TEKNILLINEN KORKEAKOULU

Tietotekniikan osasto

Markus Peuhkuri

Prosessien vuorottelun vaikutus dataliikenteeseen

Diplomityö, joka on jätetty opinnäytteenä tarkistettavaksi diplomi-insinöörin tutkintoa varten Espoossa 23. toukokuuta 1997.

Työn valvoja: Professori Jorma Virtamo, TKK

Työn ohjaaja: DI Marko Luoma, TKK

Tekijä: Markus Peuhkuri

Työn nimi: Prosessien vuorottelun vaikutus dataliikenteeseen

Päivämäärä: 23. toukokuuta 1997

Sivumäärä: 73

Osasto: Tietotekniikan osasto

Professuuri: S-38 Teletekniikka

Työn valvoja: Professori Jorma Virtamo

Työn ohjaaja: DI Marko Luoma

Tässä työssä tarkastellaan prosessien vuorottelun vaikutusta dataliikenteeseen. Dataliikenne on perusluonteeltaan purskeista muodostuen vuorottelevista täysinopeuksisesta siirtojaksoista ja tauoista näiden välillä. Useiden kilpailevien prosessien yht'aikainen suoritus kasvattaa taukojen pituuksia tehden yhden yhteyden liikenteestä enemmän purskeista, mikä asettaa suuremmet vaatimukset verkolle.

Työssä kehitettiin joukko työkaluja, joilla voitiin tutkia liikennettä synnyttävää prosessia. Käyttämällä muokattua Linux-käyttöjärjestelmää, mittausohjelmaa ja verkkoliikennetallenninta kerättiin tuloksia useista eri liikennöintitapauksista. Verkkon ja käyttöjärjestelmän välistä vuorovaikutusta voidaan tutkia synkronoimalla ja yhdistämällä eri lähteistä peräisin oleva tieto

Liikennettä karakterisointiin eri menetelmillä, muun muassa ATM-verkoissa käytettävillä liikenneparametreilla. Linux-käyttöjärjestelmä antaa saman palvelun sekä CPU- että I/O-painotteisille prosesseille: pullonkaularesurssi (verkkoliikenne) voi olla suuriakin aikoja käyttämättä. Puskurointia on lisättävä tai prosessien vuorottelua on muokattava, jotta liikenteestä saadaan tasaista. Eri sijoitusvaihtoehtoja prosessien vuorottelua ohjaavalle mekanismille arvioitiin.

Avainsanat: käyttöjärjestelmät, liikenteen hallinta, liikennemittaukset

Author: Markus Peuhkuri

Title of the thesis: Effects of process scheduling to datatraffic

Date: 23 May 1997

Number of pages: 73

Faculty: Information Technology

Professuuri: S-38 Telecommunications Technology

Supervisor: Professor Jorma Virtamo

Instructor: M.Sc. Marko Luoma

In this work is studied effects of scheduling to datatraffic. The nature of the datatraffic is bursty: it contains full-speed transfer periods and pauses between those. Increasing the number of competing processes increases the pauses also. The bursts make high requirements to network.

A toolbox was developed to get better understanding of the processes generating traffic to the network. Using an instrumented Linux kernel, a benchmark program and network traffic capture several traces was generated. The traces include network traffic with timestamps, timestamps of system calls from benchmark program and information of process and kernel status at process switch. By synchronizing and combining the traces from different sources one could understand operating system — network interaction.

The traffic was characterized by several methods, among other by the usage parameter control used in ATM networks. Linux operating system gives the same service for both CPU and I/O intensive traffic: the bottleneck resource (network) can be unutilized for long periods. To smooth traffic, one must either add buffering or modify process scheduling. Several alternatives for scheduling control mechanism were estimated.

Keywords: operating systems, traffic management, traffic measuremets

Alkulause

Tämä työ on tehty Teknillisen korkeakoulun Teletekniikan laboratoriossa Suomen Akatemian rahoittamassa Mitta-projektissa. Alkuosa työstä tehtiin Tekesin rahoittamassa Faster-projektissa.

Kaikkien työhön vaikuttaneiden henkilöiden luetteleminen lienee mahdotonta ja rajanveto vaikeaa, mutta joka tapauksessa: Kiitos.

Työn valvojaa, professori Jorma Virtamoa, haluan kiittää lukuisista parannusehdotuksista, jotka selkeyttivät työn rakennetta ja tutkimuksen esittämistä.

Työn häiriöttömästä ohjauksesta haluan kiittää DI Marko Luomaa. Työ löysi aihepiirinsä, joka oli itselleni kiinnostava. TkT Kalevi Kilkki oli suureksi avuksi tutkimuksen alkuvaiheessa ongelmakentän määrittämisessä.

Teletekniikan laboratorion henkilökuntaa haluan kiittää väsymättömästä panoksesta, joka esti minua uppoutumasta liian syvälle tutkimukseen. Pelastava kysymys tietokoneiden tai ohjelmien toiminnasta tuli usein aivan viime hetkellä. Erilaista keskusteluista oli jopa hyötyä työn kannalta. Työn aloittaminen laboratoriossa katkaisi aikoinaan hyvin alkaneet opintoni. Todennäköisesti muualla olisin valmistunut jo huomattavasti aiemmin, mutta jotain on jäänyt myöskin saamapuolelle. Laboratorion emeritus professorin, Kauko Rahkon, ansiosta tietämykseni teletekniikasta on laajempi kuin mitä se muutoin ehkä olisi.

