Kari Rapo

TCP- ja UDP-porttien uudelleenohjaus ohjelmallisesti

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
17.11.2010

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Kari Rapo TCP- ja UDP-porttien uudelleenohjaus ohjelmallisesti 34 sivua + 4 liitettä 17.11.2010
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikan suuntautumisvaihtoehto
Ohjaaja(t)	Lehtori Simo Silander Lehtori Marko Uusitalo Lehtori Jussi Alhorinne

Tämän työn tarkoituksena oli suunnitella ja toteuttaa ohjelma, joka kykenee ohjaamaan tietokoneen UDP- ja TCP-porteista vastaanotettua liikennettä uuteen osoitteeseen. Kohdeosoite voi sijaita missä tahansa verkossa, mihin kyseisellä tietokoneella on suora yhteys tai reitti. Ohjelma suorittaa liikenteelle osoitemuunnoksen, jonka jälkeen liikenne lähetetään edelleen ennalta määriteltyihin kohdeosoitteisiin. Tätä kutsutaan porttien uudelleenohjaamiseksi.

Ohjelman suunnittelu aloitettiin, koska käyttötarkoitukseen ei löytynyt vastaavaa valmista ilmaisohjelmaa, joka olisi toiminut halutulla tavalla. Tässä työssä esitelty toteutus on ainoastaan ensimmäinen versio projektista, jonka kehitys jatkuu, kunnes ohjelma saavuttaa halutun laatutason tai tarve ohjelmalle loppuu.

Ohjelma on suunniteltu toimimaan kevyen käyttöasteen alla olevissa SOHO-ympäristöissä *(Small Office / Home Office)*. Ohjelman käytölle missä tahansa muussa IP-verkkoja sisältävässä ympäristössä ei kuitenkaan ole varsinaisia teknisiä esteitä niin kauan, kun reitittimenä toimivan tietokoneen nopeus on riittävä kaiken liikenteen käsittelyyn.

Ohjelma on tarkoitus rajoittaa henkilökohtaiseen käyttöön ainakin, kunnes projektin jatkokehitys etenee pisteeseen, jossa ohjelman toteutukseen käytetyt ratkaisut ja ohjelman ominaisuudet eivät enää muutu radikaalisti. Tämän jälkeen ohjelmalle haetaan mahdollisesti lisenssi ja ohjelma vapautetaan julkiseen käyttöön.

Avainsanat	osoitemuunnos, ohjelma, TCP, UDP, NAT, PAT

Metropolia

Abstract

Author(s) Title Number of Pages Date	Kari Rapo Software based TCP and UDP port forwarding 34 pages + 4 appendices 17 Nov 2010
Degree	Bachelor of Engineering
Degree Programme	Information technology degree programme
Specialisation option	Software engineering specialization option
Instructor(s)	Simo Silander, Senior Lecturer Marko Uusitalo, Senior Lecturer Jussi Alhorinne, Senior Lecturer

The goal of this thesis-work was to design and implement software, that can re-direct incoming traffic from UDP and TCP ports to another computer. Destination address can be located in any network that the computer is directly connected with or has a route to. The software does address translation for the traffic, after which the traffic is sent to predefined destination addresses. This function is called port forwarding.

Reason for starting the development of this software was, because there were no free alternatives available, which would function in desired fashion. The implementation introduced in this thesis-work is only the first version of this software and the development will continue until software reaches desired level maturity or there isn't need for the software any longer.

Software has been designed to work in environments that are under light load conditions like most SOHO networks. However, there is no technical limitation against deploying the program into any IP network, as long as the computer acting as router is able to process all traffic that goes through it.

Software usage is planned to be limited to personal use until project reaches a point where all major features have been implemented and there will be no more radical changes in software structure between releases. After this the software might be licensed and released to public use.

Keywords	address translation, software, TCP, UDP, NAT, PAT

Sisällys

1 Johdanto

Internetissä palveluita tarjoavia palvelimia on jo pitkään siirretty yksityisiin verkkoihin tietoturvan asettamien haasteiden sekä IPv4-osoiteavaruuden asettamien rajoitusten vuoksi. Viime vuosina tämä suuntaus on kuitenkin kasvanut räjähdysmäisesti palvelinten virtuaalisoinnin ja virtuaalisten lähiverkkojen yleistymisen myötä. Palvelinten eristäminen useisiin yksityisiin verkkoihin aiheuttaa kuitenkin omia haasteitaan palveluiden tuomiselle Internetiin.

Kun palvelimet on eristetty yksityiseen verkkoon, eivät verkon ulkopuoliset palvelujen käyttäjät kykene kommunikoimaan suoraan palveluja tarjoavien palvelimien kanssa. Näissä tapauksissa täytyy reitittimen kyetä ohjaamaan julkisesta verkosta tulevat palvelupyynnöt oikeisiin osoitteisiin yksityisen verkon sisällä ja myös kyettävä palauttamaan vastaukset oikeisiin osoitteisiin julkisessa verkossa. Yhtä tämän ominaisuuden toteuttavaa tekniikkaa kutsutaan porttien uudelleenohjaamiseksi *(port forwarding)*.

Porttien uudelleenohjaus on vakio-ominaisuus lähes kaikissa nykyaikaisissa reitittimissä. Aina ei kuitenkaan ole perusteltua tai taloudellisesti mahdollista sijoittaa pääomaa kalliisiin reititysratkaisuihin. Näissä tapauksissa käytössä on usein Internet-gateway-palvelin, joka toimii reitittimenä julkisen ja yksityisen verkon välillä. Tämän tyyppisissä ratkaisuissa on kuitenkin usein varsin rajoitettu tuki porttien uudelleenohjaamiselle. Joissakin tapauksissa tätä ominaisuutta ei ole tarjolla ollenkaan.

Tämän työn tarkoituksena on suunnitella ja toteuttaa sekä Windows- että Linux-järjestelmissä toimiva ohjelma, joka mahdollistaa TCP/IP- ja UDP/IP-protokollien porttien uudelleenohjauksen kahden IP-verkon välillä. Ohjelmaa on tarkoitus käyttää kevyen käyttöasteen alaisissa verkoissa kuten SOHO-ympäristöissä. Ohjelma tulee käyttöön sekä Internetiin että yksityiseen verkkoon yhteydessä olevalle palvelimelle mahdollistaen näin ollen palvelujen tuottamisen erillisillä yksityisessä verkossa olevilla virtuaalipalvelimilla, joita voidaan tarpeen mukaan lisätä tai poistaa käytöstä.

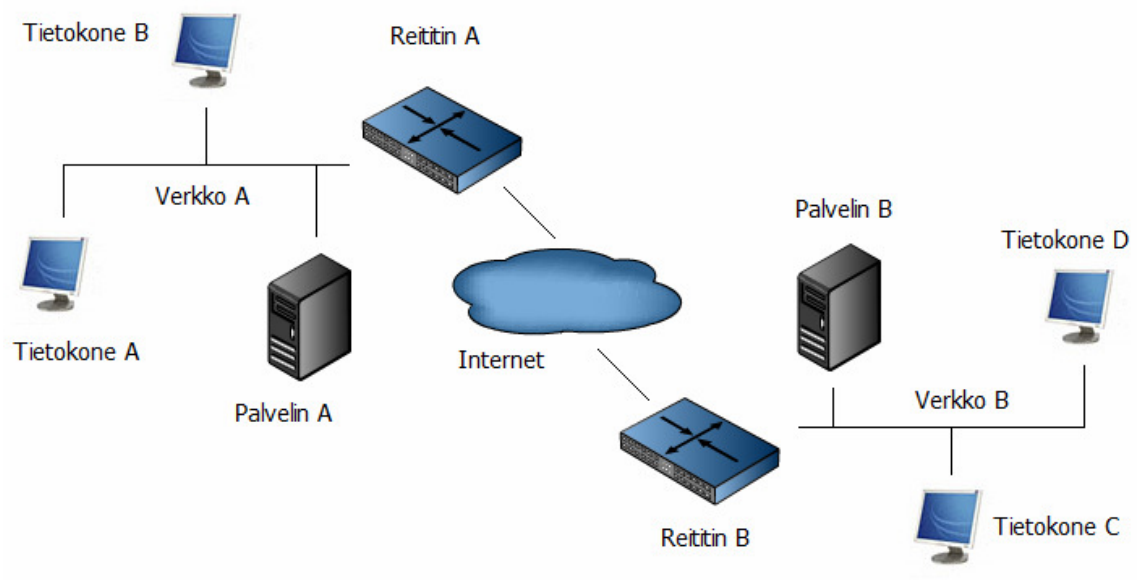
2 Perusasioita IP-verkoista

Porttien uudelleenohjaus on mekanismi, jossa tietokoneen yhteen porttiin tuleva tieto ohjataan edelleen ennalta määrättyyn kohdeosoitteeseen. Tämä osoite voi olla mikä tahansa IP-osoite-portti-pari, kun kyseisellä tietokoneella on vain yhteys IP-verkkoon, johon kohdeosoite kuuluu. Kohdeosoite voi jopa olla saman tietokoneen toinen portti, mutta tämä ei varsinaisesti ole tarkoituksenmukaista, sillä tähän tietokoneeseen saadaan jo yhteys eikä porttien ohjaukselle ole siis varsinaisesti tarvetta. Porttien uudelleenohjaamisen ja sen tarpeellisuuden ymmärtämiseksi on tunnettava IP-verkossa tapahtuvan kommunikoinnin perusteet sekä muutamia yleisiä IP-verkkoihin liittyviä käsitteitä, kuten pistokkeet ja osoitemuunnokset. Seuraavat luvut tekevätkin lyhyesti selkoa näistä käsitteistä ja siitä, kuinka ne liittyvät porttien uudelleenohjaamiseen.

2.1 IP-verkkojen käyttäytyminen

IP *(Internet Protocol)* on yksi maailman tunnetuimmista verkkoprotokollista. Se kehitettiin alun perin Yhdysvaltojen puolustusministeriölle kokeelliseksi pakettiverkoksi, mutta on sittemmin yleistynyt maailmanlaajuiseksi verkkokommunikointistandardiksi. Internet-protokolla onkin tuettu tavalla tai toisella lähes kaikissa moderneissa verkotetuissa tietokonejärjestelmissä.

Internet-protokolla toteuttaa kaksi ominaisuutta verkkoihin: osoitteet ja siirrettävän datan pilkkomisen. IP-verkkoa eli Internetiä käyttävät laitteet käyttävät IP-osoitteita datapakettien välittämiseen kohti kohdetta. Datapaketin osoitetietojen perusteella toteutettavaa pakettien käsittelyä kutsutaan yleisesti reitittämiseksi. IP-datapaketteja voidaan tarpeen vaatiessa pilkkoa pienempiin osiin, mikä mahdollistaa kommunikoinnin eri pakettikokoja käyttävien verkkojen välillä käyttäen samaa protokollaa. Pilkottu data pidetään järjestyksessä käyttäen IP-datapakettien otsikoihin sisällytettyjä tietoja. [1, s. 1.]



Kuva 1. Esimerkki IP-verkoista

Samaan IP-verkkoon liitetyt tietokoneet (kuva 1) kykenevät kommunikoimaan keskenään suoraan ilman muiden laitteiden tai palveluiden apua. Tietokoneet A ja B voivat kommunikoida keskenään ja palvelimen A kanssa, mutta ne eivät voi kommunikoida verkkoon B tai Internetiin kuuluvien laitteiden kanssa ilman reititystä. Kaikki kolme verkkoa (kuva 1) ovat itse asiassa Internet-verkkoja, mutta yleisessä puheessa julkisten IP-verkkojen nimeksi on vakiintunut Internet, joten lukijan mukavuuden vuoksi kutsumme yksityisiä Internet-verkkoja tässä työssä vain yksityisiksi verkoiksi ja julkisia Internet-verkkoja Internetiksi tai julkisiksi verkoiksi.

IP-verkot jakautuvat julkisiin ja yksityisiin verkkoihin. Yksityiset verkot ovat verkkoja, jotka käyttävät RFC1918:n mukaisia yksityisiä IP-osoitteita. [2, s. 3.] Yksityiset IP-osoitteet ovat yksityisiä, koska niitä ei koskaan jaeta millekään tietylle organisaatiolle. Nämä IP-osoitteet ovatkin yleisesti kaikkien käytettävissä. Tämä tarkoittaa myös sitä, että yksityisiin verkkoihin osoitettuja paketteja ei voida reitittää Internetin välityksellä, sillä samoja yksityisiä IP-osoitteita käytetään useissa eri verkoissa. Yksityisissä verkoissa voidaan toki käyttää myös julkisiksi määriteltyjä IP-osoitteita, mutta tällaiset verkot eivät voi mahdollisten osoitekonfliktien vuoksi kommunikoida julkisten verkkojen kanssa. Julkisiksi määriteltyjen IP-osoitteiden luvaton käyttö ei siis ole suotavaa, ellei käyttöönotettava verkko ole luonteeltaan täysin eristetty muista IP-verkoista.

Taulukko 1. Yksityiset IP-osoitteet.

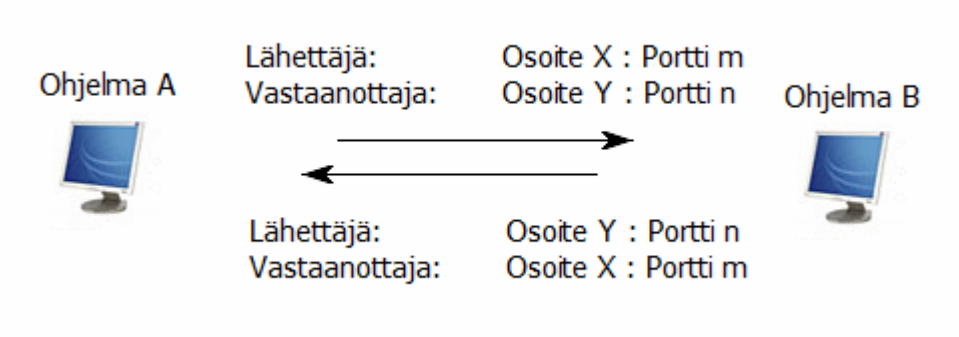
IP-osoitteet	**Osoitteiden määrä**
10.0.0.0 - 10.255.255.255	16,777,216
172.16.0.0 - 172.31.255.255	1,048,576
192.168.0.0 - 192.168.255.255	65,536

Tässä työssä käytetyssä esimerkissä oletetaan, että verkot A ja B (kuva 1) ovat yksityisiä verkkoja. On yhdentekevää, mikä osoitelohko (taulukko 1) tai osoitelohkon aliverkko on käytössä, kun se vain on yksityinen verkko, jolloin sen kommunikointi myös julkisten verkkojen kanssa voidaan järjestää. Verkkojen A ja B tietokoneet ja palvelimet voivat käyttää keskenään jopa täsmälleen samoja IP-osoitteita eikä tämä kuitenkaan aiheuta konfliktia osoitteiden välillä, sillä näiden kahden verkon tietokoneet eivät voi kommunikoida suoraan keskenään ilman erityisiä järjestelyitä.

