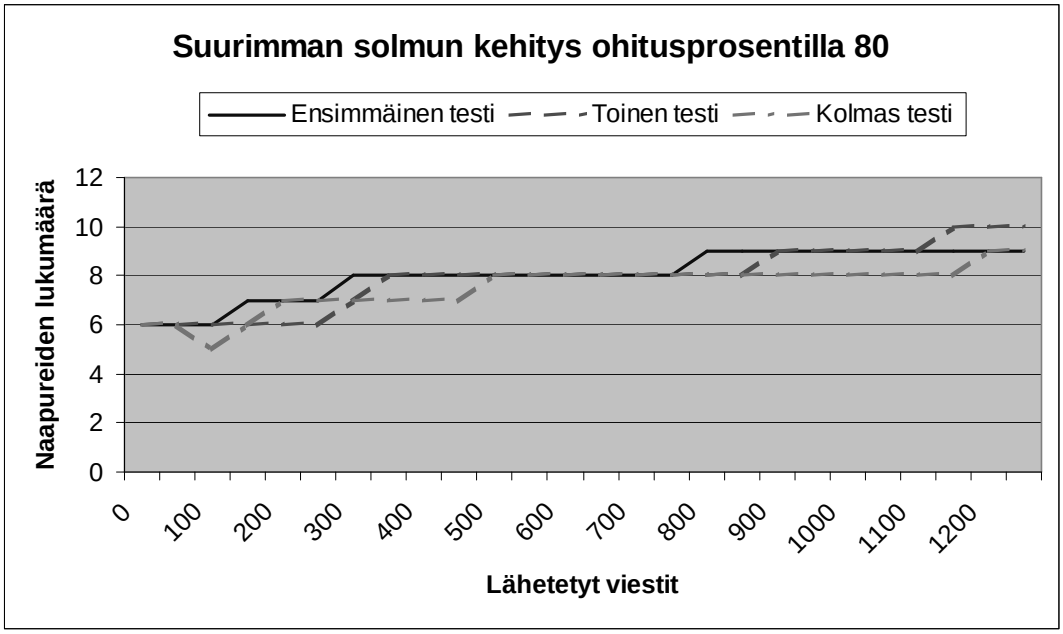
Kuva 28: Solmujen naapureiden jakauma toisen testin jälkeen

Ensimmäisessä testissä keskussolmuksi tuli solmu, joka tarjoaa sekä kyselee vähän. Tarkasteltaessa keskussolmun naapureita voitiin kuitenkin havaita, että niiden joukossa oli monta paljon tarjoavaa sekä paljon kyselevää. Myös muut paljon tarjoavat saivat ympärilleen yli kolme naapuria.

Toisessa testissä keskussolmuksi ja muiksikin suurimmiksi solmuiksi tulivat solmut, jotka tarjoavat paljon resursseja. Myös suurimman solmun ympärillä olevat solmut olivat solmuja, jotka kyselevät ja tarjoavat paljon.

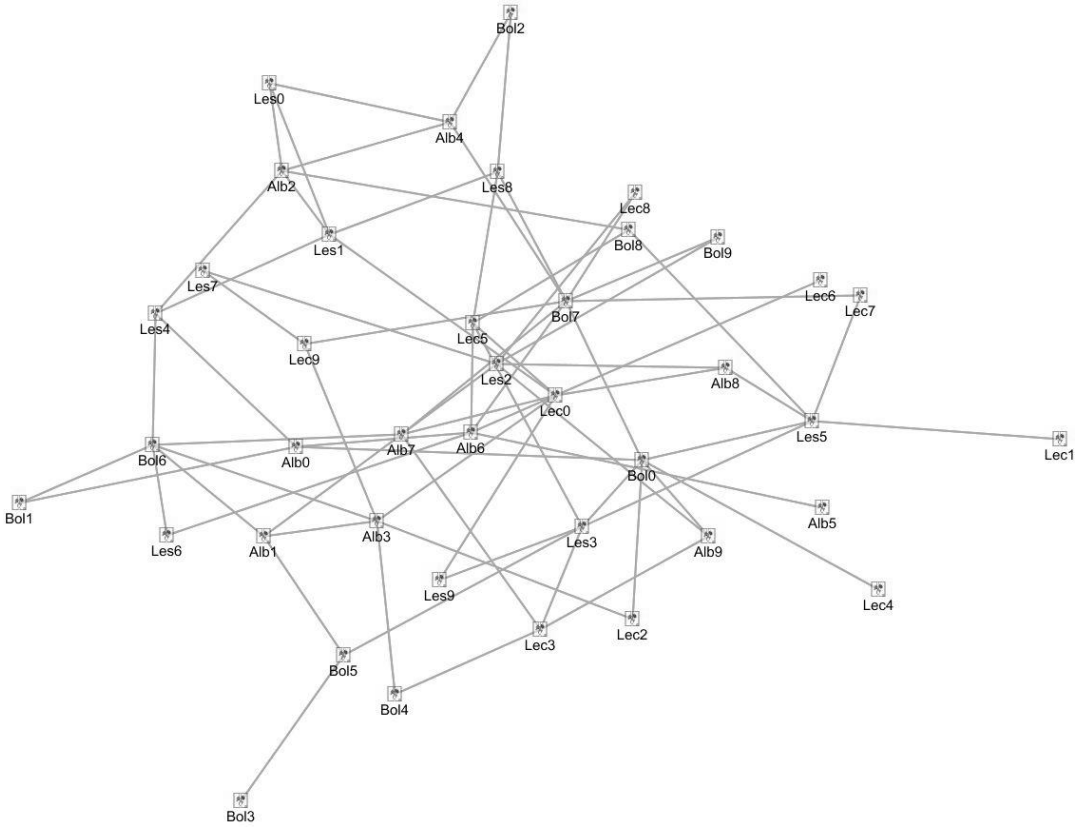
7.1.4 Ohitus 80%

Kahdeksallakymmenellä prosentilla testaus suoritettiin kolme kertaa. Saaduissa topologiagraafeissa suurimmalla osalla solmuja oli vain vähän yhteyksiä, yksi tai kaksi, ja muutamalla solmulla oli paljon naapureita, vaikka graafit eivät suoraan noudattaneetkaan potenssijakaumaa. Myös suurimpien solmujen naapureiden lukumäärä oli yhdeksän tai kymmenen, vaikka saatu keskussolmu vaihtelikin. Verkon suurimman solmun naapureiden lukumäärän kehitys kolmessa eri testissä on kuvassa 29.