Teitä elämäni sulostuksia, Katria ja Saaraa, haluan kiittää, että olette olemassa. Selvisitte hienosti loppurutistuksesta ja tuitte minua vaikka ”iso paha susi pitää isiä vankina töissä”. Jumalalle kiitos kaikesta.

Sisältö

Käytetyt lyhenteet ja termit

Tämä luettelo ei sisällä kaikkia käytettyjä lyhenteitä vaan osa lyhteistä on esitetty seuraavissa tekstissä olevissa taulukoissa:

TCP:n tilamuuttujat Taulukot 2.2 (s. 21) ja 2.3 (s. 22).

ATM-verkon liikenneparametrit Taulukko 2.4 (s. 30).

BSD prioriteettitasot Taulukko 3.1 (s. 34).

Lyhenteet ja termit

10Base2 ANSI/IEEE 802.3 50 Ω:n kaapelille (n. 0,5 cm paksulle), ”ohutethernet” [ANS96].

10Base5 ANSI/IEEE 802.3 50 Ω:n kaapelille (n. 1 cm paksulle), ”paksuethernet” [ANS96].

10BaseF ANSI/IEEE 802.3 valokuidulle [ANS96].

10BaseT ANSI/IEEE 802.3 kierretylle parikaapelille [ANS96].

4.4BSD Berkeley Software Distribution’n kehittämä käyttöjärjestelmä.

AAL3/4 ATM Sovituskerros 3/4. Vaihtelevanopeuksiselle yhteydelliselle sekä yhteydettömälle ei-realiaikaliikenteelle.

AAL5 ATM Sovituskerros 5. Vaihtelevanopeuksiselle yhteydelliselle sekä yhteydettömälle ei-realiaikaliikenteelle; yksinkertaisempi kuin AAL3/4.

ACK (TCP:ssä) Lippu, joka osoittaa kuittaussarjanumerokentässä olevan arvon olevan käytössä,

ALU Arithmetic Logic Unit. CPU:ssa laskennasta vastaava osa.

API Application Programming Interface. Sovellusten ohjelmointiin (käyttöjärjestelmän) tarjoama liitäntä, yleensä kokoelma funktioita.

ARPA Advanced Research Projects Agency. Yhdysvaltain puolustusministeriön alainen tutkimuslaitos.

ATM Tahdistamaton siirtomuoto (Asynchronous Transfer Mode).

AUI Liitäntä trancieverin ja verkkokortin välillä [ANS96].

Berkeley Software Distribution University of California at Berkeley'n ohjelmistojakelu; tunnetuin on BSD-UNIX, joka on Computer System Research Group'n kehittämä.

BSD kts. Berkeley Software Distribution.

BSD Socket Yleisimmin verkkoliikennöintiin käytetty ohjelmointiraja.

CPU Central Prosessing Unit. (Tietokoneen) keskusyksikkö.

DMA Direct Memory Access. Suora muistiinosoitus, jota eräät laiteohjaimet käyttävät. Vapauttaa CPU:n datasiirrosta.

DNS Domain Name System. Järjestelmä, joka määrittää Internetissä koneen nimestä sen IP-osoitteen [Moc87a, Moc87b].

DoD Department of Defence. Yhdysvaltain puolustusministeriö, jonka alaisuudessa toimii mm. ARPA.

emolevy Tietokoneen osa, joka sisältää yleensä CPU:n, keskusmuistin ja oheisväylät.

Ethernet Käytetään yleisnimityksenä IEEE 802.3 -pohjaisille verkoille. Alunperin Digitalin, Intelin ja Xeroxin kehittämä standardi, jota hiukan muuttamalla IEEE 802.3 -standardi on luotu. Xeroxin rekisteröimä tavaramerkki.

FCFS First Come, First Served. Vuorottelualgoritmi "ensin tullut, ensin palveltu".

FFS Fast File System. 4.4BSD:ssä yleisimmin käytetty tiedostojärjestelmä.

FIFO First In First Out. Palvelujärjestys on saapumisjärjestyksen mukainen; jono.

FIN (TCP) Lippu, joka ilmoittaa ettei sen lähettäjällä ole enempää lähetettävää.

FreeBSD Eräs[1] tuotteistus 4.4BSD:stä.

FTP File Transfer Protocol. Tiedonsiirtostandardi TCP/IP-verkoissa [PR85]. Käytetään yleisesti myös k.o. protokollaa käyttävästä ohjelmasta.

GCRA Generic Cell Rate Algorithm. ATM-verkoissa käytetty liikenteenvalvonta-algoritmi (kappale 2.5.1, s. 29).

HTML HyperText Markup Language. SGML-pohjainen kieli hypertekstidokumenttien esittämiseen [BLC95].

HTTP HyperText Transfer Protocol. Standardi (hyperteksti)dokumenttien siirtämiseen [BLFF96, FGM+97].

I/O Input/Output. Syöttö- ja tulostustoiminnot.

ICMP kts. Internet Control Message Protocol.

IDC kts. saapumisien lukumäärän hajontaindeksi.

IDE Integrated Drive Electronics. PC-koneissa käytetty kiintolevyjen liitäntästandardi.