2.2 Sovellusten välinen kommunikointi

Tietokoneet itse eivät varsinaisesti kommunikoi keskenään verkon välityksellä, vaan kyseessä on tietokoneella ajettavien ohjelmien välinen kommunikointi. Jokaisella tietokoneella on kuitenkin yleensä vain yksi IP-osoite jokaista liitettyä verkkoa kohden ja verkkoa käyttäviä ohjelmia on usein kymmeniä. Ohjelmat eivät kuitenkaan kommunikoi pelkän Internet-protokollan välityksellä, vaan käyttöön otetaan siirto-protokollat, jotka käyttävät Internet-protokollaa oman toimintansa perustana. Siirto-protokollat jakavat IP-osoitteet portteihin, joiden välillä yksittäisten ohjelmien kommunikointi tapahtuu. Ohjelmien välinen kommunikointi ei siis tapahdukaan pelkän IP-osoitteen perusteella, vaan osoitetietona käytetään yleensä TCP-. *(Transmission Control Protocol)* tai UDP- *(User Datagram Protocol)* pistokkeita *(Sockets)*.

Pistokkeet edustavat IP-verkossa kommunikoivia ohjelmia. Ne muodostetaan tietokoneen IP-osoitteesta ja käyttöön valitusta protokollasta riippuen ohjelmalle varatusta TCP- tai UDP-portista. Kun IP-osoite kertoo, millä verkkoon liitetyllä tietokoneella ohjelma sijaitsee, kertoo portti, mistä kyseisen ohjelman tavoittaa tietokoneen sisällä.



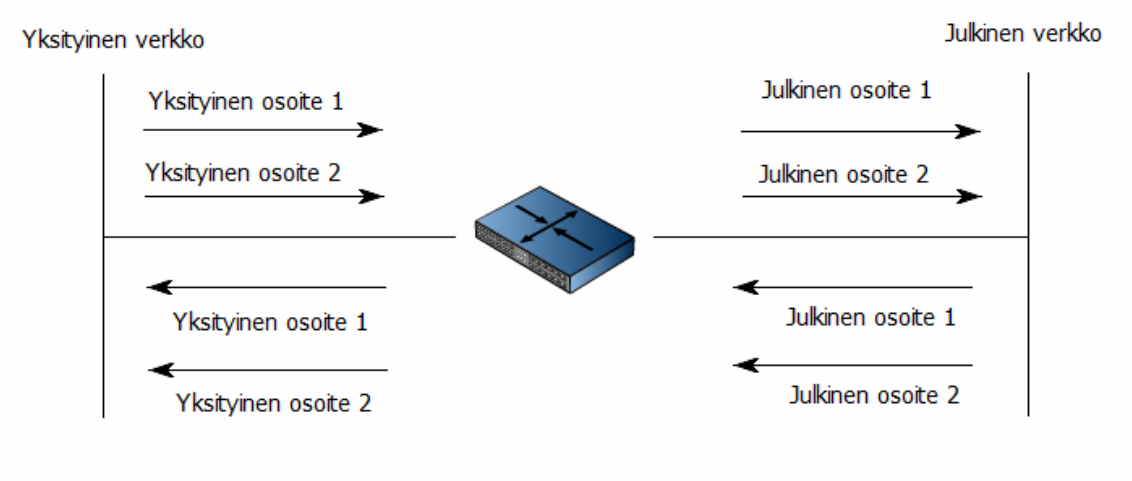
Kuva 2. Ohjelmien välinen kommunikointi

Aina, kun ohjelma A lähettää tietoa ohjelmalle B (kuva 2), liitetään lähetyspistokkeen tiedot lähetettävään datapakettiin. Jokainen datapaketti sisältää siis kaksi pistokeosoitetta, lähettäjän ja vastaanottajan. Ohjelma B:n vastatessa ohjelmalle A on vastaus lähetettävä samaan pistokkeeseen, josta ensimmäinen viesti lähetettiin. Ohjelmien välinen kommunikointi ei ole rajoitettu kahteen ohjelmaan. Yksi ohjelma voi luoda uusia yhteyksiä niin kauan, kuin vapaita portteja on käytettävissä. Yhtä pistoketta voidaan kuitenkin käyttää ainoastaan yhden pistokkeiden välisen linkin muodostamiseen kerrallaan. Jos yhdeltä tietokoneelta siis avataan tarpeeksi yhteyksiä, voivat portit teoreettisesti loppua kesken, jolloin uusien linkkien muodostaminen ei enää ole mahdollista. Pistokkeet eritellään 16 bittisillä luvuilla, joten niitä on yleisesti käytössä 65 536 kappaletta. Kun otetaan huomioon että käyttöjärjestelmät käyttävät muutamia portteja sisäiseen toimintaansa ja että kuuntelijat vaativat omat porttinsa, saadaan TCP-yhteyksien maksimi määräksi noin 32 000 yhteyttä. UDP-liikenteen tapauksessa voidaan käsitellä kaksinkertainen määrä käyttäjiä, sillä portteja vaaditaan kuuntelijan lisäksi vain yksi, TCP-yhteyksien tarvitseman kahden portin sijasta.

2.3 Yksityisen verkon liittäminen Internetiin

Kun yksityisestä verkosta halutaan kommunikoida Internetissä tarjolla olevien palveluiden kanssa, on järjestettävä reititys siten, että verkosta A (kuva 1) lähetettävät paketit osataan lähettää reititin A:lle ja tästä edelleen oikeaan julkiseen IP-osoitteeseen. Tämän lisäksi on Internet-palvelulta saatava vastaus osattava ohjata takaisin oikeaan verkon A IP-osoitteeseen. Tämän toiminnon tarjoaa IP-verkoille NAT *(Network Address Translation)* sekä PAT *(Port Address Translation)*.

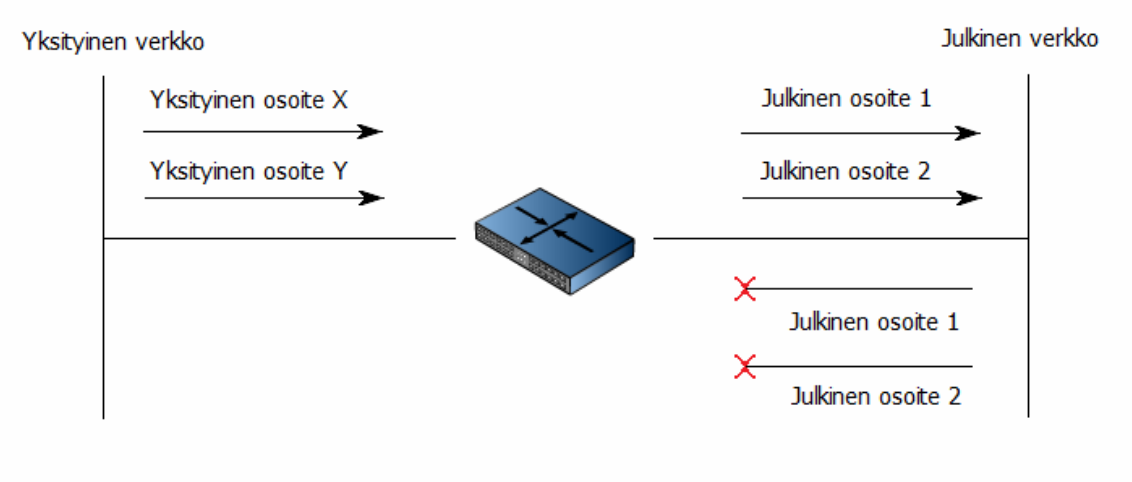
NAT on mekanismi, jossa yksityisen verkon tietokoneille annetaan reitittimellä käyttöön julkisen verkon IP-osoitteita. Reititin hallitsee osoitteenmuunnoksia joko staattisen tai dynaamisen mallin mukaan. Yhteisenä piirteenä molemmilla malleilla on kuitenkin tarvittavien julkisten osoitteiden määrä aktiivisen kommunikoinnin aikana. Jokaista NAT:ia samaan aikaan käyttävää yksityisen verkon tietokonetta kohden täytyy olla varattuna yksi julkisen verkon IP-osoite. [3, s. 2.]



Kuva 3. Staattinen verkko-osoitemuunnos

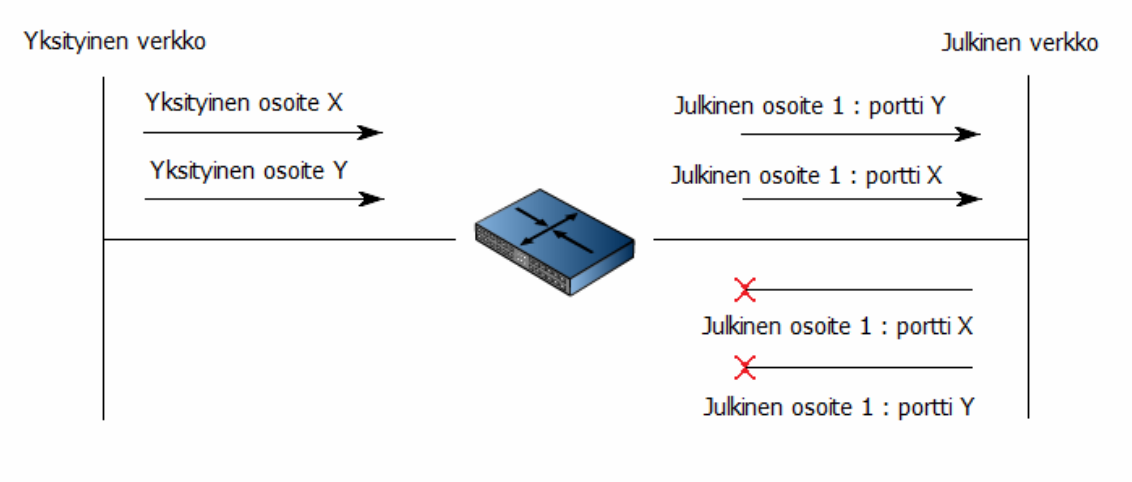
Kun käytössä on staattinen verkko-osoitemuunnos (kuva 3), määritellään reitittimellä käytettävissä olevat julkisen verkon osoitteet vastaamaan tiettyjä yksityisen verkon osoitteita. Tässä tapauksessa ainoastaan nämä valitut yksityisen verkon tietokoneet voivat avata yhteyksiä julkisen verkon palveluihin tai vastaanottaa julkisesta verkosta tulevia yhteyksiä. Tämän tyyppinen verkko-osoitemuunnos on erityisen hyödyllinen,

kun yksityisen verkon tietokoneisiin täytyy saada yhteys julkisesta verkosta tai tietyille yksityisen verkon tietokoneille täytyy taata yhteys. Riittävän monen julkisen IP osoitteen saaminen kuitenkin rajoittaa tämän tekniikan käyttöä Internetin yhteydessä. Tämän tyyppisen osoitemuunnoksen käyttö rajoittuukin usein pienien ja suurien yksityisten verkkojen saumattomaan liittämiseen sekä dynaamisen verkko-osoitemuunnoksen rinnalla käyttämiseen.



Kuva 4. Dynaaminen verkko-osoitteenmuunnos

Dynaamista tai ”perinteistä” verkko-osoitemuunnosta käytettäessä (kuva 4) jaetaan käytössä olevia julkisen verkon osoitteita tarpeen mukaan yksityisen verkon tietokoneille. Tämä ei kuitenkaan enää mahdollista julkisesta verkosta tulevien yhteyksien vastaanottamista, sillä reititin jakaa käytettävissä olevia julkisia osoitteita vain silloin, kun yksityisen verkon tietokone tarvitsee sellaisen. Tämä mahdollistaa kuitenkin yhteyksien avaamisen yksityisestä verkosta niin kauan, kun julkisia osoitteita on jaettavissa. [3, s. 2.]



Kuva 5. Portti-osoitemuunnos

Kolmas versio verkko-osoitemuunnoksesta on PAT *(Port Address Translation)*, josta käytetään myös nimityksiä osoitteen ylikuormitus *(Overloading)* yhden osoitteen NAT *(single address NAT)* tai porttitasolla limitetty NAT *(port-level multiplexed NAT)*. Tämä dynaaminen osoitemuunnos (kuva 5) on käytännössä huomattavasti yleisimmin käytetty tekniikka pienien verkkoympäristöjen yhteydessä ja Internetiin liityttäessä. Tämä muunnos muodostaa pareja sisäisen verkon pistokkeiden ja julkisen verkon porttien välillä, jolloin kaikki sisäisestä verkosta ulospäin suuntautuvat yhteydet näyttävät tulevan julkiseen verkkoon yhteydessä olevalta reitittimeltä. Tämän tekniikan vahvuutena on tarve ainoastaan yhdelle julkiselle IP-osoitteelle, mutta kuten dynaamista verkko-osoitemuunnosta käytettäessä ei julkisesta verkosta ole mahdollista muodostaa yhteyksiä yksityisen verkon tietokoneisiin ilman erityisiä järjestelyjä. [3, s. 5.]