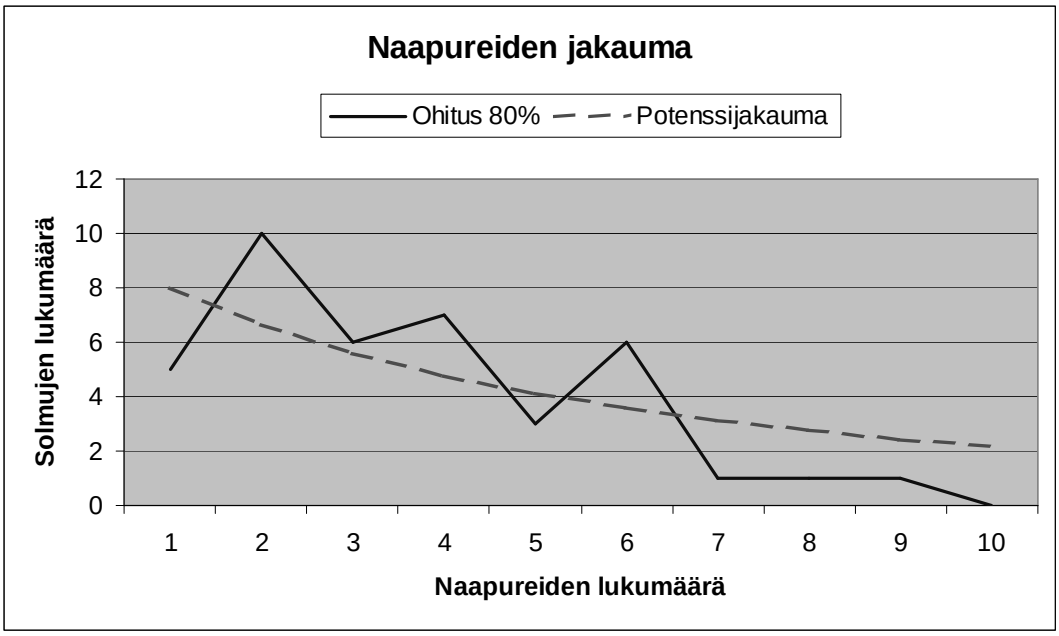


Kuva 29: Suurimman solmun naapureiden lukumäärän kehitys ohitusprosentilla 80.

Ensimmäisessä testissä saatiin lopputuloksena kuvan 30 kaltainen topologia. Suurimmalla solmulla oli yhdeksän naapuria ja eniten verkossa oli kahden naapurin solmuja. Saatu jakauma poikkesi eniten potenssijakaumasta. Jakauman ja siihen sovitetun potenssijakauman virhearvoksi saatiin 2.65. Topologia konvergoitui täydellisesti eikä siinä tapahtunut mitään muutoksia 850 viestin jälkeen. Topologiasta tehty naapureiden jakauma on kuvassa 31.

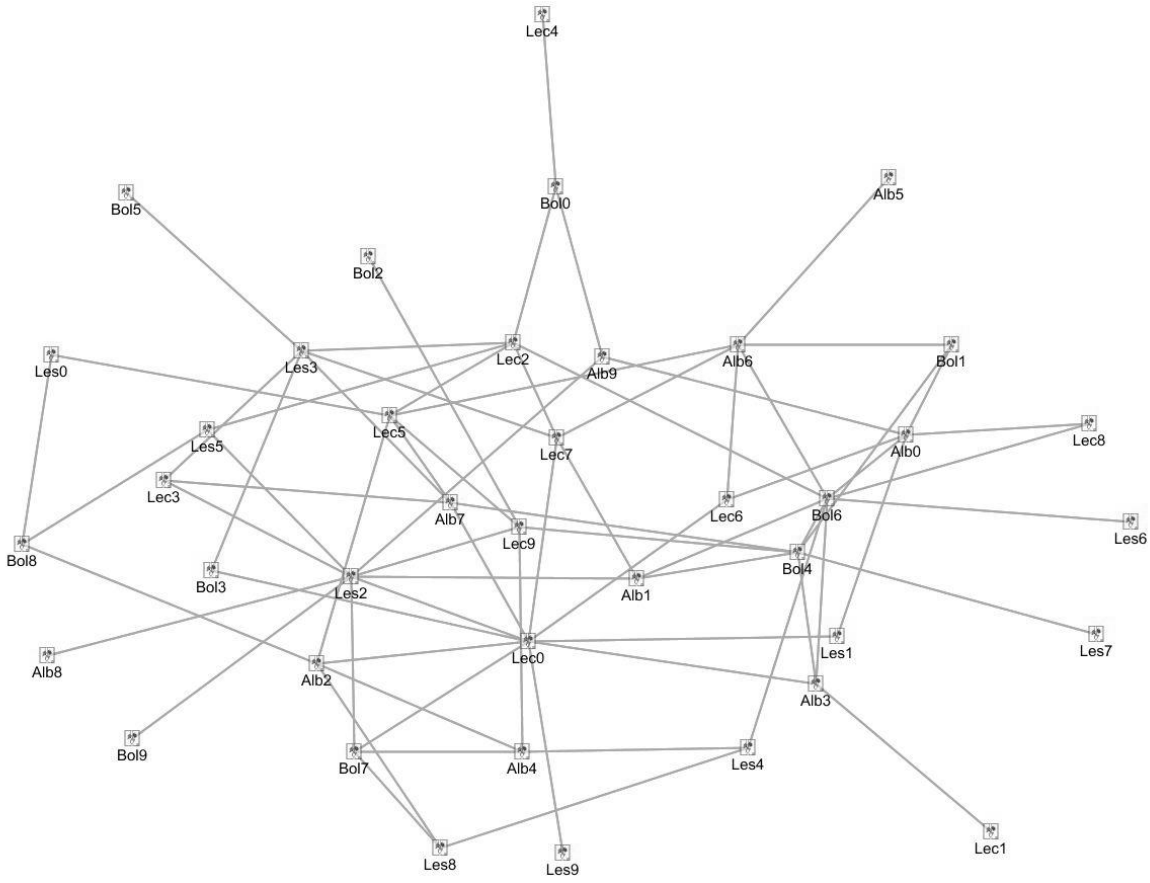


Kuva 30: Verkon topologia ensimmäisen testin jälkeen.