IEEE 802.3 Kilpavarausväylä törmäyksen tunnistuksella [ANS96].

IETF Internet Engineerin Task Force. Vastaa Internetin teknologian kehityksestä ja standardisoinnista.[2]

IGMP kts. Internet Group Management Protocol.

Index of Dispersion for Counts kts. saapumisien lukumäärän hajontaindeksi.

inetd Internet services daemon. Toimii edustaohjelmana useille palveluille; käynnistää tarvittavat ohjelmat tarvittaessa. Säästää keskusmuistia, koska varsinaisia palvelinohjelmia ei tarvitse koko aikaa pitää muistissa.

inode Information node. Tietorakenne, joka 4.4BSD:n levyjärjestelmässä pitää kirjaa tiedoston ominaisuuksista, mm. tiedoston levylohkoista.

[1] http://www.freebsd.org/
[2] http://www.ietf.org

Internet Control Message Protocol IP:n apuprotokolla, jota käytetään virheiden ilmaisuun ja diagnostiikkaan [Pos81a].

Internet Group Management Protocol IP:n apuprotokolla, jota käytetään jakeluryhmien hallintaan [Dee89].

Internet Protocol Yleisimmin Internetissä käytetty verkkokerroksen protokolla [Pos81b, MP85, Mog84a, Mog84b, Pos81a, Dee89].

IP kts. Internet Protocol.

ISO International Standards Organization. Kansainvälinen standardoimisorganisaatio.[3]

kiireodotus Prosessi odottaa tapahtumaa tarkistamalla jonkun tilaa jatkuvasti. Tämä on moniajojärjestelmissä tehotonta, koska tällöin prosessi vie suoritusaikaa muilta prosesseilta.

käyttäjätila Prosessorin tila, jossa normaalit käyttäjäohjelmat suoritetaan. Oikeudet esimerkiksi muistin ja oheislaitteiden suhteen on rajoitettu, vrt. ydintila.

LFS Log-structured File System. 4.4BSD:ssä käytetty vikasietoinen tiedostojärjestelmä.

liikennöintipiste Käyttöjärjestelmän tietorakenne, jonka avulla sovellus voi kommunikoida toisten sovellusten kanssa. Englanniksi ”socket”, slangiterminä ”töpseli”.

Linux Suomalaisen Linus Torvaldsin alunperin kehittämä POSIX-yhteensopiva käyttöjärjestelmä.[4]

LRU Least Recently Used. Välimuisteissa käytetty korvausalgoritmi, jossa korvataan pisimpään käyttämättä ollut alkio.

lukkiuma Tilanne, jossa suoritus ei pääse etenemään, koska joukon kaikki prosessit odottavat jotain resurssia, joka on toisella joukon prosessilla varattuna.

[3] http://www.iso.ch/
[4] http://www.linux.org/

MAC Media Access Control. Lähiverkoissa OSI:n siirtoyhteyskerrokselle kuuluva osa, joka määrittelee verkkoon pääsyn (vuoronvarauksen).

MAU Media Access Unit. Verkkokortin fyysinen liitäntä verkkoon, joka on nykyään yleensä integroitu verkkokortille, mutta on 10Base5:ssä erillinen osa.

Mbit/s Siirtonopeuden yksikkö. Yleensä $1\ 000\ 000 = 1000 \times 1000$ bittiä sekunnissa mutta voi olla myös $1\ 048\ 576 = 1024 \times 1024$ bittiä sekunnissa.

MBS Maximum Burst Size. Suurin purskekoko ($MBS = BT/PCR$).

mbuf 4.4BSD-pohjaisissa järjestelmissä verkkotoiminnoissa käytetty tietorakenne, joka mahdollistaa tiedon liittämisen ja poistamisen sekä alusta että lopusta.

MFS Memory-based File System. 4.4BSD:ssä väliaikaistiedostoille käytetty tiedostojärjestelmä, joka hyödyntää virtuaalimuistia.

muokkain ATM-verkkokortilla oleva osa, joka rajoittaa tarvittaessa soluvirran lähetysnopeutta, jotta se täyttäisi liikennesopimuksen.

muuttujaosa Prosessin muistiavaruudessa oleva osa, joka on varattu pysyville muuttujille.

saapumisien lukumäärän hajontaindeksi Lukuarvo, joka kuvaa liikenteen vaihteluja aikajaksojen välillä (kappale 2.5.2, s. 31).

NFS Network File System. Sun Microsystemssin[5] kehittämä verkkolevyjärjestelmä.

$nice$ Käyttäjän aseteltavissa prosessin kohteliaisuusarvo, joka vaikuttaa prosessin prioriteettiin.

ohjelmalaskuri CPU:ssa oleva rekisteri, joka ilmoittaa seuraavaksi suoritettavan käskyn muistiosoitteen.

OSI Open Systems Interconnection. ISO:n 1970-luvun loppupuolella kehittämä viitemalli, jonka tarkoitus on selkeyttää tietoliikenteeseen liittyviä käsitteitä ja helpottaa standardien luomista [ISO94].

[5] http://www.sun.com/

palvelun esto Vihamielinen hyökkäys järjestelmää vastaan, jossa tarkoituksena on estää järjestelmää suorittamasta normaaleja palvelutehtäviä.

palvelun laatu Palvelun laatua voidaan arvioida monilla eri tasoilla alkaen tiedonsiirron viiveestä ja bittivirhesuhteesta aina käyttäjän subjektiiviseen kokemukseen [IT93c].