2.4 Dynaamisen osoitteenmuunnoksen puutteiden paikkaaminen

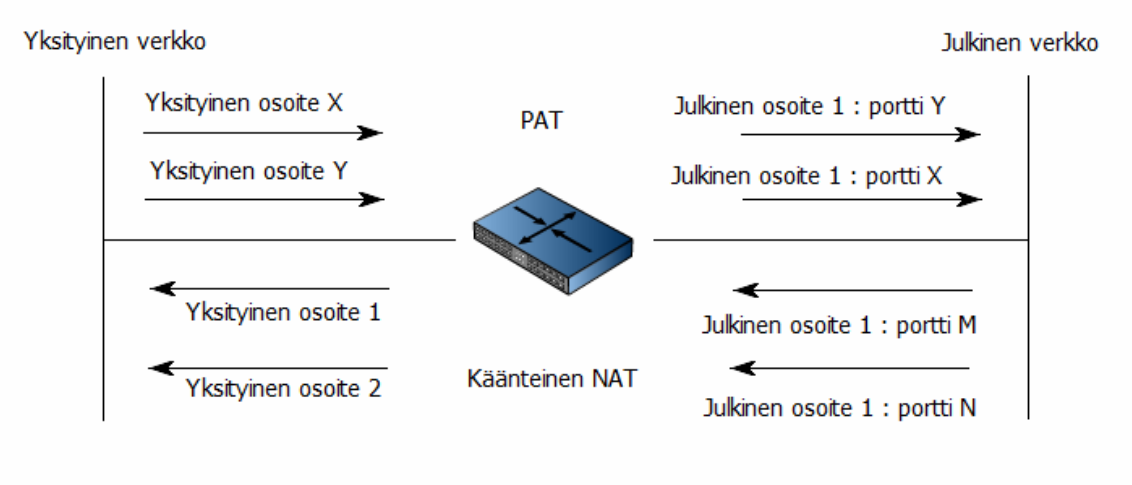
Sisäisessä verkossa olevat palvelimet eivät ole tietoisia muista kuin oman verkkonsa osoitteista eivätkä näin ollen voi kommunikoida suoraan muiden kuin yksityisen verkon jäsenten kanssa. Kun palvelin haluaa kommunikoida jossakin muussa verkossa sijaitsevan tietokoneen kanssa, se itse asiassa kommunikoi reitittimen kanssa. Reititin puolestaan lähettää tiedon haluttuun julkiseen osoitteeseen, jossa suurella todennäköisyydellä on toinen reititin, mikä puolestaan ohjaa liikenteen toiseen yksityiseen verkkoon.

Samoin kuin yksityisessä verkossa sijaitsevat palvelimet, eivät julkisessa verkossa sijaitsevat palvelimet ja tietokoneetkaan ole tietoisia muista kuin julkisen verkon osoitteista, joten niiden näkökulmasta kaikki kommunikointi tapahtuu yksityisten verkkojen reitittimien kanssa. Jos siis jokin julkisessa verkossa sijaitseva tietokone haluaa lähettää tietoa sisäisessä verkossa olevalle palvelimelle, on sen tapahduttava reitittimen välityksellä.

Kun kahden verkon välille on asetettu staattinen verkko-osoitemuunnospalvelu, voidaan yhteyksiä luoda kummankin verkon puolelta. Dynaaminen verkko-osoitemuunnos tai portti-osoitemuunnos ei kuitenkaan yksinään salli verkon A (kuva 1) ulkopuolelta aloitettujen yhteyksien reititystä verkon A tietokoneille. Kun jokin verkon A tietokoneessa sijaitseva ohjelma aloittaa kommunikoinnin jonkin toisessa verkossa sijaitsevan ohjelman kanssa, lähetetään ohjelman data reititin A:lle, joka lisää tarvittavan osoitemuunnoksen muistiinsa. Näin ollen verkon A ulkopuolella sijaitsevalta ohjelmalta saatavat paluuviestit voidaan reitittää takaisin oikeaan osoitteeseen verkon A sisällä. Tämä osoitemuunnos pysyy kuitenkin voimassa ainoastaan, kunnes kommunikointi lopetetaan, jonka jälkeen pakettien paluureitti on käyttökelvoton.

Jotta verkon A (kuva 1) palveluihin olisi mahdollista ottaa yhteyttä ulkopuolisista verkoista, on otettava käyttöön järjestelmä, joka suorittaa tarvittavan osoitemuunnoksen. Tällaista järjestelmää kutsutaan käänteiseksi verkko-osoitemuunnokseksi *(Reverse NAT)* tai porttien uudelleenohjaamiseksi *(Port Forwarding)*. Nimitys käänteinen NAT on varsinaisesti hiukan harhaanjohtava, sillä

kyseessä on reitittimen julkisen verkon puoleisiin portteihin tulevien yhteyksien ohjaaminen yksityisen verkon osoitteisiin. Mutta nimitys on jäänyt käyttöön, sillä portti-osoitemuunnosta pidetään verkko-osoitemuunnoksen yhtenä muotona.



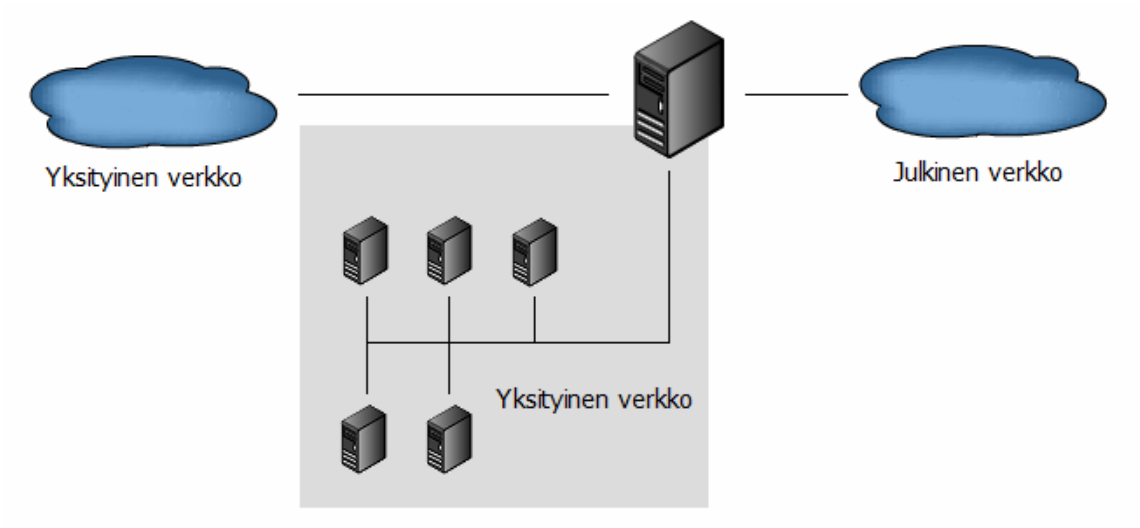
Kuva 6. Käänteinen NAT

Jos yksityisen ja julkisen verkon välissä olevalla reitittimellä otetaan käyttöön käänteinen NAT (kuva 6), saadaan aikaan pitkälti staattista verkko-osoitemuunnosta muistuttava tilanne. Erona on tarve ainoastaan yhdelle julkiselle IP-osoitteelle. Portti-osoitemuunnos tai dynaaminen verkko-osoitemuunnos toimivat edelleen normaalisti. Käänteinen NAT lisää vain staattisen listan julkisen ja yksityisen verkon pistokkeista muodostetuista pareista, joiden perusteella voidaan suorittaa osoitemuunnos yksityisen verkon ulkopuolelta aloitetulle kommunikoinnille.

3 Ohjelman toteutus

3.1 Ohjelman kohdeympäristö

Ohjelman kehitysympäristö sekä aiottu käyttöympäristö muodostuvat palvelimesta ja kahdesta verkosta. Palvelin isännöi virtuaalikoneita ja tarjoaa samalla reitityspalvelut paikalliselle lähiverkolle, johon myös virtuaalikoneet on liitetty.



Kuva 7. Ohjelman kohdeympäristö

Ympäristö (kuva 7) sisältää siis kaksi erillistä IP-verkkoa, joista yksi on julkinen verkko *(Internet)* ja toinen on yksityinen verkko, joka palvelee paikallisia tietokoneita sekä virtuaalikoneita. Vaikka virtuaalikoneet ovat olemassa ainoastaan niitä isännöivän palvelimen sisällä, näkyvät ne muille verkon käyttäjille tavallisina tietokoneina muiden joukossa. Yksityisen verkon käyttäjät voivat siis kommunikoida virtuaalikoneiden kanssa ilman porttien uudelleenohjausta, mikä säästää virtuaalikoneita isännöivän palvelimen resursseja.

Ohjelma käyttää osoitemuunnoksien toteuttamiseen IP-osoitteita ja portteja, eli pistokeosoitteita. Ohjelma ei käytä nimipalveluita, kuten WINS *(Windows Internet Name Service)* tai DNS *(Domain Name System)* varsinaisen osoitemuunnosprosessin aikana, sillä tämän olisi mahdollista aiheuttaa pakettien käsittelylle huomattavaa viivettä. IP-osoitteet on mahdollista etsiä nimipalveluiden, kuten WINS:n tai DNS:n avulla, mutta tämä toiminto on suoritettava osoitemuunnossäännön luonnin yhteydessä.

Koska osoitemuunnosohjelma ei kommunikoi nimipalveluiden kanssa, on osoitemuunnoksen kohteena olevien IP-osoitteiden oltava muuttumattomia.

Käytännössä tämä tarkoittaa DHCP-varauksien tekemistä tai staattisten osoitteiden käyttöä. Tässä tapauksessa jokaiselle virtuaalikoneelle on annettu staattiset osoitetiedot, mitkä on valittu DHCP-palvelun käyttämän osoitealueen ulkopuolelta. Suuremmassa ja monimutkaisemmassa ympäristössä on toki perusteltua käyttää DHCP-varauksia kiinteiden osoitteiden asettamiseen. Tällä tavoin on vaivattomampaa toteuttaa mahdollisia muutoksia myöhemmässä vaiheessa.

3.2 Työn suunnittelu

Kun porttien uudelleenohjausohjelman suunnittelu aloitettiin, asetettiin sille selkeät vaatimukset. Ohjelman tuli olla suoritettavissa sekä Windows- että Linux-järjestelmissä ilman erillistä kääntämistä. Ohjelman tarkoitus oli kyetä ohjaamaan sekä UDP- että TCP-liikennettä ja ohjelman täytyi suoriutua normaalin toiminnan aloittamisesta käynnistämisen yhteydessä ilman käyttöliittymän lataamista, jotta se olisi mahdollista käynnistää palvelimissa, joissa graafisen käyttöliittymän käytön mahdollistavat moduulit ladataan vain etäyhteyksien yhteydessä.

Ohjelman toteutukseen valittiin kieleksi Java huolimatta Java-virtuaalikoneiden suorituskyvyn vuosien mittaan saamasta kritiikistä. Käyttämällä ohjelmointikieltä mikä hyödyntää suoraa natiiveja kirjastoja, olisi ollut mahdollista saavuttaa pienempi muistijälki ja vähentää suorittimen käyttöaikaa. Tämä olisi kuitenkin vaatinut huomattavasti enemmän lähdekoodin optimointia, jotta järjestelmän kaikki teho olisi saatu käyttöön. Optimointiin kuluva aika olisi puolestaan pidentänyt kehitysaikaa huomattavasti. Koska ohjelman käyttöasteen odotettiin olevan varsin pieni, pääteltiin että käytännön ero Java-toteutuksen ja suoraan natiiveja kirjastoja käyttävän toteutuksen välillä tulisi olemaan häviävän pieni ja toteutustavaksi valittiin vähemmän kehitysaikaa vaativa ratkaisu.

Itse ohjelman toteutus ei kestänyt muutamia päiviä pidempään, sillä suuri osa käytetystä lähdekoodista oli jo valmiiksi toteutettu muiden projektien yhteydessä. Lisäksi ohjelma suunniteltiin modulaariseksi, eli jokainen ohjelman osio kykenee toimimaan itsenäisesti heti käynnistämisensä jälkeen. Tästä syystä oli ohjelman rungon ympärille yksinkertaista ja nopeaa koota joukko säikeitä, joista jokainen toteuttaa

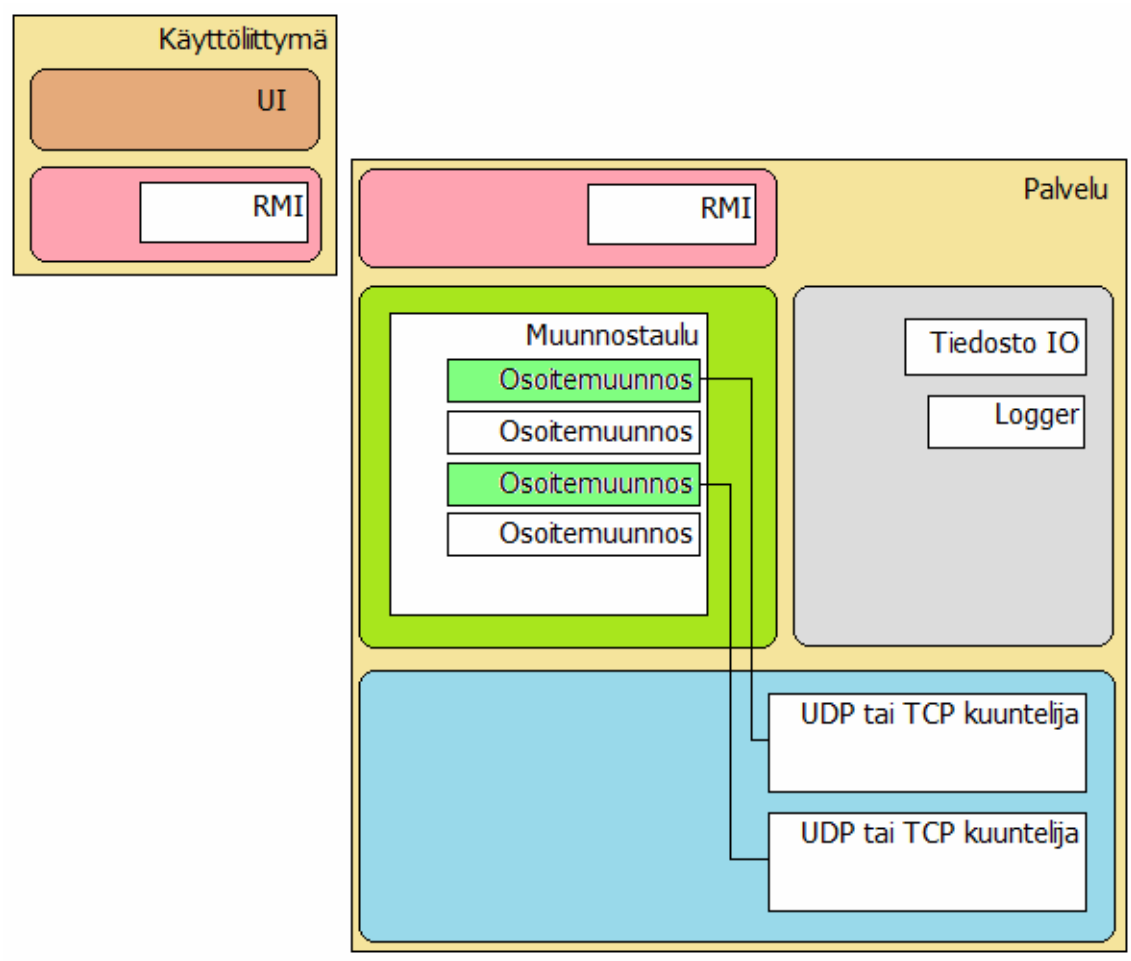
oman osa-alueensa ohjelman toiminnassa. Ohjelman modulaarinen rakenne helpottaa myös mahdollista jatkokehitystä olkoon kyseessä sitten jonkin toiminnon korvaaminen uudella toteutuksella tai uusien toimintojen lisääminen ohjelmaan.