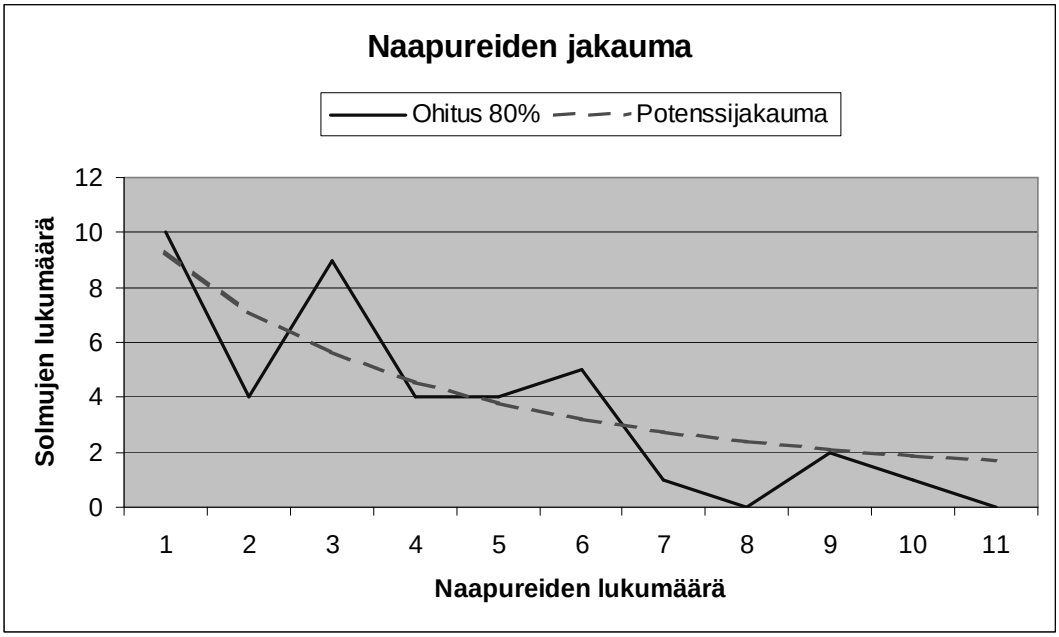


Kuva 31: Solmujen naapureiden jakauma ensimmäisen testin jälkeen.

Toisessa testissä saatu verkko ei konvergoitunut yhtä aikaisessa vaiheessa, vaan vasta 1100 viestin jälkeen. Tosin tätäkin ennen muutokset edellisten sadan viestin aikana olivat verkossa hyvin pieniä. Lukumäärällisesti eniten verkossa oli yhden naapurin solmuja, vaikka kolmen naapurin solmuja oli lähes yhtä paljon. Suurimmat solmut verkossa olivat kymmenen naapurin solmu sekä kaksi solmua, joilla oli yhdeksän naapuria. Naapureiden jakauma erosi siihen sovitetusta potenssijakaumasta virhearvolla 2.20. Testin tuloksena saatu topologia on kuvassa 32 ja siihen liittyvä naapureiden jakauma kuvassa 33.

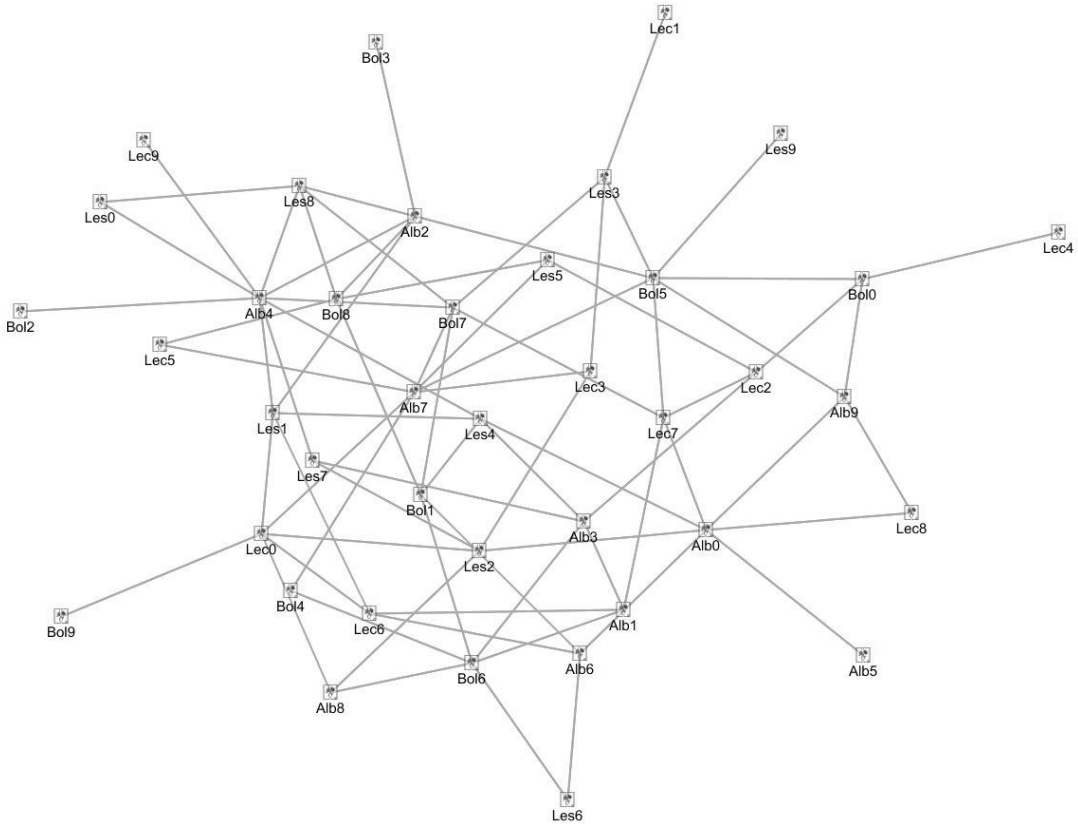


Kuva 32: Verkon topologia toisen testin jälkeen.

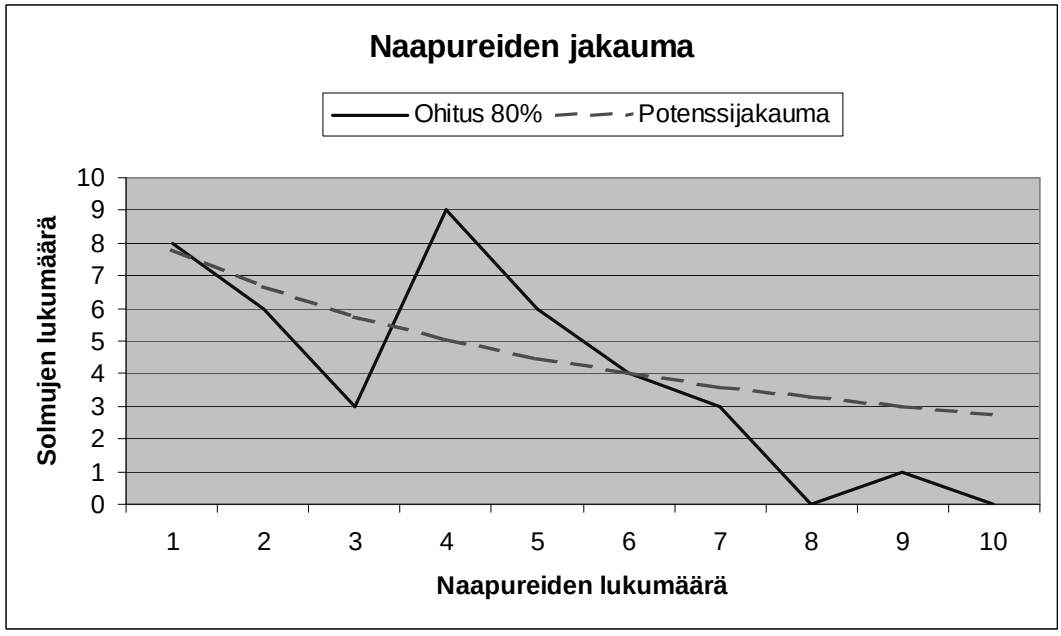


Kuva 33: Solmujen naapureiden jakauma toisen testin jälkeen.