PC Tässä esityksessä käytetty Intel 8086 -yhteensopivaa prosessoria käyttävästä, MS-DOS -käyttöjärjestelmän ajamiseen kykenevästä koneesta.

Pentium Intelin[6] kehittämä, PC-koneissa käytettävä prosessori.

Pentium Pro Pentium-prosessorista kehitetty versio, joka toimii 32-bittisillä käyttöjärjestelmillä nopeammin.

perl Practical Extraction and Report Language. Ajonaikaisesti käännettävä C:n kaltainen kieli, joka on erityisen tehokas tekstitiedostojen käsittelyyn.

PID Process IDentifier. Prosessitunniste tietokoneissa.

pino Tietorakenne, josta viimeksi talletettu saadaan ensimmäisenä. Prosessin osa, johon yleensä tallennetaan aliohjelmien paikalliset, tilapäiset muuttujat ja paluuosoitteet.

PIO Programmed I/O. Tiedonsiirto hoidetaan prosessorilla, vrt. DMA.

POSIX Portable Operating System for unIX. Määrittelykokonaisuus, jonka pyrkimyksenä on saada eri UNIX-murteet käyttäjän ja ohjelmoijan kannalta samankaltaisiksi.

PSH (TCP:ssä) Lippu, joka ilmoittaa, että tieto on toimitettava välittömästi sovellukselle eikä puskuroitava.

puiminen Tila, jossa tietokone käyttää kaiken aikansa prosessien osien siirtämiseen levymuistin ja keskusmuistin välillä, koska käytössä oleva keskusmuisti ei riitä prosessien yhtäaikaiseen suorittamiseen.

putki (UNIX:ssa) Prosessien väliseen kommunikointiin käytetty menetelmä ja tietorakenne.

[6] http://www.intel.com/

QoS kts. palvelun laatu.

RedHat Eräs Linuxin paketointiin erikoistunut yritys.[7]

RR Round Robin. Kiertävän vuoron vuorottelualgoritmi.

RST (TCP:ssä) Lippu, joka ilmoittaa yhteyden katkaisemisesta tai hylkäämisestä esimerkiksi virheellisen sarjanumeron perusteella.

RTT Round Trip Time. Aika, joka kuluu kuittauksen saamiseen lähetetystä viestistä.

segmentti (1) Lähiverkon osa, jossa ei ole aktiivisia verkkokomponentteja välissä ts. sähköisesti yhtä. (2) TCP-yhteyden osa, joka kuljetetaan yhdessä IP-sähkeessä.

SGML Standard Generalized Markup Language. Tekstitiedon esitystapa, jolla dokumentin looginen rakenne erotetaan fyysisestä ulkoasusta. Suosittu alunperin teknisessä dokumentaatiossa mutta nykyisin suurin käyttö on HTML, joka on osajoukko SGML:stä. Standardi ISO 8879 [ISO86].

SJF Shortest Job First. Vuorottelu algoritmi, joka valitsee lyhyimpään kestävän työn ensin.

SMTP Simple Mail Transfer Protocol. Sähköpostin siirtoprotokolla Internetissä [Pos82].

SNMP Simple Network Management Protocol. Verkonhallintaprotokolla [SFDC90].

SPEC Järjestö, joka kehittää menetelmiä tietokonejärjestelmien suorituskyvyn mittaukseen ja julkistaa niiden tuloksia.[8]

STM-1 Synchronous Transfer Mode 1. Linkkikerroksen tiedonsiirtotapa, jossa siirtonopeus on 155 Mbit/s.

SunOS Sun Microsystems'n[9] käyttöjärjestelmä.

superlohko Levyn lohko, jonka avulla pystytään saamaan tarvittava tieto levystä ja tiedostojen sijoittelusta. Nykyisin usein toistettu, jotta yksittäinen levyvaurio ei tuhoaisi koko levyn tietoja.

[7] http://www.redhat.com/
[8] http://www.specbench.org/
[9] http://www.sun.com/

SYN (TCP:ssä) Lippu, joka ilmoittaa yhteyden luomisesta.

TCP kts. Transmission Control Protocol.

TCP/IP Käytetään yleisnimikkeenä tarkoitettaessa IP-protokollan toimivien protokollien perhettä.

Telnet Virtuaalipääteprotokolla verkon yli [PR83b, PR83a].

TPC Transaction Processing performance Council.[10] Järjestö, joka kehittää tietokantajärjestelmien suorituskykymittareita ja julkistaa tuloksia.

tranciever Ethernet-kortin AUI-liitäntään liitettävä verkkoliitäntä (MAU). Yleensä käytetään 10Base5:n ja 10BaseF:n kanssa.

Transmission Control Protocol Internetin siirtokerroksen protokolla, joka tarjoaa kaksisuuntaisen virheenkorjaavan virtuaaliyhteyden [Pos81c].

työala Muisti, joka on prosessilla aktiivisesti käytössä. Useimmissa sovelluksissa on huomattavaa paikallisuutta, joka vaihtelee ajan mukana, näin työalan sijainti ja määrä vaihtelee myöskin.

UDP kts. User Datagram Protocol.