Ohjelma käyttää varsin raskaasti säikeitä eri ominaisuuksien toteuttamiseen. Vaikka ohjelman eri osien toteuttaminen säikeillä kuluttaakin yleensä hieman enemmän käytettävissä olevia resursseja verrattuna pelkällä pääohjelmalla toteutettuihin ohjelmiin, on näiden resurssien käytöstä saatava etu mittaamattoman arvokas, kun käsitellään dataa, jonka käsittelyllä on aikaraja. Säikeiden käyttö ei ainoastaan mahdollista useamman kuin yhden pistokkeen yhtaikaisen käsittelyn, mutta myös mahdollistaa käytännöllisesti katsoen jatkuvan pistokkeiden kuuntelun.

Osoitemuunnoksen toteuttamiseen oli ohjelman suunnitteluvaiheessa käytettävissä kaksi vaihtoehtoista tapaa. Muunnos on mahdollista toteuttaa käsittelemällä yksittäisten pakettien osoitetietoja tai paketoimalla vastaanotettu data uudelleen kohdeverkkoon sopivaksi. Tähän ohjelmaan otettiin käyttöön jälkimmäinen tapa, koska sen toteutus ohjelmatasolla on yksinkertaisempaa. Osoitetietojen muokkausta käyttävä toteutus voisi mahdollisesti säästää järjestelmän resursseja ja nopeuttaa pakettien käsittelyä, mutta ohjelmalle suunnitelluilla käyttäjämäärillä ei myöskään uudelleen-paketoinnin aiheuttama kuorma ole huomattava missään modernissa järjestelmässä.

Toteutustavaksi valittiin siis yksinkertaisempi mutta silti ohjelman määrittelyn puitteissa toimiva ratkaisu. Jos ohjelmaa olisi tarkoitus ajaa suurilla käyttäjämäärillä ja minimaaliset resurssit omaavalla alustalla, tulisi osoitetietojen käsittelystä suurella todennäköisyydellä ainut käyttökelpoinen vaihtoehto. Tällaisessa tapauksessa tulisi kyseeseen myös ohjelmointikielen vaihto sekä natiivien kirjastojen käyttö, jotta järjestelmästä saataisiin kaikki mahdollinen teho käyttöön. Toisaalta tällaisessa tapauksessa olisi suositeltavaa ottaa käyttöön dedikoitu reititin laadukkaan palvelun takaamiseksi. Koska tämän ohjelman kohderyhmänä ovat kevyen kuorman alla olevat ympäristöt, päätettiin valitun toteutuksen olevan riittävä ohjelman määrittelyn asettamien vaatimusten täyttämiseksi.

3.3 Ohjelman perusrakenne



Kuva 8. Ohjelman rakenne

Ohjelman runkona toimii tietorakenne, joka sisältää kaikki ohjelmalle annetut osoitemuunnokset. Näitä osoitemuunnoksia edustaa ohjelmassa *Mapping*-tyyppiset oliot. Oliot erotellaan toisistaan kunkin osoitemuunnoksen luomisen yhteydessä generoiduilla *java.util.Calendar*-luokan instansseilla. Tällä tavoin päästään tarpeeksi lähelle uniikkia tunnistetta tämän ohjelman asettamien tarpeiden täyttämiseksi.

Luomisen yhteydessä osoitemuunnokset (kuva 8) asetetaan passiiviseen tilaan, jossa ne eivät pääse häiritsemään olemassa olevien muunnosten toimintaa. Osoitemuunnokset voidaan asettaa aktiiviseen tilaan ja takaisin passiiviseen tilaan keskeyttämättä ohjelman muuta toimintaa. Kun reititys on asetettu passiiviseksi, se on

ainoastaan läsnä pääohjelman tietorakenteessa. Kun osoitemuunnoksia puolestaan asetetaan aktiivisiksi, niille kullekin käynnistetään oma kuuntelija (kuva 8), joka hoitaa uusista lähteistä aloitetun kommunikoinnin osoitemuunnokset. Nämä säikeet pysyvät aktiivisina, kunnes osoitemuunnos asetetaan takaisin passiiviseen tilaan tai ohjelma sammutetaan. Jos kuuntelija jostain syystä lopettaa toimintansa tai sammuttaa itsensä kesken ohjelman suorittamisen, kirjataan vikatila ja JVM:n palauttama virheviesti ”error.log”-tiedostoon käyttäen ohjelman omaa Logger-palvelua (kuva 8).

Ohjelman käynnistämisen yhteydessä luetaan aikaisemmin tallennetut osoitemuunnokset tiedostosta ja samalla pyritään käynnistämään tarvittavat kuuntelijat aktiivisiksi merkityille osoitemuunnoksille. Jos vanhoja osoitemuunnoksia sisältävää tiedostoa ei löydy tai sitä ei voida muuttuneiden lukuoikeuksien tai korruptoitumisen vuoksi lukea, ohitetaan vanhojen osoitemuunnosten lataaminen ja ohjelma käynnistetään käyttäen tyhjää tietorakennetta.

```
ObjectInputStream input = new ObjectInputStream(
        new FileInputStream(“file.dat”));
MyObject object = (MyObject) input.readObject();
```

Esimerkkikoodi 1. Olion lukeminen tiedostosta

```
ObjectOutputStream output = new ObjectOutputStream(
        new FileOutputStream(“file.dat”));
output.writeObject(object);
```

Esimerkkikoodi 2. Olion kirjoittaminen tiedostoon

Osoitemuunnokset on talletettu tiedostoon oliomuodossa, mikä tarkoittaa sitä, että tieto täytyy myös lukea tiedostosta oliomuotoon. Tämä saadaan aikaan käyttämällä Java olio-tietovirtoja. Yksinkertaisesti sanottuna tiedosto avataan käyttäen tiedosto-tietovirtaa, mikä puolestaan kääritään olio-tietovirtaan. Tämän jälkeen luetaan olio-tietovirran sisältö oikeantyyppisen olion sisään (esimerkkikoodi 1). Osoitemuunnoksien tallentaminen toimii vastaavalla tavalla (esimerkkikoodi 2).

Pääohjelman ainut tarkoitus on toteuttaa käyttöliittymältä tulevia pyyntöjä, mitkä rajoittuvat reititykset sisältävän tietorakenteen ylläpitoon ja kuuntelijoiden aktivoimiseen sekä sammuttamiseen. Pääohjelman välityksellä voi siis lisätä, poistaa, käynnistää ja sammuttaa osoitemuunnoksia. Jokaisen muutoksen jälkeen osoitemuunnokset sisältävä tietorakenne tallennetaan tiedostoon käyttäen Tiedosto IO -palvelua (kuva 8). Osoitemuutoksia voidaan lisätä ja poistaa ja niiden tiloja voidaan muuttaa mielivaltaisesti ohjelman ajon aikana. Ainoana rajoituksena on kuuntelijan käynnistysfunktion yhteyteen toteutettu tarkastus, joka estää kuuntelijan käynnistyksen, jos samalla protokollalla ja samassa portissa on jo aktiivinen kuuntelija. Uusia reitityksiä voidaan asettaa käyttämään samaa porttia niin monta kuin on tarpeen, mutta ainoastaan yksi näistä kuuntelijoista voi olla aktiivisena.

Ohjelma toimii siltana kahden eri verkon välillä. Protokollasta riippuen ohjelma hallitsee joko yhteyksiä tai datapaketteja kahden pistokeparin välillä. Ohjelma ei millään tavoin lisää käytettävissä olevien porttien määrää. Käytössä ovat ainoastaan reitittimenä toimivan palvelimen portit, kun ohjelma ottaa vastaan yhteyksiä julkisen verkon puolelta. Tämän ohjelman rakennetta mukaillen on toki mahdollista toteuttaa kuormanjako, missä jokaiseen osoitemuunnokseen liitetään useita kohteita, mutta tässäkin tapauksessa listasta valitaan vain yksi osoite, johon liikennettä ohjataan. Tätä ominaisuutta ei kuitenkaan ole toteutettu ohjelman tähän versioon.

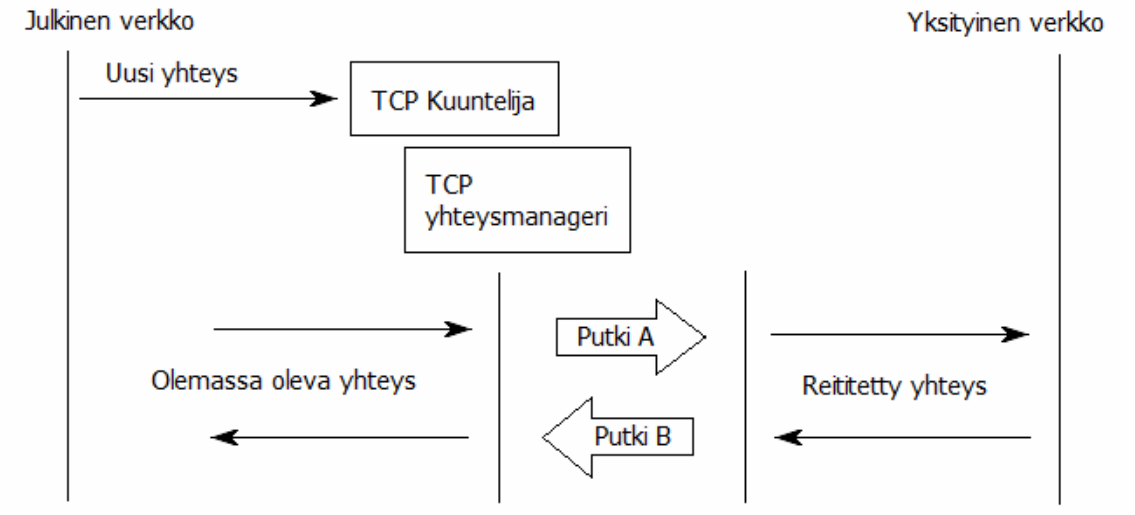
3.4 TCP/IP-yhteyksien käsittely

TCP on yhteysperustainen luotettavaksi suunniteltu protokolla, joka muodostaa linkkejä ohjelmien välille. Se on tarkoitettu käytettäväksi toisen protokollan yhteydessä, joka tarjoaa TCP-protokollalle yksinkertaisen tavan siirtää datapaketteja verkon yli. Tämän työn tapauksessa tämä toinen protokolla on Internet-protokolla [4, s. 2]. TCP-protokollaa käytetään yleensä tilanteissa, missä tiedon täytyy siirtyä ohjelmien välillä oikeassa järjestyksessä, ja kun virheet tai jopa kokonaisten pakettien puuttuminen ei tule kyseeseen. TCP-protokolla varmistaakin tiedon perillepääsyn kuittauksilla ja uudelleenlähetyspyynöillä, mikä tekee protokollasta varsin luotettavan tiedonsiirtoprotokollan. Kuittaukset ja uudelleenlähetykset aiheuttavat kuitenkin ylimääräistä liikennettä, minkä johdosta TCP-protokolla ei aina kykene täysin

hyödyntämään tarjolla olevaa siirtonopeutta. TCP-protokollan suorituskyky heikentyy huomattavasti varsinkin kun hukattujen IP-pakettien määrä kasvaa suureksi tai verkon latenssi on suuri TCP-pakettien sisältämän datan määrään nähden, jolloin kuittausten odottamiseen kuluva aika kasvaa huomattavan pitkäksi.

TCP-osoitemuunnoksen tapauksessa ohjelma ei varsinaisesti käsittele yksittäisiä paketteja, vaan osoitteenmuunnos toteutetaan yhteystasolla. Käytännössä Ohjelma pitää yllä kahta TCP-yhteyttä jokaista julkisesta verkosta tulevaa yhteyttä kohden. Ensimmäinen yhteyksistä tulee julkisesta verkosta reitittimelle, ja toinen muodostetaan reitittimen ja yksityisessä verkossa sijaitsevan palvelimen välille. Tiedonsiirto näiden kahden yhteyden välillä suoritetaan bittivirtoina, kun varsinaisten pakettien muodostus jätetään Java-virtuaalikoneen huoleksi.

TCP-kuuntelija (kuva 9) käyttää *java.net.ServerSocket*-tyyppistä pistoketta uusien yhteyksien vastaanottamiseen. Kun TCP-kuuntelija ottaa vastaan uuden yhteyspyynnön, se luo yhteyttä varten uuden pistokkeen ja käynnistää TCP-yhteysmanagerin, jolle annetaan luotu pistoke sekä yksityisen verkon osoitetiedot parametreinä. Tämän jälkeen TCP-kuuntelija siirtyy jälleen odottamaan uusia yhteyspyyntöjä.