Myös kolmas saatu verkko erosi potenssijakautuneesta verkosta. Naapureiden jakaumasta saatu virhefunktion arvo oli 2.46. Verkon suurimmalla solmulla oli yhdeksän naapuria. Sen sijaan eniten verkossa oli solmuja, joilla oli neljä naapuria, vaikka yhden naapurin solmuja olikin vain yhtä vähemmän. Verkko myös konvergoitui vasta myöhemmin kuin muissa testeissä saadut verkot niin, että 1200 viestin kohdalla piirretty jakauma ei eronnut lopputuloksesta. Kuvassa 34 on testissä saatu topologia ja kuvassa 35 vastaava naapureiden jakauma.



Kuva 34: Verkon topologia kolmannen testin jälkeen.



Kuva 35: Solmujen naapureiden jakauma kolmannen testin jälkeen.

Saaduissa verkoissa paljon tarjoavat solmut olivat verkon keskellä. Ensimmäisessä testissä verkon suurin keskussolmu oli paljon tarjoava solmu, sen sijaan toinen ja kolmas suuri solmu oli vähän tarjoava. Vähän tarjoavien solmujen naapurit tosin olivat kaikki joko paljon tarjoavia tai kyseleviä.

Toisessa testissä vain kaksi paljon tarjoavaa solmua oli kauempana kuin verkon suurimpien keskussolmujen naapureina. Suurin keskussolmu oli vähän tarjoava, mutta seuraavat kaksi solmua yhdeksällä naapurillaan olivat molemmat paljon resursseja tarjoavia.

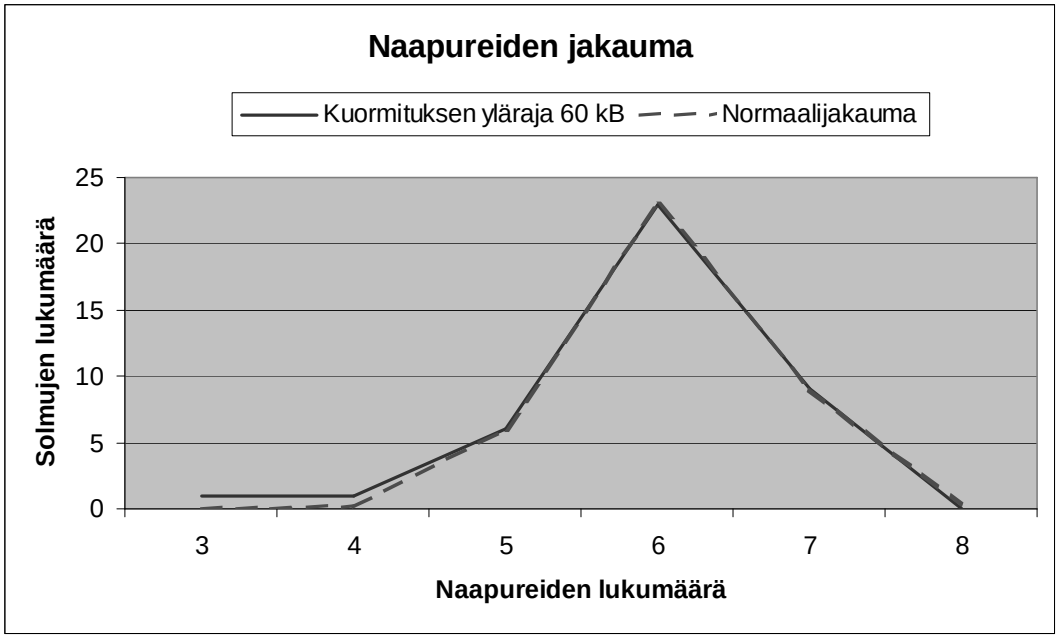
80 prosentilla saatiin topologialtaan paras verkko. Vaikkei 80 prosentin ohitus noudattanutkaan parhaiten potenssijakaumaa verrattuna muihin ohitusprosentteihin, siinä oli kuitenkin potenssijakauman merkkejä. Verkko on tällöin vikasietoisempi kuin satunnaisesti muodostettu verkko ja keskussolmut ovat kuitenkin verkon muiden solmujen kannalta hyviä solmuja. Tämä yhdistettynä täydelliseen konvergoitumiseen aiheutti sen, että 80 prosenttia valittiin parhaimmaksi ohitusprosentiksi. Tällöin verkossa vältetään turhaa liikkumista ja siirrytään vain siinä tapauksessa, että solmu, johon siirrytään, on todellakin parempi kuin muut ympärillä olevat naapurisolmut.

7.2 Kuormituksen arviointi

Kuormituksen testauksessa seurattiin Solmun valinta ja Solmun poisto -algoritmien toimivuutta. Tutkimuksessa ajan tilalle mittayksikkönä vaihdettiin solmun kautta kulkeneiden resurssikyselyiden lukumäärä, jota tarkoitetaan myöhemmin puhuttaessa aikavälistä. Kuormitustilanne tarkastettiin jokaisen viidenkymmenen lähetetyn viestin jälkeen. Koska testikoneet olivat muussakin kuin vain tämän tutkimuksen käytössä, niissä esiintyi paikoin huomattavaakin hitautta ja viestien käsittelyyn tarvittava aika vaihteli. Lisäksi P2PStudio saattoi kadottaa monitoroitavia solmuja, mistä johtuen viestit jäivät lähettämättä verkkoon. Tämän havaitseminen ja lähettämättä jääneiden viestien uudelleenlähettäminen vei aikaa. Vaihtamalla aikaväliksi viestien lukumäärän päästiin eroon näiden kahden ongelman vaikutuksesta testituloksiin ja saatuja tuloksia voitiin verrata paremmin toisiinsa, koska kaikissa testeissä lähetetyt viestit olivat aina samoja tarkasteltaessa kuormitustilannetta.

Kuormituksen ylärajoina testeissä käytettiin 20, 25, 30, 35, 40, 45, 50, 55, 60 ja 65 kilotavua. Vastaavat alarajat olivat 16, 20, 24, 28, 32, 36, 40, 44, 48, ja 52 kilotavua. Kaikille solmuille asetettiin sama raja.

Tuloksena saatujen verkkojen naapureiden jakaumat noudattivat odotetusti normaalijakaumaa huipun ollessa pienillä raja-arvoilla kolme ja suurimmilla seitsemän. Kuvassa 36 on esitettynä naapureiden jakauma, kun liikenteen ylärajaksi oli määritelty 60 kilotavua. Tällöin verrattaessa jakauman eroa normaalijakaumaan saatiin virhefunktion arvoksi 0.92.