UNIX (tässä esityksessä) Yleisnimitys käyttöjärjestelmille, jotka pohjautuvat tai muistuttavat alunperin AT&T:n[11] kehittämää UNIX-käyttöjärjestelmää.

UNIX/32V AT&T:n vuonna 1978 esittelemä UNIX-versio, johon useat muut UNIX-versiot perustuvat.

URG (TCP:ssä) Lippu, joka ilmoittaa segmentissä olevan kiireellistä tietoa, esimerkiksi keskeytyskomento, joka tulee välittömästi toimittaa sovellukselle ohi normaalin datan.

User Datagram Protocol Internetin siirtokerroksen protokolla, joka tarjoaa yhteydettömän epäluotettavan sähkepalvelun [Pos80].

VBR Variable Bit Rate. Vaihtelevan bittinopeuden liikenne, esimerkiksi vakiolaadulla koodattu videokuva tuottaa VBR-liikennettä.

[10] http://www.tpc.org/
[11] http://www.att.com/

VFS Virtual File System. Rajapinta, joka tarjoaa yhtenäisen rajapinnan kaikille erityyppisille tiedostojärjestelmille LINUX'ssa.

vnode Virtual node. Levyjärjestelmäriippumaton tieto tiedostosta, vrt. inode.

vuo Yhteys. Määrittelytarkkus erilainen eri protokollakerroksilla.

vuotava ämpäri GCRA-algoritmin havannollistus.

WWW World Wide Web. Käytetään yleisnimityksenä (yleensä) HTTP-protokollalla noudettavista HTML-sivuista ja näihin liittyvistä dokumenteista.

ydintila Prosessorin tila, jossa käyttöjärjestelmän koodi suoritetaan. Oikeudet esimerkiksi muistin ja oheislaitteiden suhteen ovat rajoittamattomat vrt. käyttäjätila.

yhteys Kokonaisuus, joka määrää prosessin käytettävissä olevan muistin ja muut suorituksen kannalta tarpeelliset resurssit kuten pinon ja ohjelmalaskurin.

Luku 1

Johdanto

Perinteisessä verkoissa tietoliikennekapasitettia jaetaan kiinteinä kaistanleveyksinä, joka on varattu yhteydelle riippumatta siitä kuinka paljon yhteys siitä käyttää. Tämä on tehotonta verkkoresurssien hyväksikäyttöä. Nykyisin pyritään jakamaan siirtokapasiteettia useille yhteyksille hyödyntäen tilastollista kanavoitumista, esimerkiksi ATM-tekniikkaa hyväksikäyttäen. Liikenteenhallinnalla pyritään takaamaan riittävä laatutaso yhteyksille ja samalla hyödyntämään verkkoresurssia optimaalisesti.

Liikenteenhallinnan kannalta purskeinen liikenne on ongelmallista, erityisesti on-off-liikenne, jota tietokoneiden datasiirto tyypillisesti tuottaa. Hyvä purskeiden sieto yhdistettynä tehokkaaseen siirtokapasiteetin jakoon vaatii suuria puskureita, mikä aiheuttaa kustannuksia verkkoelementeissä sekä voi lisätä reaaliaikaliikenteen viivettä. Onkin järkevää pyrkiä vähentämään liikenteen purskeisuutta päätelaitteissa. Tulevaisuudessa yhteyden kustannuksiin tai sen saamaan palvelun laatuun voivat vaikuttaa sen karakteriset ominaisuudet, esimerkiksi purskeisuus [Kil97].

Liikenteen purskeisuuteen ja purskeiden luonteeseen vaikuttaa käytetty prosessien vuorottelualgoritmi[1] ja kilpailevien prosessien määrä [Lin96]. Lin on tutkinut tätä lisensiaattityössään mittaamalla liikennettä kahdella eri käyttöjärjestelmällä erilaisilla taustaprosessien määrillä. Eri tapauksien liikenteille etsittiin häviöttömät vuotavan ämpärin parametrit sekä määritettiin saapumisien lukumäärän hajontaindek-

[1] scheduling algorithm

si. Käyttämällä takaisinkytkentää muokkaimelta[2] vuorotelujärjestelmään voitiin simuloinnissa estää soluhäviöt.

Tämän työn tarkoituksena on varmentaa tehtyjä havaintoja ja käyttöjärjestelmien toimintaa tutkimalla löytää perusteet havaituille ilmiöille. Mittauksien toistettavuuden helpottamiseksi luodaan mittausympäristö.

Aluksi tarkastellaan vuorottelua ja verkkoliikennettä yleisesti, tavoitteena on luoda perusteet käsitteille. Lisäksi tiivistelmänomaisesti käsitellään liikenteen karakterisointia. Mittaustuloksiin sovellettiin ATM-verkon liikenneparametreja, vaikka mittauksia ei tehtykään ATM-verkossa, koska nämä ovat yleisesti käytettyjä ja jatkotutkimuksissa on tarkoitus mitata liikennettä myöskin ATM-verkoissa.

Lähemmin tutustutaan kahteen melko laajassa käytössä olevaan käyttöjärjestelmään. Erityisesti vertaillaan niiden ominaisuuksia aikaisemmin esitettyyn yleiseen rakenteeseen ja keskenään.