Kuva 9. TCP-yhteyksien käsittely

```
Pipe inbound = new Pipe(
          new BufferedInputStream(
                public_socket.getInputStream()),
          new BufferedOutputStream(
                private_socket.getOutputStream()));
Pipe outbound = new Pipe(
          new BufferedInputStream(
                private_socket.getInputStream()),
          new BufferedOutputStream(
                public_socket.getOutputStream()));
try { inbound.start();
      outbound.start(); } catch { … }
```

Esimerkkikoodi 3. TCP-putkien käynnistys

TCP-yhteysmanagerin (kuva 9) ainoat tehtävät ovat yksityisen verkon sisäisen yhteyden luominen, ja kahden nyt olemassa olevan yhteyden välisten tietovirtojen eli putkien käynnistys (esimerkkikoodi 3), minkä jälkeen TCP-yhteysmanageri sammuttaa itsensä. Nämä tehtävät voitaisiin toteuttaa myös TCP-kuuntelijassa, mutta käyttämällä omaa säiettä näiden tehtävien suorittamiseen, saadaan minimoitua aika, jolloin TCP-kuuntelija ei pysty vastaanottamaan uusia yhteyksiä.

```
int byteCount = -1;
while((byteCount = in.read()) != -1) {
   out.write(byteCount);
   int buffered = in.available();
   if(buffered = > 1) {
      buffer = new byte[buffered];
      byteCount = in.read(buffer, 0, buffered);
      out.write(buffer, 0, byteCount);
   }

   out.flush();
}
```

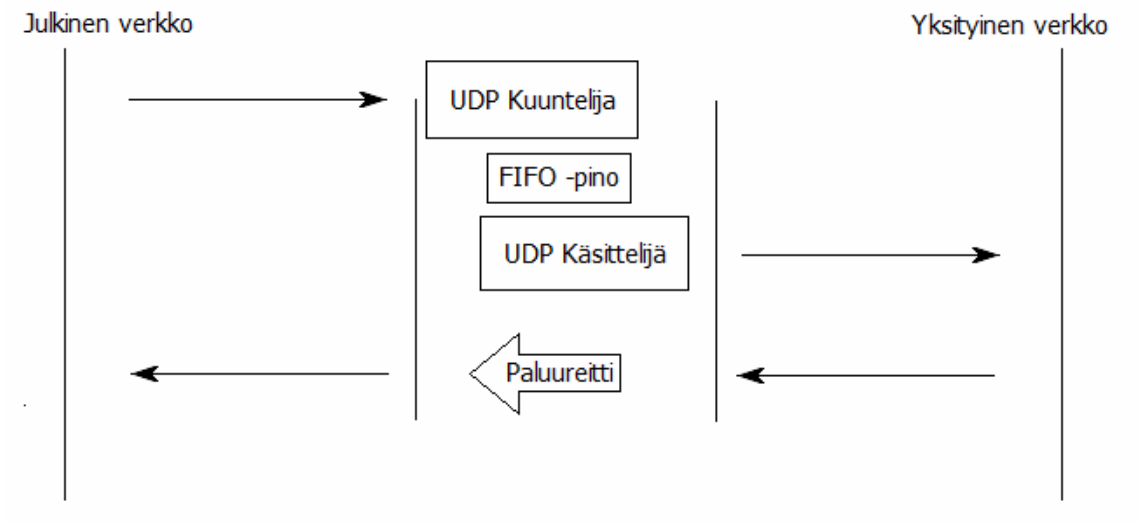
Esimerkkikoodi 4. TCP-tietovirtojen käsittely

Tietovirtojen hallintaa hoitavat kaksi säiettä, joita kutsutaan tässä sovelluksessa putkiksi (kuva 9). Kaikessa yksinkertaisuudessaan putket lukevat bittejä yhden pistokkeen tietovirrasta ja siirtävät ne toisen pistokkeen tietovirtaan (esimerkkikoodi 4). Sama funktio jatkuu, kunnes toisen pistokkeen tietovirran loppu havaitaan. Tämä aiheuttaa käsiteltävän pistokkeen sulkeutumisen, mikä puolestaan aiheuttaa myös putkessa jäljellä olevan pistokkeen sulkemisen ja lopulta koko putken pysäyttämisen. TCP-yhteyksiä ei siis tarvitse erikseen sulkea, kun yhteys katkaistaan joko julkisen tai yksityisen verkon puolelta, vaan putkien rakenne hoitaa yhteyksien hallinnan automaattisesti.

3.5 UDP/IP-pakettien käsittely

UDP-protokolla on yhteydetön tiedonsiirtoprotokolla, jota käytetään yleensä Internet-protokollan yhteydessä. UDP-protokolla ei muodosta yhteyksiä pistokkeiden välille, kuten TCP-protokolla, eikä myöskään sisällä minkäänlaista mekanismia tiedon perillepääsyn takaamiseksi. [5, s. 1.] Tästä seuraa oikeissa olosuhteissa huomattava suorituskykyetu TCP-protokollaan nähden. Koska UDP-protokolla ei itse lähetä kuittausviestejä tai pyydä pakettien uudelleenlähetystä, voidaan kaikki käytettävissä oleva tiedonsiirtokapasiteetti käyttää varsinaisen tiedon siirtämiseen. UDP onkin suosittu protokolla erilaisten suoratoisto- eli *streaming*-sovellusten ja VOIP-sovellusten *(Voice Over Internet Protocol)* yhteydessä. Kun parin satunnaisen paketin katoaminen ei kriittisesti vaikuta palvelun laatuun, kuten musiikin tai videon suoratoistossa, on UDP hyvä valinta tiedonsiirtoprotokollaksi.

Toisin kuin TCP/IP-protokolla, UDP/IP-protokolla ei muodosta yhteyksiä sovellusten välille, eikä kuunneltavaa pistoketta voida sitoa yhteen linkkiin, joten liikennettä ei voida käsitellä tietovirtoina. Jokainen vastaanotettu UDP-paketti täytyy käsitellä erikseen ja samalla täytyy pitää yllä taulukkoa julkisista osoitteista, jotta yksityisessä verkossa sijaitsevalta palvelimelta saatavat vastaukset voidaan lähettää edelleen oikeisiin osoitteisiin.



Kuva 10. UDP-pakettien käsittely

Kun UDP-kuuntelija (kuva 10) ottaa vastaan paketin, se ei itse jää käsittelemään sitä, vaan työntää sen välittömästi FIFO-tyyppiseen *(First In, First Out)* tietorakenteeseen, josta aiemmin käynnistetty UDP-käsittelijä käy sen noutamassa. Tällä toiminnalla pyritään minimoimaan aika, jolloin UDP-kuuntelija ei kykene ottamaan vastaan uusia paketteja ja näin ollen minimoimaan hukattujen pakettien määrä. UDP-kuuntelija on tätä silmällä pitäen pyritty toteuttamaan mahdollisimman kevyeksi (esimerkkikoodi 5).

```
while(active) {

        dg_packet = new DatagramPacket(
        new byte[65507], 65507);

        try { dg_socket.receive(dg_packet); }
        catch (IOException IOEx) {}

        if (!manager.get_state(mapping.get_ID()))
                break;

        try { buffer.offer(dg_packet); }
        catch (InterruptedException e) { return; }

}
```

Esimerkkikoodi 5. UDP-pakettien vastaanottaminen

Kuuntelija ja käsittelijä eivät tietenkään voi lukea ja kirjoittaa samaa pakettia samaan aikaa. Tästä syystä on pakettien siirtämiseen käytetyn tietorakenteen funktiot synkronoitu, mikä tarkoittaa tässä tapauksessa yksinkertaisesti sitä, että vain yksi säie kerrallaan voi käsitellä tietorakennetta. (liite 4). Käsittelijän on myös turha lukea tietorakennetta, jos se on tyhjä. Jos tietorakenteessa ei ole käsiteltäviä paketteja, keskeytetään käsittelijän toiminta, kunnes uusia paketteja on tarjolla. Tämän mallin heikkoutena on tilanne, jossa UDP-kuuntelija saattaa joutua odottamaan vuoroaan, kunnes käsittelijä on lopettanut oman funktionsa suorittamisen. Tämä puolestaan pidentää jaksoa, minkä aikana kuuntelija ei kykene ottamaan uusia paketteja vastaan. Ideaaliseen tilanteeseen päästäisiinkin, jos kuuntelija kykenisi tarvittaessa keskeyttämään muiden säikeiden toiminnan ja palaamaan pistokkeen kuuntelutilaan mahdollisimman nopeasti. Tällaisen mekanismin toteuttaminen jää kuitenkin ohjelman tulevien versioiden aiheeksi, jos nykyisen mekanismin nopeus havaitaan riittämättömäksi.

UDP-käsittelijä (kuva 10) sisältää varsinaisen pakettien käsittelyyn tarvittavan logiikan. Kun käsittelijä saa uuden paketin, se vertaa vastaanotetun paketin lähetyspistoketta UDP-kuuntelijan käynnistämän väliaikaisia osoitemuunnoksia ylläpitävän tietorakenteen sisältöön. Jos vastaavaa osoitetta ei löydy tietorakenteesta, käsittelijä avaa uuden pistokkeen ja muodostaa uudesta pistokkeesta sekä paketin lähetys-pistokkeesta parin, joka lisätään taulukkoon. Tämän jälkeen säie käynnistää toisen kuuntelijan, joka ohjaa yksityisestä verkosta saatavat vastauspaketit takaisin oikeaan julkisen verkon osoitteeseen. Jos lähetyspistoke löytyy tietorakenteesta tarkastuksen aikana, olettaa käsittelijä, että tällä ”UDP-yhteydellä” on jo aktiivinen paluureitti ja se siirtyy suoraan varsinaiseen paketin käsittelyyn. Kun on varmistettu, että palvelimelta saatavat paluuviestit voidaan lähettää takaisin julkiseen verkkoon, annetaan paketille uusi kohdeosoite ja lähetetään paketti takaisin verkkoon.

Vaikka jotkin UDP:tä käyttävät asiakas-palvelinjärjestelmät kertovat käyttäjille, että asiakasohjelman ja palvelimen välille on muodostettu yhteys, ei varsinaista yhteyttä ole kuten TCP-protokollaa käytettäessä. Tästä johtuen täytyy järjestelmän hallita dynaamisesti omaa osoitelistaansa. Jos yksityisen verkon puolelle avattuja UDP-pistokkeita ei koskaan suljeta, loppuvat pistokkeet lopulta kesken eikä uusista julkisen verkon osoitteista tulevia paketteja voida enää lähettää eteenpäin. Tämä ongelma on

ratkaistu asettamalla yksityisen verkon puolelle avatun pistokkeen aktiivisuudelle aikaraja. Jos pistokkeesta ei vastaanoteta liikennettä aikarajan puitteissa, pistoke suljetaan ja pistoke poistetaan UDP-kuuntelijan osoitetaulukosta. Järjestelmän tässä versiossa aikaraja on asetettu 120 000 millisekuntiin, eli kahteen minuuttiin, mikä on osoittautunut toimivaksi tähän mennessä testattujen palveluiden kanssa. Jos siis halutaan luoda illuusio yhteyden ylläpitämisestä, kun käytössä on UDP, on dataa lähetettävä molempiin suuntiin vähintään kerran kahdessa minuutissa. Aikarajaa olisi mahdollista lyhentää suuremman pistokemäärän pitämiseksi käytössä, mutta ohjelmalle suunniteltujen yhteys- ja liikennemäärien kanssa tämä ei ole tarpeellista. Tulevia järjestelmäversioita silmälläpitäen on joka tapauksessa suunnitteilla käyttäjän valittavissa olevan aikarajan lisääminen reitityksiin, jotta ohjelma saadaan tukemaan paremmin sovelluksia, joiden asetuksia ei päästä muuttamaan vapaasti.

3.6 RMI:n käyttö käyttöliittymärajapintana

Ohjelman käyttöliittymän liittämistavaksi valittiin RMI-rajapinta *(Remote Method Invocation)*. RMI on tekniikka, joka sallii ohjelman suorittaa funktioita verkkoyhteyden toisessa päässä sijaitsevan eli etäohjelman sisällä. Etäohjelma pystyy myös kutsumaan paikallisen ohjelman metodeita, jos paikallisen ohjelman tunnistamiseen vaadittavat funktiot on toteutettu ohjelmassa.