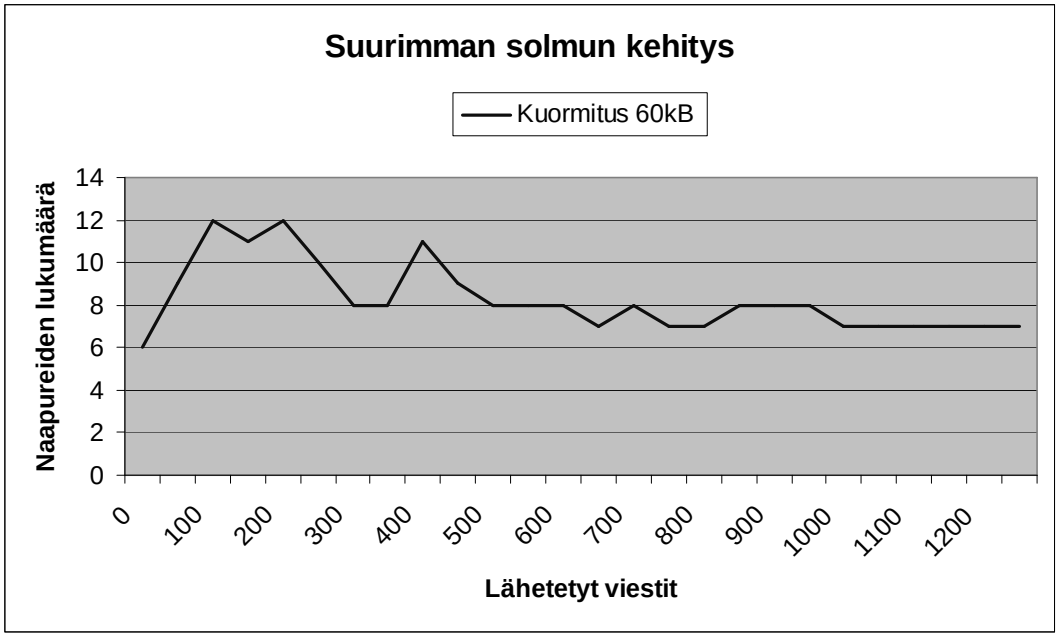


Kuva 36: Naapureiden jakauma, kun liikenteen ylärajana oli 60 kilotavua.

Normaalijakauma selittyy sillä, että kaikilla solmuilla käytetty liikenteen raja oli sama. Verkon pienuudesta ja käytetystä yleislähettävästä hakualgoritmista johtuen liikenteen määrä kasvoi verkossa tasaisesti.

Tarkasteltaessa kahta paljon tarjoavaa ja kahta vähän tarjoavaa verkon solmua oli havaittavissa, että tutkimuksessa algoritmi suosii olemassa olevia yhteyksiä. Tähän vaikuttaa tietenkin aikaväli, jolla kuormitusta testataan. Lyhyellä aikavälillä uuden solmun pitää olla todella hyvä ja tarjota tai välittää solmulle paljon resurssivastauksia, jotta sitä ei pudotettaisi solmun ylikuormittuessa. Pidemmällä aikavälillä uudella solmulla on enemmän aikaa parantaa omaa hyvyyttään verrattuna vanhojen solmujen hyvyyksiin.

Kuvassa 37 on esitetty verkon suurimman solmun naapureiden lukumäärän kehitys, kun kuormituksen arvioinnissa liikenteen ylärajaksi oli määritelty 60 kilotavua. Kuvasta voidaan havaita, että 500 lähetetyn viestin jälkeen suurimman solmun naapureiden lukumäärä vaihteli enää vain yhdellä ja 1000 viestin jälkeen suurimman solmun naapureiden lukumäärä pysyi seitsemässä.



Kuva 37: Suurimman solmun naapureiden lukumäärän kehitys, kun kuormituksen raja-arvona oli 60 kilotavua.

Solmujen naapureita seurattaessa voitiin havaita, että lähtötilanteessa solmun naapureina olevista solmuista oli lopputilanteessa solmun naapureina 40 kilotavun raja-arvolla keskimäärin kaksi ja 60 kilotavun raja-arvolla jopa kaikki. Tämä päti sekä paljon tarjoavilla ja kyselevillä että vähän tarjoavilla ja kyselevillä solmuilla. Suuremmilla raja-arvoilla solmun liikenne jää alkutilanteessa annettujen raja-arvojen väliin, jolloin pudottamista ei tarvitse tehdä ja solmuilla on enemmän aikaa tulla hyviksi kuin pienemmillä raja-arvoilla.

Tarkasteltaessa solmujen lisäämistä tarjonnan kannalta sekä hyvät että huonot solmut valitsivat naapureikseen etupäässä paljon tarjoavia solmuja. Useimmiten huonot solmut tosin menettivät seuraavalla tarkastuskierroksella tuon valitsemansa hyvän solmun. Solmujen poistosta ei voida sanoa, että solmu olisi pudottanut naapureistansa aina sen huonoimman pois. Tämän tosin selittää jo aikaisemminkin mainittu määritelty aikaväli.

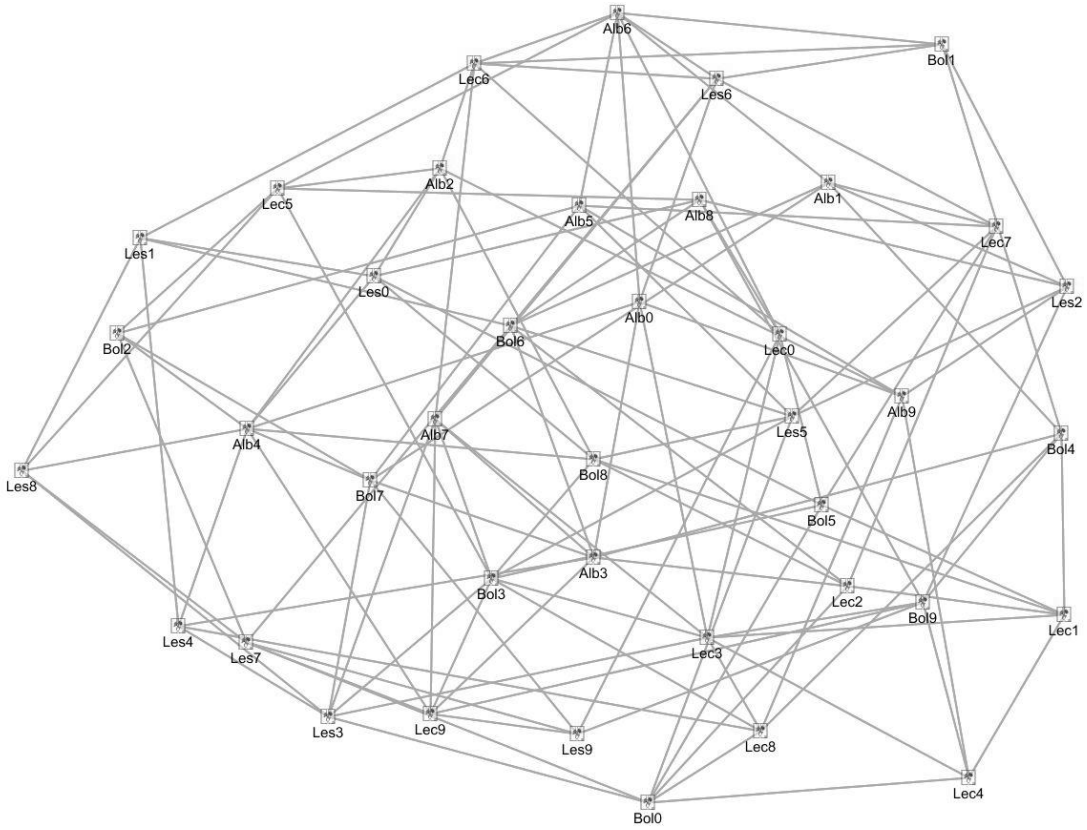
Määritelty liikenteen ala- ja ylärajan ero toimi, eikä algoritmin vaikutuksesta verkossa tapahtunut jatkuvaa edestakaista solmun poistamista ja lisäämistä jokaisella tarkastuskerralla.