Mittaukset tehtiin kahdessa vaiheessa: ensimmäisessä selvitettiin tapahtumien aikatasoja, jotta tietojen tallennus voitaisiin kohdistaa oikein. Toisia mittauksia varten työkalua kehitettiin, jotta tietojen tallennus onnistuisi tehokkaasti. Ensimmäisessä mittauksessa esiintyneitä virhelähteitä pystyttiin poistamaan. Työn tulokset perustuvat tähän jälkimmäiseen mittaukseen.

Mitattua liikennettä sekä ohjelmasta sekä käyttöjärjestelmästä kerättyä tietoa arvoitiin aikakaavioilla sekä peräkkäisten tapahtumien välisten aikojen histogrammien avulla. Verkosta mitattua liikennettä karakterisoitiin ATM-verkon liikenneparametreillä ja saapumisten määrän hajontaindeksillä.

Työn lopuksi arvioidaan mahdollisuuksia käytännössä toteuttaa vuorottelun ohjaus verkkoliikenteen perusteella sekä esitetään jatkotutkimusten kohteita.

[2] shaper

Luku 2

Prosessien vuorottelu

Yleisesti käytössä olevat mikroprosessorit pystyvät suorittamaan vain yhden tehtävän kerrallaan. Rajoittuminen vain yhden tehtävän suorittamiseen kerrallaan alusta loppuun ei ole järkevää vaan tehtäviä kannattaa suorittaa useampaa limittäin. Tämä tehostaa resurssien käyttöä ja mikäli ajettavaa ohjelmaa vaihdetaan useita kertoja sekunnissa, käyttäjille syntyy usean yhtäaikaisen tehtävän vaikutelma. Valinta ajettavan ohjelman eli prosessin välillä luodaan vuorottelupolitiikan[1] perusteella, varsinaisen vaihdoksen toteuttaa vuorottelumekanismi.[2]

Politiikka ja mekanismi voidaan erottaa toisistaan: ainoastaan mekanismi vaatii erityisoikeuksia. Useimmissa järjestelmissä nämä kuitenkin muodostavat yhtenäisen, tiukasti säikeisiin, yhteyden vaihdoksiin[3], kelloihin ja ajastimiin kytketyn kokonaisuuden. Usein olisi hyödyllistä sovelluskokonaisuudessa jakaa prosessoriaikaa prosessien kesken erilaisin kriteerein. Tämä helpottaisi muun muassa järjestelmän suojaamista erilaisilta palvelun esto[4]-hyökkäyksiltä sekä tehostaisi multimediaohjelmien toimintaa [FS96].

Vuorottelupolitiikan parametreja on käytetty usein väärin. Eräässä järjestelmässä prosessia pidettiin interaktiivisena, mikäli se tulosti ruudulle alle sekunnin välein. Ohjelmoija muutti ohjelmaansa tulostamaan merkin ruudulle vajaan sekunnin välein, jolloin

[1] scheduling policy
[2] scheduling mechanism
[3] context switch
[4] denial-of-service

prosessi pysyi interaktiivisten prosessien luokassa ja sai näin korkeamman prioriteetin [SG94, s. 157]. Nykyisin erilaisilla verkkosovelluksilla[5] on mahdollista saattaa kone käyttökelvottomaksi luomalla suuri määrä säikeitä. Tasaisen jaon perusteella jokainen säie saa saman verran prosessoriaikaa: käyttäjän sovellusohjelmille jää alle prosentti prosessoritehosta, mikäli säikeitä on sata [Jav96].

2.1 Prosessi

”Prosessi” tarkoittaa tässä työssä yhtä ajossa olevaa ohjelmaa eli yksisäikeistä prosessia. Uusimmissa käyttöjärjestelmissä samassa prosessissa voi olla useita säikeitä eli keveitä prosesseja.

Prosessi muodostuu yleisesti varsinaisen ohjelmakoodin, muuttujaosan ja ajoaikasen tilatiedon lisäksi pinosta, joka sisältää tilapäistä tietoa kuten aliohjelmien parametrit, niiden paluuosoitteet ja tilapäiset muuttujat. Ajoaikaisesta tilatiedosta tärkeimmät ovat muistimääritykset, ohjelmalaskuri ja muut prosessorin rekisterit, jotka tallennetaan kunkin prosessin hallintatietueeseen. Tilatieto määrää prosessin yhteyden; ajettavaa prosessia vaihdettaessa tämä vaihtuu.