RMI-palvelu pyritään käynnistämään ohjelman käynnistämisen yhteydessä. Palvelun käynnistymisessä mahdollisesti tapahtuvat virheet eivät kuitenkaan estä koko ohjelman toimintaa. Virhetilassa ohjelma yksinkertaisesti ohittaa RMI:n käynnistämisen ja kirjoittaa virheen aiheuttaman viestin *error.log*-tiedostoon. Ohjelmassa RMI:n tarvitsemat rekisterit on sulautettu itse ohjelmaan (esimerkkikoodi 6) eikä niitä tarvitse näin ollen käynnistää erikseen ohjelman käynnistämisen yhteydessä. Jotta RMI-rajapintojen palveluita voitaisiin käyttää, ne täytyy julkaista palvelua käynnistettäessä. Manuaalisesti tehtynä tämä toteutettaisiin *java.rmi.server.UnicastRemoteObject*-luokan avulla (esimerkkikoodi 7). Tässä ohjelmassa tämä ei kuitenkaan ole tarpeen sillä RMI rajapinnan toteutus "*Remote*" perii *UnicastRemoteObject*-luokan ja näin ollen rajapinnan toteutus julkaistaan rajapinta-olion alustuksen yhteydessä.

```
int port = 65000;
String rmiURL = ”rmi://localhost” + port + “/Remote”;
RemoteIface remote = new Remote(this);
LocateRegistry.createRegistry(port);
Registry registry = LocateRegistry.getRegistry(port);
registry.rebind(rmiURL, remote);
```

Esimerkkikoodi 6. RMI-palvelun käynnistäminen

```
RemoteIface remote = new Remote(this);
UnicastRemoteObject.exportObject(remote);
```

Esimerkkikoodi 7. RMI-rajapinnan julkaiseminen

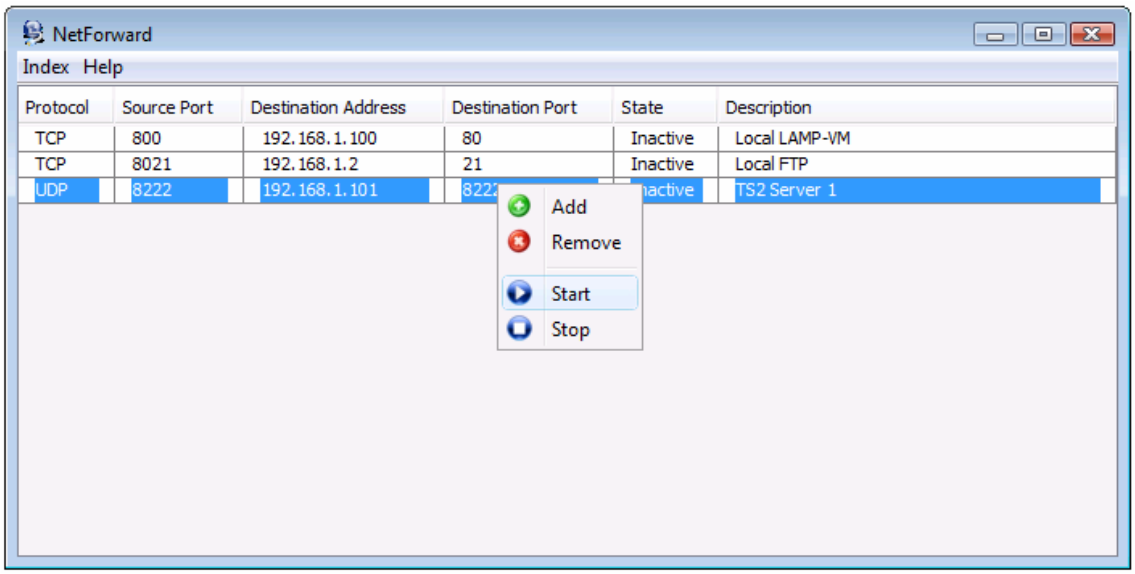
Syy RMI:n valinnalle oli yksinkertainen. RMI on helppo ja varma tapa välittää oliomuotoista tietoa kahden ohjelman välillä. Tarvittavat rakenteet oli myös työtä aloitettaessa jo olemassa, joten ajankäytön kannalta tämä oli varsin edullinen ratkaisu. Ominaisuuksiensa ja yksinkertaisuutensa puolesta tämä tekniikka on loistava valinta järjestelmään, jossa kahden tai useamman eri ohjelman on tarkoitus kommunikoida keskenään. Suunnittelun yhteydessä ei kuitenkaan osattu ottaa huomioon RMI-rekisterien aiheuttamaa muistinkäyttöä, mikä on muun ohjelman kokoon nähden valtava. Kun viimein päästiin vaiheeseen, jossa rekisteri integroitiin itse suoritettavaan ohjelmaan, havaittiin, että ohjelman muistinkäyttö oli kasvanut yli nelinkertaiseksi. Päätös RMI:n käytöstä oli jo kuitenkin tehty ja ohjelman kommunikointi oli tämän mukaan suunniteltu, joten ohjelman tähän versioon ei enää lähdetty kehittämään vaihtoehtoista ratkaisua.

3.7 Ohjelman käyttöliittymät

Ohjelma on suunniteltu ja toteutettu käyttämään client-server-arkkitehtuuria *(C-S-arkkitehtuuri)*. Tämä tarkoittaa, että järjestelmä koostuu kahdesta osasta. Palvelimesta, mikä on tässä tapauksessa varsinainen osoitemuutoksia suorittava palvelu, sekä vierasohjelmasta, mikä on ohjelman käyttöliittymä. C-S-arkkitehtuuri valittiin, jotta ohjelman palveluosio olisi mahdollista suorittaa palveluna palvelimen käynnistyksen yhteydessä. Jos palvelu-osio halutaan käynnistää ilman

sisäänkirjautumista, täytyy ohjelma paketoida Windows-järjestelmissä Win32-kääretiedostoon, mikä voidaan asentaa palveluksi. Linux-järjestelmissä puolestaan voidaan ohjelmalle kirjoittaa omat käynnistys- ja sammutus-skriptit tai yksinkertaisesti lisätä käynnistyskomento *rc.local*-tiedostoon, mikä suoritetaan Linux-järjestelmän käynnistämisen päätteeksi. Graafisen käyttöliittymän eristäminen muusta ohjelmasta mahdollistaa ohjelman käyttämisen järjestelmissä, jotka lataavat graafisen esityksen vaatimat moduulit vain esimerkiksi terminaali-yhteyksien yhteydessä.

Ohjelmalle toteutettu graafinen käyttöliittymä (kuva 11) kommunikoi palvelun kanssa RMI-rajapinnan välityksellä. Sen tehtävä on osoitemuunnosten visualisointi ja hallinta eli uusien sääntöjen lisääminen, vanhojen sääntöjen poistaminen sekä olemassa olevien sääntöjen aktivoiminen ja sammuttaminen.

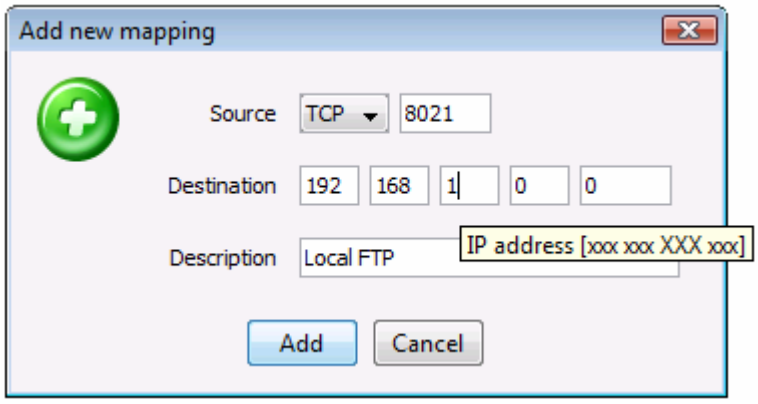


Kuva 11. Graafinen käyttöliittymä

Graafinen käyttöliittymä koostuu kahdesta osasta. Käytettävissä on valikkopalkki, mikä tarjoaa käyttöliittymän perustoiminnot, kuten uusien osoitemuunnoksien lisäämisen ja poistamisen sekä varsinaisen käyttöliittymän hallinnan. Toinen näkyvä osa on taulukko olemassa olevista osoitemuunnoksista. Taulukko kertoo käyttäjälle muunnokseen liittyvät osoitetiedot, siihen liitetyn kuvauksen sekä osoitemuunnos-säännön tilan, joka

voi olla aktiivinen tai passiivinen. Hiiren oikean näppäimen alle on myös toteutettu valikko, jolla voidaan hallita osoittimen alla olevaa osoitemuunnosta.

Osoitemuunnokset haetaan taulukkoon käyttöliittymän käynnistyksen yhteydessä sekä aina jonkin muutoksen aiheuttavan toimenpiteen jälkeen. Taulukko näyttää siis aina palvelun sisäisen tilan, pois lukien tilanteet, joissa aktiivinen osoitemuunnos sammuttaa itsensä vikatilan vuoksi. Tällaisessa tapauksessa paikkansa pitävä informaatio saadaan näkyviin vasta seuraavan päivityksen yhteydessä tai kun käyttöliittymä käynnistetään uudelleen.



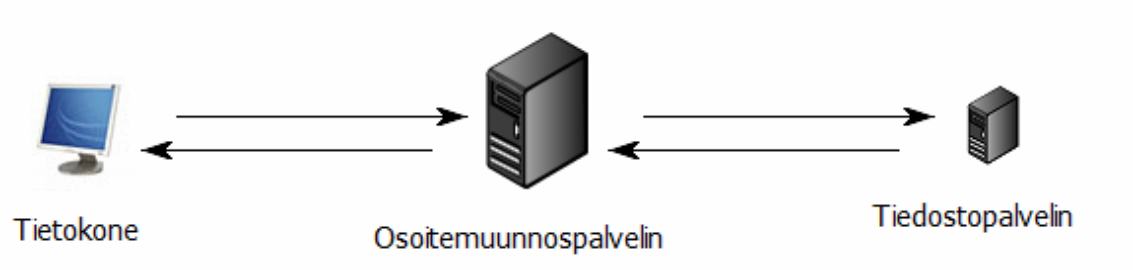
Kuva 12. Osoitemuunnossäännön lisääminen

Osoitemuunnossääntöjen lisäämiselle on toteutettu oma paneeli (kuva 12). Tässä kysytään käyttäjältä kaikki osoitemuunnokseen liittyvät tiedot ja ilmoitetaan, jos osoitemuunnos on laiton tai muuten mahdoton toteuttaa. Muunnokseen liitetty kohdeosoite on annettava neljänä numerosarjana. Tähän olisi haluttaessa mahdollista toteuttaa IP-osoitteen selvittäminen nimipalveluiden avulla, mutta varsinaiselle palvelulle osoite välitetään IP-osoitemuodossa. Ohjelman tämä versio ei sisällä työkaluja nimipalveluiden käyttöön, mutta tuki tälle ominaisuudelle liitetään ohjelmaan mahdollisesti tulevaisuudessa.

Ohjelmalle on tarkoitus toteuttaa myös komentorivikäyttöliittymä *(CLI)*. Tällainen käyttöliittymä on toteutettu ohjelman yksittäisten osien testausta varten, mutta kaikkia ominaisuuksia tukevaa versiota ei ole toteutettu. Täysi komentorivikäyttöliittymä tullaan lisäämään ohjelman ominaisuuksiin, kun ohjelman toteuttamat palvelut vakiintuvat tulevissa versioissa. Ohjelman kehityksen tässä vaiheessa ei komentorivikäyttöliittymää kuitenkaan ole tarjolla.

3.8 Ohjelman testaus

Ohjelman testaus suoritettiin kahdesta näkökulmasta. Ohjelman vakautta testattiin käyttämällä useita palveluita ohjelman läpi pitkällä aikavälillä, mikä oli tässä tapauksessa noin viikon mittainen jakso, jonka aikana ohjelmaa tai palvelinta ei käynnistetty uudelleen. Lisäksi ohjelman resurssien käyttöä mitattiin siirtämällä levykuvia ohjelman läpi ja tarkastelemalla palvelimen suorittimen ja muistinkäyttöä siirron aikana.



Kuva 13. Resurssienkäyttötestin kokoonpano

TCP-yhteyksien käsittelyn resurssien käyttöä testattiin siirtämällä levykuvatiedostoja ohjelman läpi noin sata megabittiä sekunnissa. Testi suoritettiin yhdellä yhteydellä sekä kymmenellä samanaikaisella yhteydellä. Osoitemuunnospalvelimeksi (kuva 13) valittiin tehottomin käytettävissä oleva tietokone, mikä tässä tapauksessa oli kotikäytössä ollut kannettava tietokone (taulukko 2).

Taulukko 2. Osoitemuunnospalvelimen kokoonpano

Tietokone	Dell Latitude E6400
CPU	Intel Core2 Duo P8600 2,40 Ghz
RAM	4,00 Gt
OS	Windows Vista Business SP2 (x86_64)
JRE	Java SE build 1.6.0_07-b06

Testissä käytettiin tiedostopalvelimena (kuva 13) tavallista http-palvelinta, jota ajettiin virtuaalikoneeksi asennetun Linux-järjestelmän päällä. Lataukset asiakastietokoneelle (kuva 13) aloitettiin normaalin selain-ohjelman kautta. Molemmat testit suoritettiin useita kertoja yksittäisten poikkeustapauksien tunnistamiseksi ja saadusta aineistosta valittiin tuloksiksi resurssienkäytön yleistä tilaa parhaiten kuvaavat testikierrokset.

Tuloksista (liite 1) ja (liite 2) voidaan päätellä, että ohjelman resurssien käyttö on vähemmissä määrin riippuvainen yhteyksien määrästä ja suurin työtaakka ohjelmalle aiheutuu varsinaisen datan käsittelystä. Siirrettäessä tietoa noin sata megabittiä sekunnissa yhden yhteyden läpi käytti ohjelma noin yhdeksän prosenttia prosessorin laskentakapasiteetista ja noin 145 kilobittiä tai noin kahdeksantoista kilotavua muistia. Kun sama tiedonsiirtokapasiteetti jaettiin kymmenen yhteyden kesken, nousi prosessorin käyttöaste vain noin kaksi ja puoli prosenttia ja muistinkäyttö lisääntyi noin neljällä kilobitillä, mikä vastaa puolta kilotavua. Kun verrattiin suoraa kommunikointia tiedostopalvelimen kanssa ja kommunikointia osoitemuunnospalvelimen välityksellä, ei havaittu huomattavaa eroa tiedonsiirtonopeuksissa. Tilanne saattaisi muuttua, jos tiedonsiirtonopeutta kasvatettaisiin huomattavasti yli testissä käytetyn siirtonopeuden, mutta tämä tilanne ei kuulu ohjelman aiottuun kohdeympäristöön ja käyttökelpoisuuden rajojen hakeminen jätettiin näin ollen tämän testin ulkopuolelle. Näiden tulosten perusteella voidaan todeta, että TCP-yhteyksien osoitemuunnoksia hoitavan moduulin tekninen toteutus on onnistunut hyvin ja ohjelma kykenee suorittamaan tämän osa-alueen tehtävästään aiotussa kohdeympäristössä.