Ajettaessa pienillä liikenteen rajoilla solmuja jäi hetkellisesti ilman yhtään naapuria. Solmut kuitenkin selvisivät hyvin naapureidensa kadotuksesta muodostaen jälleen yhtenäisen verkon seuraavalla kuormituksen tarkistuskerralla.

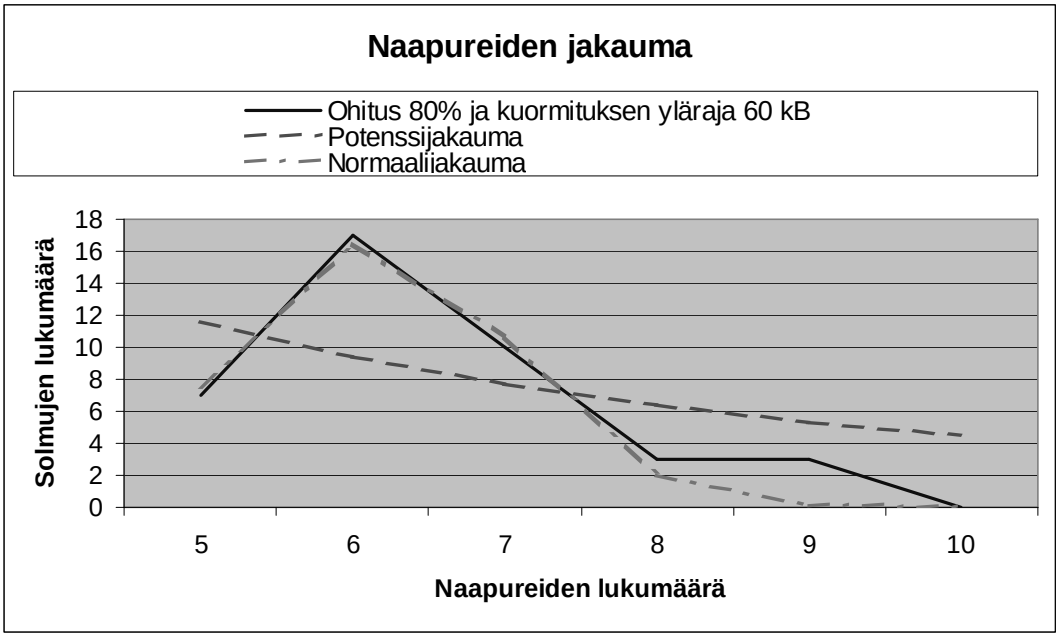
Määritellyllä aikavälillä voidaan vaikuttaa lisättäviin solmuihin. Vaikka lyhyellä aikavälillä solmu suosiikin ensimmäisiä yhteyksiä, joilta se on saanut paljon resursseja, niin toisaalta lyhyellä aikavälillä on se etu, että jos solmu joutuu eristyksiin muista solmuista, se pääsee nopeasti takaisin mukaan. Jos halutaan käyttää pidempää aikaväliä, mikä taas antaa paremmat mahdollisuudet uusille solmuille säilyä solmun naapureina, kannattaa solmulle määritellä kuormituksesta riippumaton uusien solmujen lisäämistapa, jos kaikki naapurit katoavat. Testissä olosuhteista johtuen oli kaikilla solmuilla sama kuormituksen raja-arvo ja aikaväli. Käytännössä aikavälin voisi pienemmillä raja-arvoilla säätää pienemmäksi, jolloin solmu toipuisi naapureidensa menetyksestä nopeammin.

7.3 Ohituksen ja kuormituksen yhteisvaikutus

Tutkimuksessa tutkittiin myös Solmun ohitus ja Kuormituksen arviointi -algoritmien yhteistoimintaa. Siinä ohitusprosenttina käytettiin Solmun ohitus -algoritmin testauksessa parhaaksi pääteltyä prosenttilukua ja kuormituksesta käytettiin tälle ympäristölle saatua hyvää ylärajaa. Yhteisvaikutusta testattiin 80% ohituksella ja 60 kilotavun kuormituksella. Edelleen kuormitustilanne tarkastettiin 50 lähetetyn resurssiviestin välein. Testissä saatu topologia ja sitä vastaava naapureiden jakauma ovat kuvissa 38 ja 39.



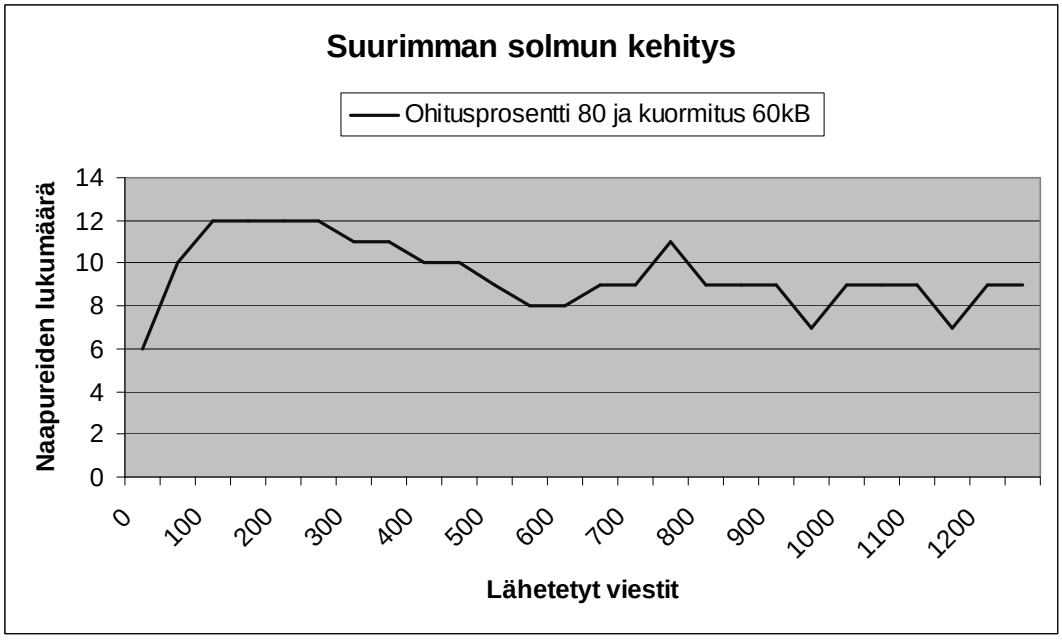
Kuva 38: Verkon topologia, kun ohitus oli 80% ja liikenteen raja 60 kilotavua.