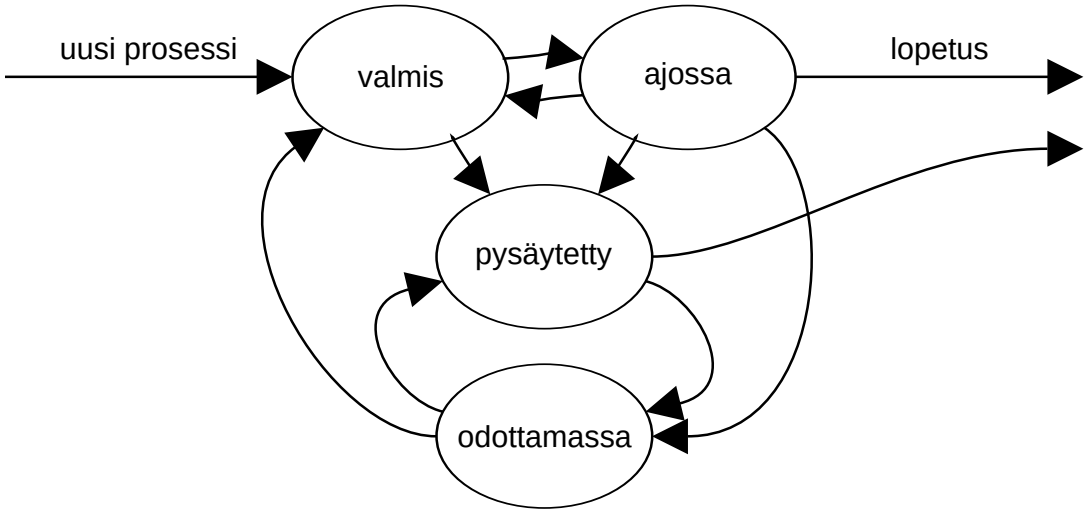
2.1.1 Prosessin tila

Prosessi asetetaan ajokelpoisten prosessien jonoon, kun se luodaan.[6] Prosessiin liittyy tilatieto (hallintatietue), joka sisältää tiedon prosessin tilasta (kuva 2.1), ohjelmalaskurin, CPU:n rekisterit, tiedon vuorottelusta, muistinhallintatiedon, laskutustiedon ja I/O:n tilan.

Prosessit siirtyvät valmis-jonosta ajoon kukin omalla vuorollaan. Odottamaan prosessi joutuu esimerkiksi I/O-tapahtumien aikana, koska nämä tyypillisesti kestävät kymmenestä millisekunnista aina minuutteihin; ylärajaa ei ole. Esimerkiksi näppäimenpainallusten väli interaktiivisessa käytössä voi olla hyvinkin pitkä.

[5] applet

[6] Eräissä järjestelmissä luotu prosessi otetaan välittömästi ajoon



uusi	prosessi on juuri luotu
valmis	prosessi on ajokelponen ja odottamassa vuoroaan prosessorille.
ajossa	prosessi suorittaa käskyjä
odottamassa	prosessi odottaa jotain tapahtumaa tai resurssia, esimerkiksi I/O-pyyntöä tai kirjoitinta
pysäytetty	ohjelman suoritus on (ulkoa) tilapäisesti pysäytetty
lopettanut	ohjelma on lopettanut suorituksensa

Kuva 2.1: Prosessin tilakaavio [SPG91].

Useat vuorottelupolitiikat voivat poistaa prosessin ajosta myös prosessoriajan käytön perusteella, ts. jokaisella prosessilla on maksimiaika, jonka prosessi voi kerrallaan käyttää prosessoria.

Sovellukset voidaan jakaa vuorotteluvaatimusten perusteella [GGV96]:

erittäin aikakriittinen[7] Tehtävillä on tiukat aikataulut, joista myöhästyminen voi aiheuttaa vahinkoja; esimerkiksi prosessinohjaus.

aikakriittinen[8] Tehtävillä on aikataulut, mutta niistä myöhästyminen ei aiheuta vahinkoja vaan on esimerkiksi lievästi häiritsevää. Nämä sovellukset vaativat tietyn palvelun laadun (QoS) esimerkiksi maksimiviiveen ja läpäisykyvyn suhteen.

paras mahdollinen[9] Eri tehtävät pyritään suorittamaan mahdollisimman nopeasti, mutta tapahtumilla ei ole aikatauluja eikä aikatakeita. Perinteiset vuorottelujärjestelmät tarjoavat yleensä ainoastaan tämän palvelun.

2.2 Vuorottelupolitiikka

Prosessin suoritus muodostuu vuorottelevista CPU- ja I/O-purskeista. Näiden keskinäinen pituus vaihtelee järjestelmästä ja sovelluskokonaisuudesta toiseen, mutta tyypillisesti jakauma on (hyper)eksponentiaalinen: suurin purskeista osa on lyhyitä ja vain harvat ovat pitkiä [SG94].

Vuorottelupolitiikka voidaan jakaa useisiin eri aikatasoihin:

pitkä aikaväli (yli 10 s) Tämä on käytössä erityisesti eräajojärjestelmissä, joissa prosesseille voidaan asettaa maksimiajoaika ja -muistinkäyttö. Näin saadaan järjestelmä toimimaan optimaalisella kuormituksella, koska reaalimuistin täyttyessä suorituskyky laskee ja prosessori on ”tyhjäkäynnillä” odottaen prosessien muistisivuja näennäismuistista [Kos96].

Useimmissa järjestelmissä ei ole pitkän aikavälin politiikkaa vaan kuormituksen säätely toimii ainoastaan ihmisten mukavuudenhalun ohjaamana: ei käytetä järjestelmää tai käynnistetä enempää prosesseja, jos järjestelmä on liian hidas.

keskipitkä aikaväli (1 – 10 s) Useissa järjestelmissä pitkään odottamassa oleva prosessin käytössä olevan muistialueen sisältö tai sen osia siirretään levylle (sivutetaan), jolloin keskusmuistia saadaan ajokelpoisten prosessien ja levypuskureiden käyttöön.