UDP-pakettien käsittelyssä on otettava huomioon tiedonsiirtonopeuden lisäksi myös käsiteltävien pakettien koko. Internet-protokollaversiossa neljä suurin pakettikoko on

rajoitettu 65 535 tavuun. Tämä tarkoittaa sitä, että suurin mahdollinen UDP-datapaketti voi olla 65 515 tavun mittainen, IP-paketin otsikon ollessa 20 tavua pitkä. Kun UDP-paketin otsikolle varataan tilaa kahdeksan tavua, jää UDP-paketin varsinaiselle sisällölle tilaa maksimissaan 65 507 tavua. Ethernet-verkoissa standardi käytössä oleva MTU *(Maxinum Transmission Unit)* -koko on 1500 tavua, mikä edellä mainitun perusteella jättää varsinaiselle UDP-paketin datalle tilaa 1472 tavua. Huolimatta siitä, että verkkolaitteet osaavat pilkkoa IP-paketit verkolle sopivaan kokoon, käytetään joissakin sovelluksissa kuitenkin pieniä paketteja siinä toivossa, että tällä saavutetaan parempaa suorituskykyä. On myös sovelluksia, joissa suurien pakettikokojen käyttö ei yksinkertaisesti ole järkevää tai mahdollista. Käytettävien pakettien koon valinnan syihin enempää puuttumatta valittiin nämä kaksi arvoa tämän testin ääriarvoiksi. [1, s. 12; 5, s. 1.]

UDP-pakettien käsittelyn testaamiseen käytettiin itse tehtyä yksinkertaista tiedostonsiirto-ohjelmaa, mikä mahdollisti erilaisten pakettikokojen käytön testin aikana. Tämä ominaisuus osoittautuikin varsin paljastavaksi arvioitaessa ohjelman suorituskykyä ja sen käytön rajoitteita. Kun ohjelman suorituskykyä testattiin suurimmalla mahdollisella pakettikoolla, mikä on UDP-datapakettien tapauksessa 65 507 tavua, olivat tulokset (liite 3) hyvin rohkaisevia. Prosessin käyttämä keskimääräinen suorittimen käyttöaste pysyi alle viidessä prosentissa eikä prosessin käyttämässä muistin määrässä tapahtunut merkittäviä muutoksia tiedonsiirron aikana. Kun siirrettävien pakettien kokoa pienennettiin, alkoi myös käsittelystä aiheutuva kuorma kasvaa. Käytettäessä 1472 tavun pakettikokoa oli suorittimen keskimääräinen käyttöaste jo yli kuudenkymmenen prosentin. On siis täysin mahdollista, että tarpeeksi pientä pakettikokoa käytettäessä loppuisi testissä käytetyn tietokoneen suorituskyky kesken. Tulokset osoittavat joka tapauksessa sen, että UDP-pakettien käsittelyn nopeus on riittävä aiottuun kohdeympäristöön ja odotetuille tiedonsiirtonopeuksille, mitkä tuskin tulevat nousemaan yli kymmeneen megabittiin sekuntia kohden.

Ohjelman vakautta testattiin ottamalla ohjelman testiversio käyttöön viikon ajaksi. Palvelinta, jolla ohjelmaa ajettiin tai ohjelmaa itseään, ei käynnistetty uudelleen kertaakaan tämän testin aikana. Kaikki palvelut, joita ohjelman välityksellä käytettiin, toimivat koko testijakson ajan ongelmitta.

4 Jatkokehitysmahdollisuudet

4.1 Muutoksien toteuttaminen

Vaikka ohjelman nykyisen version toteutus täyttää kaikki ohjelmalle asetetut vaatimukset, voidaan ohjelman käytettävyyttä ja tehokkuutta parantaa monin tavoin. Kuten useimpiin kehityskaaren alkuvaiheessa oleviin projekteihin, myös tähän ohjelmaan liittyy ominaisuuksia, joita ei voitu syystä tai toisesta toteuttaa ohjelman nykyiseen versioon. Tämän lisäksi muutamiin ohjelman nykyisen version suunnitteluvaiheessa tehtyihin rakenteeseen ja toteutukseen liittyviin ratkaisuihin on aiheesta kertyneen kokemuksen myötä löytynyt parempia toteutustapoja, jotka voivat mahdollisesti parantaa ohjelman käytettävyyttä ja suorituskykyä. Näiden ominaisuuksien toteuttaminen vaatii kuitenkin lähes poikkeuksetta suuria muutoksia ohjelman perusrakenteeseen, joten niitä ei voitu ohjelman toteutusvaiheen aloittamisen jälkeen enää liittää ohjelman tähän versioon. Ohjelman kehitys jatkuu kuitenkin tulevissa versioissa, joihin nykyisestä versiosta puuttuvia ominaisuuksia ja ohjelman osien uusia ja paranneltuja toteutuksia voidaan liittää.

Ohjelman rakenne helpottaa jatkokehitystä huomattavasti. Koska ohjelman eri osa-alueiden toiminta ei ole varsinaisesti riippuvaista muiden osa-alueiden sisäisestä toteutuksesta, voidaan uudet toteutukset integroida ohjelmaan ja testata varsin helposti. Jos yksi osa vaihdetaan uuteen ja tämän jälkeen ilmenee uusia ongelmia, voidaan olla kohtuullisen varmoja siitä, että ongelma on uuden osan toteutuksessa, sillä vanha koodi on jo todettu toimivaksi. Kontrolloimalla uusien osien integrointia tähän tapaan vältytään tilanteelta, missä kaikki ohjelman osat täytyy testata uudelleen, kun ei tiedetä, minkä osan toteutus aiheuttaa varsinaisen ongelman.

4.2 Asetuksien eristäminen muusta lähdekoodista

Ohjelman nykyisessä toteutuksessa kaikki ohjelman suorittamiseen liittyvät asetukset on kovakoodattu lähdekoodin sisään ja ympäri eri moduuleita. Tämä vaikeuttaa huomattavasti tiettyjen muutosten tekemistä tulevaisuudessa sekä erilaisiin järjestelmiin asentamista. Esimerkiksi Windows- ja Linux-järjestelmät käyttäytyvät polkujen käsittelyn osalta hyvin erilailla. Polku, joka osoittaa Windows-järjestelmässä ohjelman ajokansioon, voi Linux-järjestelmässä osoittaa järjestelmän juurihakemistoon. Ohjelman tulevassa versiossa on tarkoitus korjata tämä tilanne toteuttamalla oma luokka asetuksien ajonaikaista ylläpitoa varten ja toteuttamalla asetusten tallentaminen tiedostoon tai tietokantaan. Tämä mahdollistaa asetusten sovittamisen eri järjestelmiin, mikä parantaa ohjelman käytettävyyttä ja yhteensopivuutta erilaisten järjestelmien kanssa. Ulkoisesti tallennetut asetukset yksinkertaistavat myös varmuuskopioiden tekemistä ja asetusten siirtämistä järjestelmien välillä.

Toteutuksen tarkoitus on lukea asetustiedot ohjelman käynnistyksen yhteydessä, ja pitää yllä sen sisältöä ohjelman suorituksen aikana. Kaikki ohjelman ajamiseen liittyvät muuttujat kuten tiedostojen nimet ja polkumääritykset on tarkoitus pitää tässä luokassa ja jakaa muulle ohjelmalle tätä kautta. Ohjelman palveluosiolle on tietenkin pystyttävä generoimaan asetukset joko käyttöliittymästä käsin tai erillisellä apuohjelmalla. Tämä vaatimus puoltaa tietokannan käyttöä asetusten tallentamiseen, koska tietokantaa käytettäessä ei tarvitse ylläpitää useita asetustiedostoja ja asetusten hallinta on mahdollisesti yksinkertaisempaa keskittää käyttöliittymään. Sekä käyttöliittymä että palvelu vaativat kuitenkin vähintään tiedon tietokannan nimestä ja sijainnista käynnistämisen yhteydessä, joten asetustiedostoista ei täysin päästä tässäkään tapauksessa eroon uhraamatta joustavuutta, mitä asetusten eristämisellä nimenomaan pyritään luomaan.

Koska asetusten käsittely käyttöliittymästä käsin vaatii tietyissä tapauksissa käyttöliittymän käynnistämistä, vaikka palvelu ei käynnissä olisikaan, on tämän muutoksen tekeminen sidottu RMI-rajapinnan poistamiseen. Tarvittava toiminta voitaisiin toteuttaa myös RMI:n avulla, mutta RMI:n toiminnan kehittäminen ei ole tässä vaiheessa enää perusteltua, koska sen tarjoamaa toiminnallisuutta ollaan korvaamassa toisella toteutuksella.

4.3 Käytettävyyden parantaminen

Vaikka nykyinen käyttöliittymä on toimiva, ei se ole täysin käyttäjäystävällinen, koska ohjelman käyttäjän on tunnettava palvelimien IP-osoitteet uusia osoitemuunnos-sääntöjä määriteltäessä. Osoitemuunnoksien lisäyslomakkeeseen onkin suunnitteilla uusi ominaisuus, mikä sallii IP-osoitteiden etsimisen nimipalveluiden avulla. Nimipalveluiden käyttöä ei kuitenkaan tulla lisäämään ohjelman palveluosion pakettienkäsittelyyn, sillä tämä synnyttäisi lisää viivettä uusien TCP-yhteyksien avaamisessa ja aiheuttaisi tätäkin enemmän viivettä UDP-datapakettien käsittelyssä, sillä jokaisen paketin kohdalla pitäisi kommunikoida nimipalvelimen kanssa eikä tämä ole ohjelman käyttötarkoituksen huomioon ottaen tarkoituksenmukaista. Tämän ominaisuuden käyttämiseksi on reitittimenä toimivan palvelimen oltava tietoinen yksityisen verkon nimipalvelimista, mikä lisää ohjelman yhteensopivuudelle asetettuja vaatimuksia. Osoitehaun täytyykin sallia myös tavallisen IP-osoitteen käyttäminen, jotta nimipalveluiden toteuttamatta jättämisestä ei tule rajoittavaa tekijää ohjelman käytölle.

Ohjelman käyttöliittymää ei myöskään ole suojattu millään tavalla, minkä vuoksi kuka tahansa voi käydä muuttamassa osoitemuunnoksia. Ohjelman tulevat versiot on tarkoitus suojata vähintään salasanalla, minkä kyselyn voi kuitenkin asettaa halutessaan pois käytöstä. Erilaisten käyttäjätasojen toteuttaminen olisi myös mahdollista, mutta tälle ei ainakaan tässä vaiheessa ole varsinaista tarvetta.

4.4 RMI-rajapinna korvaaminen tietokannalla

Ohjelman suurin muistin käyttäjä on RMI-rekisteri, mikä toimii ohjelman nykyisessä versiossa rajapintana palveluosion ja käyttöliittymän välillä. RMI tarjoaa kaikki tarvittavat toiminnot, mutta ratkaisu on osoittautunut kuitenkin liian raskaaksi lopun ohjelman kokoon nähden. RMI-palvelu vaatii myös kaksi TCP-pistoketta toimiakseen, mikä kuluttaa varsinaiseen osoitemuunnokseen käytettävissä olevia pistokkeita. Lisäksi RMI-rajapinta avaa mahdollisen tietoturvariskin järjestelmään. Koska palvelua ajetaan järjestelmässä joka on yhteydessä julkiseen verkkoon, on kenen tahansa mahdollista muuttaa ohjelman asetuksia. RMI-rajapinta on suunniteltu poistettavaksi ohjelman seuraavassa versiossa ja korvattavaksi kevyellä tietokantatoteutuksella.

Tämän muutoksen tavoitteena on toteuttaa osoitemuunnossääntöjen tuominen käyttöliittymältä palvelulle huomattavasti pienemmällä muistinkäytöllä ja ilman TCP- tai UDP-pistokkeiden käyttöä. Lokaalin tietokannan käyttäminen mahdollistaa myös palvelun sisäisen osoitemuunnossääntöjä ylläpitävän tietorakenteen sisällön rajoittamisen ainoastaan aktiiviseen tilaan asetettuihin osoitemuunnossääntöihin, mikä pienentää edelleen ohjelman palveluosion muistijälkeä. Toisaalta ohjelman palveluosiolle on toteutettava uusi moduuli, mikä tarkastelee tietokantaa aktiivisesti ja pitää yllä palvelun sisäisen tietorakenteen eheyttä. Tämä puolestaan vaatii hiukan muistia ja prosessorin aikaa. Tästä muutoksesta on kuitenkin suurella todennäköisyydellä huomattavasti enemmän etua kuin haittaa.

4.5 Kuormanjako-ominaisuuden toteuttaminen

Ohjelman nykyinen versio osaa tehdä muunnoksen vain yhteen kohdeosoitteeseen. Joskus on kuitenkin suotavaa jakaa palvelulle tulevaa kuormaa useampien palvelimien kesken. Koska ohjelman nykyinen rakenne käsittelee jo tiedonsiirtoa pistokkeiden välillä, on kohtuullisen yksinkertaista lisätä enemmän kuin yksi kohde jokaiselle kuuntelijalle. Jotta kuormaa saadaan jaettua palvelimien välillä, on kohteita kierrätettävä samaan tahtiin kuin uusia yhteyksiä avataan. Yksinkertaisin tapa olisi ohjata uudet yhteydet vuorotellen kaikille kohteille, mikä saadaan toteutettua kierrättämällä kohteita FIFO -tyyppisessä listassa. Tämä ominaisuus on kuitenkin varsin

lähellä toteutettavien muutosten listan loppupäätä, sillä vaikka ominaisuus lisää jossain määrin ohjelman käyttömahdollisuuksia, on ohjelmassa myös sellaisia ominaisuuksia, jotka täytyy korjata ennen uusien ominaisuuksien toteuttamista.

5 Yhteenveto

Työn tavoitteena oli suunnitella ja tuottaa ohjelma, joka mahdollistaa kommunikoinnin kahden verkon välillä tapauksissa, joissa reititystä ei voida järjestää molempiin suuntiin. Tällainen tilanne on esimerkiksi yksityisen IP-verkon liittäminen julkiseen IP-verkkoon. Ohjelman oli tarkoitus pystyä käsittelemään sekä TCP- että UDP-protokollien välityksellä siirrettävää dataa ja ohjelmaa piti pystyä ajamaan sekä Windows- että Linux-järjestelmissä.