Kuva 39: Naapureiden jakauma, kun ohitus oli 80 % ja liikenteen raja 60 kilotavua.

Saatu naapureiden jakauma ei noudata selkeästi potenssijakaumaa, mutta ei myöskään normaalijakaumaa. Jakauma erosi potenssijakaumasta virhefunktion arvolla 7.09 ja normaalijakaumasta arvolla 2.26. Verkossa suurimmalla osalla solmuja on kuusi naapuria, mutta toisaalta verkossa on myös solmuja, joilla on enemmän naapureita. Nämä kuusi eniten naapureita ympärilleen kerännyttä solmua olivat yhtä lukuun ottamatta paljon tarjoavia ja suurin osa myös paljon kyseleviä. Vähän tarjoavakin solmu oli kerännyt ympärilleen paljon tarjoavia solmuja, mikä selittää suuren naapureiden lukumäärän.

Kuvassa 40 on verkon suurimman solmun naapureiden lukumäärän kehitys testissä. 800 viestin jälkeen voidaan havaita, että suurimman solmun naapureiden lukumäärä tasoittuu ja vaihtelee sen jälkeen vain kahdella naapurilla 150 lähetetyn viestin välein.



Kuva 40: Suurimman solmun naapureiden lukumäärän kehitys, kun ohitusprosenttina oli 80 ja kuormituksessa liikenteen ylärajana 60 kilotavua.

Konvergoitumista ei kuorman arvioinnista johtuen päässyt tapahtumaan, koska solmuja lisättiin ja pudotettiin pois. Yhteisvaikutus ei siis tuottanut haluttuja tuloksia, koska konvergoituminen ja potenssijakauma jäivät puuttumaan.

7.4 Johtopäätökset

Tutkimuksen tarkoituksena oli tutkia topologian hallintaan kehitettyjen algoritmien toimintaa. Tässä vaiheessa tutkimuksessa keskityttiin tarkastelemaan algoritmien luomaa topologiaa ja varmistumaan, että käyttäytyminen on oikeanlaista. Topologioita arvioitiin niiden konvergoitumisen sekä naapureista saadun jakauman perusteella. Tarkoituksena oli mahdollisimman vähillä verkon muutoksilla saada verkko, jossa solmujen väliset etäisyydet olisivat lyhyitä.

Ohitusalgoritmilla saadut topologiat noudattivat potenssijakautunutta verkkoa, jossa keskussolmuina oli joko paljon tarjoavia solmuja tai keskussolmun naapurit olivat näitä hyviä solmuja. Potenssijakautuneet verkot ovat vikasietoisia ja solmujen väliset etäisyydet ovat lyhyempiä kuin satunnaisessa verkossa, joten ohitusalgoritmi toimi hyvin. Testiverkko saatiin lisäksi konvergoitumaan, jolloin verkossa saavutettiin solmujen välille tasapaino ja vältettiin turhia muutoksia.

Kuorman arvioinnissa jouduttiin testissä tyytymään arvioimaan solmujen lisäämiseen ja poistamiseen käytettyä kriteeriä. Solmujen hyvyyden määrittely näytti toimivan, vaikka tuloksiin vaikuttikin valittu kuorman arvioinnin tarkastusväli. Parempi solmun hyvyyden määrittely voitaisiin tosin saada suhteuttamalla osumat solmulle lähetettyihin kyselyihin, jolloin ensimmäiset yhteydet eivät saisi yhtä suurta etua kuin nyt.

Ohitusalgoritmin ja kuorman arvioinnin yhteisvaikutuksena saatiin verkko, jossa oli muutama muita suurempi solmu. Nämä solmut olivat lisäksi verkon kannalta hyviä solmuja, joten päästiin konvergoitumista vailla samaan tilanteeseen kuin ohitusalgoritminkin kanssa. Edelleen yhteisvaikutuksessakin tulokseen vaikutti valittu kuorman tarkastusväli sekä se, että kaikille solmuille oli määriteltynä sama raja liikenteen määrälle.

8 Yhteenveto ja tulevaisuus

Tutkimuksessa selvitettiin, miten vertaisverkkoalustaan kehitetyt algoritmit vaikuttavat muodostuvan verkon topologiaan. Tutkielman teoriaosuudessa käytiin läpi luvussa 2 vertaisverkkojen teoriaa sekä samaan tutkimusalaan liittyviä muita tehtyjä tutkimuksia. Vertaisverkoista käytiin tarkemmin läpi Gnutella, joka oli lähtökohtana kehitetyssä Chedar-vertaisverkkoalustassa. Koska tutkimus keskittyi tässä vaiheessa lähinnä saatujen topologioiden analysointiin, esiteltiin tutkielmassa graafeista tämän tutkimuksen kannalta olennainen osa ja eritoten potenssijakauma luvussa 3.

Käytännön osuus työstä oli Chedar-alusta ja siihen kehitetyt algoritmit. Chedar-vertaisverkkoalustan toiminta, sen käyttämät viestit ja alustan monitorointityökalu esiteltiin luvussa 4. Alustan topologian hallintaan käyttämät Solmun ohitus, Solmun valinta, Solmun poisto sekä Kuormituksen arviointi -algoritmit kuvattiin luvussa 5. Luvussa 6 esiteltiin tutkimusympäristö. Luvussa 7 kuvattiin käytetyt tutkimustapaukset, analysoitiin saadut tulokset ja käytiin läpi tutkimuksesta tehdyt johtopäätökset.

Jatkotutkimuksessa algoritmeja testataan suuremmalla solmumäärällä simulaation avulla, jolloin päästään eroon verkosta ja työasemista aiheutuvista ongelmista ja viivästyksistä ja saadaan luotettavampaa tietoa algoritmien käyttäytymisestä. Näin saadaan myös verkon solmujen määrää lisättyä huomattavasti. Tarkoituksena on ottaa arviointiin mukaan tietoja myös resurssikyselyihin saaduista vastauksista, jolloin voidaan verrata algoritmeilla saatuja topologioita esimerkiksi sen suhteen, kuinka monen hypyn takaa vastaukset löytyvät. Paitsi että verkko on vikasietoinen dynaamisessa ympäristössä, sen pitää olla myös tehokas haun kannalta. Tässä tutkimuksessa on vikasietoisuudessa ja haun tehokkuudessa luotettu ohitusalgoritmilla saatuun potenssijakaumatyyppiseen topologiaan, jossa paljon tarjoavat solmut ovat lyhyen etäisyyden päässä muista solmuista.

Algoritmeja on tarkoitus tulevaisuudessa tarkentaa ja lisätä niissä käytettyjä parametreja. Solmun poistossa on tarkoituksena etsiä hyvyyden määrittelyssä solmun naapureiden välittämille osumille painokerroin. Tällä hetkellä solmun naapureiden välittämiä osumia painotetaan saman verran kuin solmun osumia. Solmun hyvyys voidaan määritellä myös suhteessa solmuille lähetettyihin kyselyihin.

Cheese Factory -tutkimusprojektissa on käytetty neuroverkkoja hyvän hakualgoritmin muodostamiseksi vertaisverkossa ja saatu siitä lupaavia tuloksia. Neuroverkkoja on tarkoitus tulevaisuudessa käyttää myös topologian hallinta -algoritmien kehittämiseen ja niissä käytettyjen parametrien hyvien arvojen automaattiseen löytämiseen, jotta lopputuloksena saataisiin tehokas ja vikasietoinen vertaisverkko.

Lähteet

[1] Adar E., Huberman B.A., Free Riding on Gnutella, First Monday, vol. 5-10, lokakuu 2000.

[2] Agora Center, <URL: `http://www.jyu.fi/agora-center/`>.

[3] Alima L.O., Mesaros V., Van Roy P., Haridi S., *NetProber: A Component for Enhancing Efficiency of Overlay Networks in P2P Systems*, In Proceedings of the Second International Conference on Peer-to-Peer Computing, 2002, s. 25-32.

[4] Barabási A., ”Linked”, Perseus Publishing, Massachusetts, 2002.

[5] Barkai D., “Peer-to-Peer Computing”, Intel Press, 2002.

[6] Cheese Factory, <URL: `http://tisu.it.jyu.fi/cheesefactory/`>.

[7] Chawathe Y., Ratnasamy S., Breslau L., Lanham N., Shenker S., *Making Gnutella-like P2P Systems Scalable*, in ACM SIGCOMM, 2003, Germany.

[8] Cooper B.F., Garcia-Molina H., *Ad hoc, self-supervising peer-to-peer search networks*, In Technical Report, USA, 2003.

[9] Freenet, <URL: `http://freenet.sourceforge.net/`>.

[10] Glossary of Measurement Terms, <URL: `http://www.hemweb.com/library/glossary.htm/`>.

[11] Gnutella, <URL: `http://gnutella.wego.com/`>.

[12] Gnutella hosts, <URL: `http://www.gnutellahosts.com/`>.

[13] Heawood graph, <URL: `http://www.win.tue.nl/~aeb/drg/graphs/Heawood.html/`>.

[14] Hyytiälä H., *Fenfire in Peer-to-Peer Environment*, Pro gradu -tutkielma, Jyväskylän yliopisto, 2003.

[15] Iles M., Deugo D., *A Search for Routing Strategies in a Peer-to-Peer Network Using Genetic Programming*, In Proceedings of 21st IEEE Symposium on Reliable Distributed Systems, 2002, s. 341-346.

[16] IRC, <URL: `http://www.irc.org/`>.

[17] Krishna Ramanathan M., Kalogeraki V., Pryune J, *Finding Good Peers in Peer-to-Peer Networks*, In Proceedings of the International Parallel and Distributed Processing Symposium, 2002, s. 24-31.

[18] Lv Q., Ratnasamy S., Shenker S., *Can Heterogeneity Make Gnutella Scalable?*, In Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), USA, 2002.

[19] Napster, <URL: `http://www.napster.com` />.

[20] Oram Andy, "Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology", O'Reilly Books, 2001.

[21] Pandurangan G., Raghavan P., Upfal E., *Building Low-Diameter P2P Networks*, In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, s. 492-499.

[22] Ratnasamy S., Francis P., Handley M., Karp R., Shenker S., *A Scalable Content-Addressable Network*, In Proceedings of ACM SIGCOMM, USA, 2001.

[23] Ripeanu M., *Peer-to-Peer Architecture Case Study: Gnutella Network*, In Proceedings of the First IEEE International Conference on Peer-to-Peer Computing, Sweden, 2001, s. 99-100.

[24] Ripeanu M., Foster I., Iamnitchi A., *Mapping the Gnutella network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design*, Internet Computing, IEEE, Vol 6. Is. 1, 2002, s. 50-57.

[25] Rowstron A., Druschel P., *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Germany, 2001, s. 329-350.

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

[27] Stoica I., Morris R., Karger D., Kaashock M., Balakrishnan H., *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications*, In Proceedings of the ACM SIGCOMM, USA, 2001, s. 149-160.

[28] Tanenbaum A. S., Van Steen M., ”Distributed Systems: Principles and Paradigms”, Prentice Hall, USA, 2002.

[29] Usenet, <URL: `http://www.usenet.com/`>.

Liitteet

Liite 1: Solmun ohitus -algoritmi

```
overtake(Connection connection)
  Connection overtakeCon = getOvertakingNode(connection,
  overtakingPercent)
  sum = 0.0
  biggest = 0.0
  Connection bestNeighbor = null
  sum += getHits(connection)
  sum += getRelayedHits(connection)
  for all neighbors of connection
    hitValue = relayedHits(neighbor)
    proportion = hitValue/sum
    if(proportion >= overtakingPercent/100.00 && hitValue >
    1) then
      if(proportion > biggest) then
        biggest = proportion
        bestNeighbor = neighbor
      end if
    end if
  if(bestNeighbor != null) then
     if(addConnection(bestNeighbor))then
       disconnectConnection(connection)
     end if
  end if
```

Liite 2: Solmun valinta -algoritmi

```
addNewConnection(time)
  Connection connection = null
  Vector  connections  =  searchFromHistory("resource/hits",
  time)
  for all connections
    if(connect(connection))then
       return connection
    end if
  connections = searchFromHistory(null, time)
  for all connections
    if(connect(connection))then
       return connection
    end if
return null
```

Liite 3: Solmun poisto -algoritmi

```
getWorstConnection()
  goodness = 0
  lowest = biggestGoodnessValue
  Connection drop = null
  for all connections
    goodness = hits + relayedHits
    if(goodness <= lowest) then
      lowest = goodness
      drop = connection
    end if
  return drop
```

Liite 4: Kuormituksen arviointi -algoritmi

```
checkTrafficMeter(time)
  value = trafficMeter/1024.0
  Connection con
  if(value > trafficLimit) then
    con = getWorstConnection()
    if(con != null)  then
      disconnectConnection(con)
    end if
  else if(value < lowerTrafficLimit) then
    addNewConnection(time)
  end if
  resetTrafficMeter()
```