Toteutettu ohjelma täyttää kaikki nämä vaatimukset, joten toteutuksen osalta projektia on pidettävä onnistuneena. Ohjelmaan toteutetuille ratkaisuille on kuitenkin olemassa vaihtoehtoisia ratkaisuja, joita ei kaikkia osattu harkita vielä ohjelman suunnitteluvaiheessa. Lisäksi ohjelman määrittely jätti monia seikkoja suunnittelijan mielikuvituksen varaan, eikä se näin ollen ollut tarpeeksi kattava. Vaikka ohjelman nykyinen toteutus täyttää kaikki ohjelmalle asetetut vaatimukset, havaittiin ohjelman toteutuksen yhteydessä niin suuria parannuskohteita, että ohjelman määrittely ja suunnitteluosuuksia ei voida pitää täysin onnistuneina.

Ottaen huomioon projektin lopputuloksena saadun toimivan ohjelman on projektia kokonaisuudessaan pidettävä onnistuneena. Ohjelmasta löydettyjä heikkouksia voidaan käsitellä tulevien versioiden määrittelyn ja suunnittelun yhteydessä. Uuden paremman version kehittämisen aikana voidaan käyttää nykyistä toimivaa ohjelmaa.

Lähteet

1 Postel, Jon. 1981. Internet Protocol. Verkkodokumentti. <www.ietf.org/rfc/rfc791>. Luettu 21.10.2010.

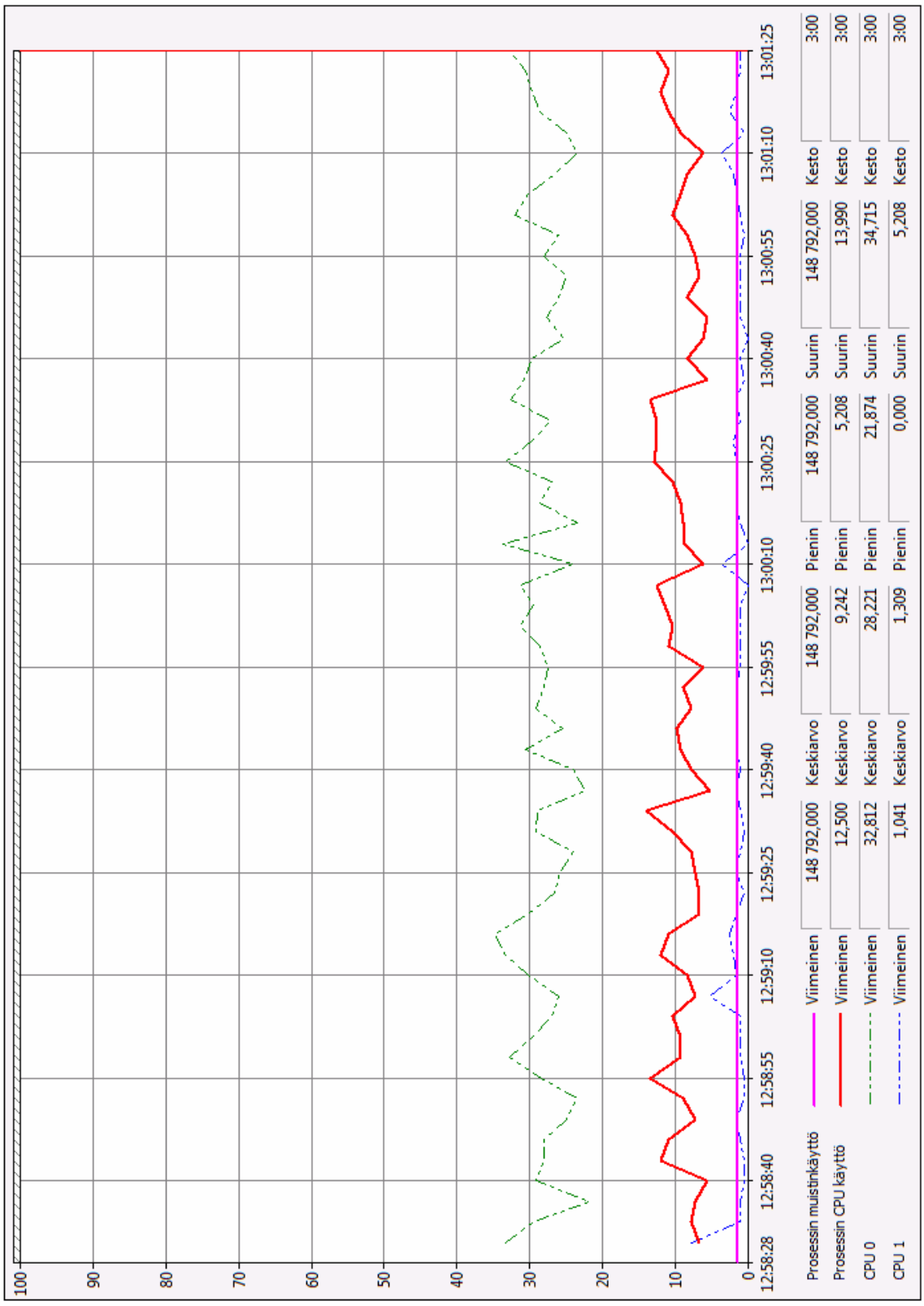
2 Address Allocation for Private Internets. 1996. Verkkodokumentti. The Internet Engineering Task Force. <www.ietf.orf/rfc/rfc1918>. Luettu 25.10.2010.

3 Traditional IP Network Address Translator. 2001. Verkkodokumentti. The Internet Engineering Task Force. <www.ietf.org/rfc/rfc3022>. Luettu 26.10.2010.

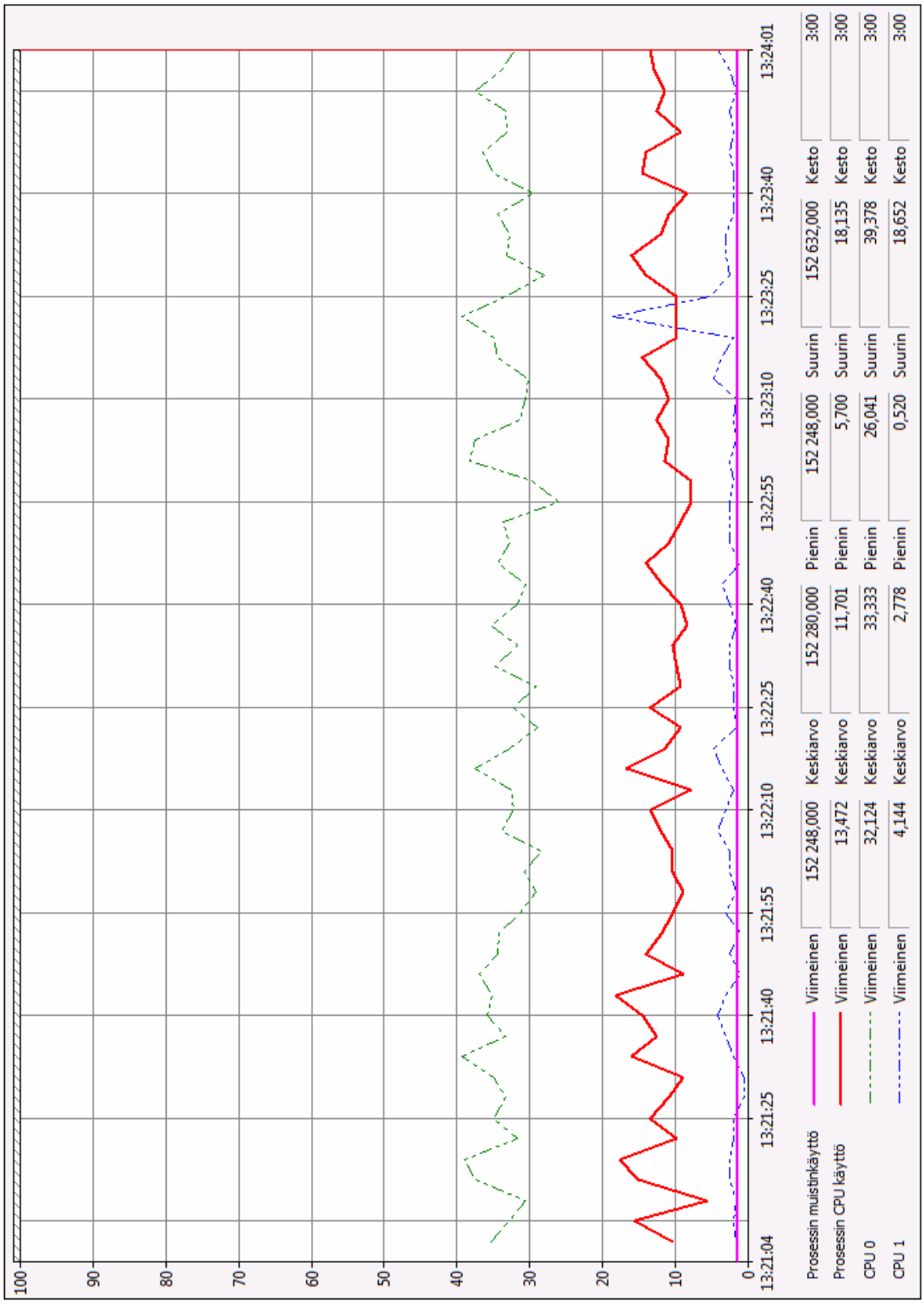
4 Postel, Jon. 1981. Transmission Control Protocol. Verkkodokumentti. <www.ietf.org/rfc/rfc793>. Luettu 4.11.2010.

5 Postel, Jon. 1980. User Datagram Protocol. Verkkodokumentti. <www.ietf.org/rfc/rfc768>. Luettu 13.11.2010.

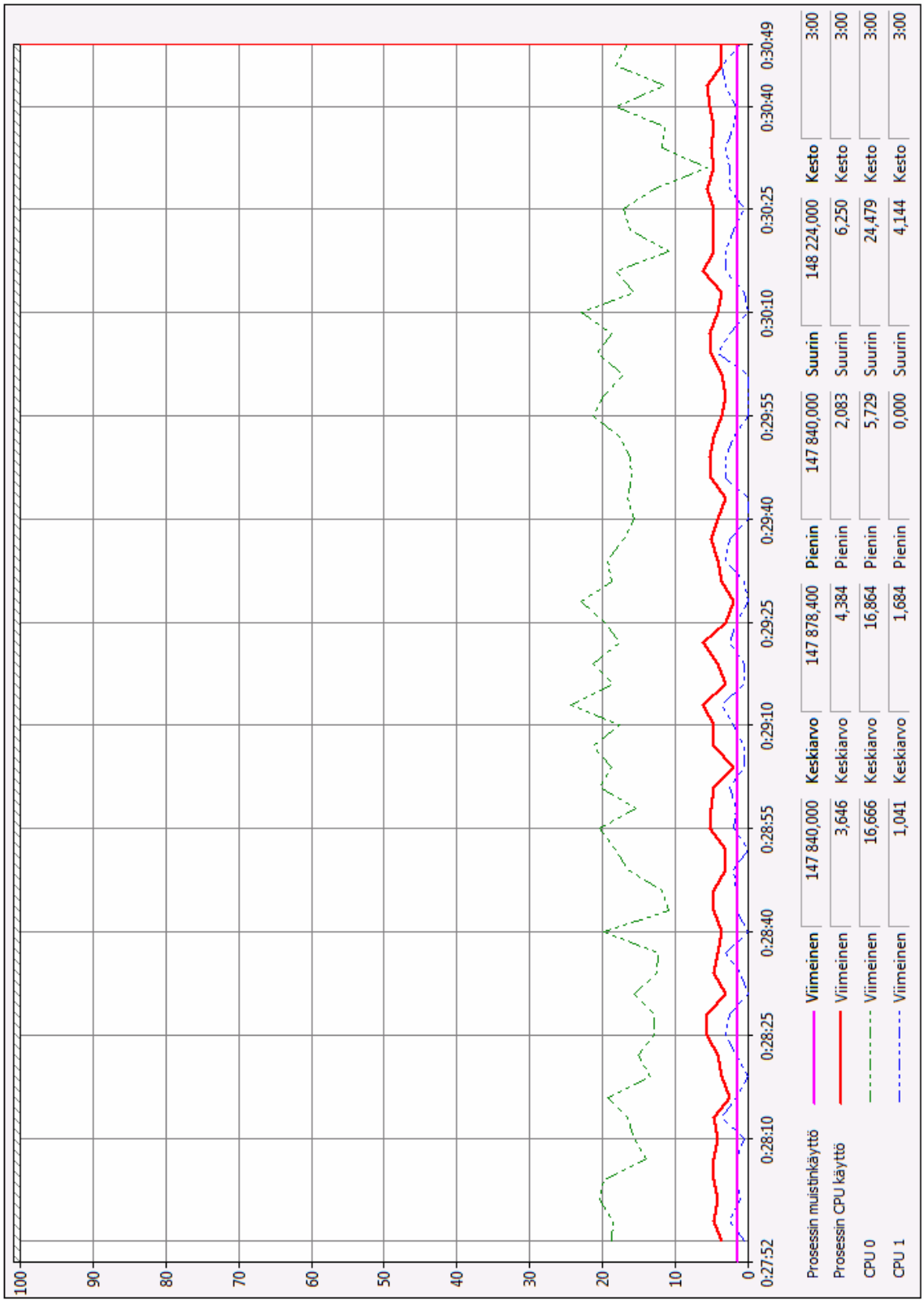
Yhden TCP-yhteyden käsittely



Kymmenen TCP-yhteyden käsittely



UDP-datapakettien käsittely



Synkronoitutietorakenne

```
import java.util.LinkedList;
import java.util.Queue;

public class SyncQueue {

        private Queue<Object> buffer;
        private boolean hasContent = false;

        public SyncQueue() {
                this.buffer = new
                LinkedList<Object>();
        }

        public synchronized void offer(Object object)
                throws        InterruptedException {
                while (hasContent) {
                    try { wait(); }
                    catch (InterruptedException e) {}
                }
                buffer.offer(object);
                hasContent = true;
                notify();
        }

        public synchronized Object poll() throws
                InterruptedException {
                while (!hasContent) {
                    try { wait(); }
                    catch (InterruptedException e) {}
                }
                Object object = buffer.poll();
                if(buffer.size() == 0){
                    hasContent = false; }
                notify();
                return object;
        }

}
```