Valikoiden voidaan siirtää kokonaisia prosesseja levylle, mikäli muistista on pulaa. Jäljellä oleville prosesseille saadaan näin käyttöön muistia työalan[10] verran. Sopivin väliajoin (muutamia kymmeniä sekunteja) muistista poistettuja prosesseja vaihdetaan. Näin voidaan välttää ”puiminen”[11], jossa koko prosessoriteho kuluu sivujen siirtämiseen levylle ja takaisin [MBKQ96].

lyhyt aikaväli (alle 1 s) Tämän tarkoitus on pitää prosessori mahdollisimman tehokkaalla käytöllä ja antaa käyttäjille vaikutelma yhtäaikaisesta suorituksesta. Tyypillisesti valinta ja mahdollinen vaihto suoritetaan 10 – 100 millisekunnin välein. Valinta on tehtävä nopeasti, jotta siitä ei syntyisi tarpeetonta tehohukkaa.

[10] working set of pages

[11] trashing

Vuorottelualgoritmi vaikuttaa moniin tekijöihin ja parhaan algoritmin valinnassa voidaan järjestelmän käyttötarkoituksesta riippuen eri tavoin painottaa seuraavia tekijöitä [SPG91]:

CPU:n käyttöaste CPU:n käyttöaste pyritään pitämään mahdollisimman korkeana: laskentaintensiivisessä käytössä (vähän I/O:ta) voidaan päästä hyvin lähelle 100 % käyttöastetta, kun taas I/O-painotteisessa käytössä käyttöaste voi olla vain muutamia kymmeniä prosentteja.

läpäisykyky Mitta-arvona voidaan käyttää suoritettuja prosesseja aikayksikössä. Asteikko riippuu tehtävien suuruudesta: suurissa laskentasovelluksissa puhutaan tehtävistä tunnissa, tietokantakäytössä tapahtumia minuutissa. Interaktiivisessa käytössä yksikkönä on usein tehtäviä sekunnissa.

Tietokonejärjestelmien läpäisykyky ilmaistaan usein SPECrate-arvona, josta on erikseen kokonaislukupainotteinen ja liukulukupainotteinen versio. Arvo saadaan suorittamalla suorituskykymittaussovelluksia toistuvasti ja mittaamalla kuinka monta kertaa ne voidaan suorittaa tunnissa. Ongelmana suorituskykymittaussovelluksilla on niiden ikä: monet niistä ovat tyypillisesti jopa vuosikymmeniä vanhoja ohjelmia eivätkä vastaa rakenteeltaan nykyisiä ohjelmia. Tietokantajärjestelmien suorituskykyä ja hinta-suorituskykysuhdetta mitataan TPC-arvoilla.

läpimenoaika Käyttäjille merkittävin kriteeri on tehtävän suorittamiseen kuluva aika. Tämä sisältää varsinaisen suoritukseen, I/O-toimintoihin, valmis-jonossa odottamiseen ja käynnistämiseen kuluneen ajan.

odotusaika Vuorottelualgoritmit eivät yleensä muuta varsinaiseen suoritukseen tai I/O toimintoihin kuluvaa aikaa, vaan ne vaikuttavat siihen kuinka kauan prosessi on valmis-jonossa. Vuorottelualgoritmeilla voi olla vaikutusta välimuistien toiminnan tehokkuuteen.

vasteaika Interaktiivisessa käytössä merkittävin suure ei ole koko sovelluksen suorituksen kokonaisaika vaan kuinka nopeasti sovellus vastaa käyttäjän syötteeseen eli *käyttäjän syötteen aiheuttaman toiminnon läpimenoaika*.

> Sujuvaan käyttöön ensimmäisen vasteen (esimerkiksi merkin tulostuminen näyttöön näppäimenpainalluksen jälkeen) tulisi tulla 0,1 s. Käyttäjän ajatus pysyy yleensä kasassa alle sekunnin viiveessä, mutta käyttäjältä katoaa tunne suorasta käsittelystä. Yli sekunnin kestävissä tehtävissä tulee olla jonkinlainen ilmoitus tehtävän edistymisestä. Käyttäjälle tulee antaa arvio jäljellä olevasta ajasta, mikäli tehtävän suorittaminen kestää yli kymmenen sekuntia. Näin pitkillä odotusajoilla käyttäjien keskittyminen herpaantuu ja he yleensä alkavat tehdä jotain muuta odottaessaan [Nie93].

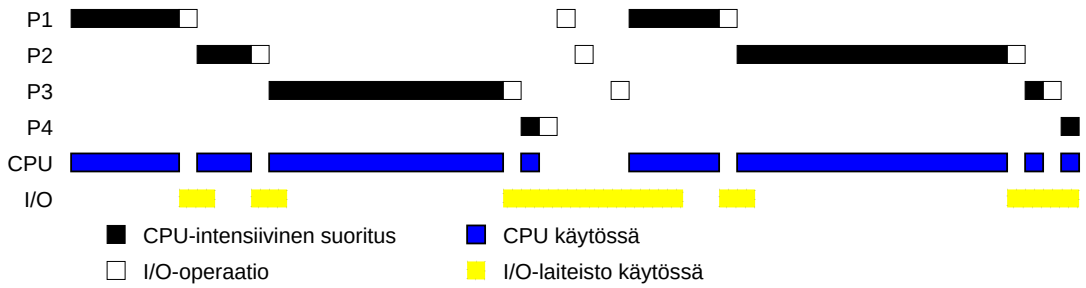
Optimaalisessa algoritmissa käyttöaste ja läpäisykyky ovat mahdollisimman suuret ja eri aikasuureet mahdollisimman lyhyet. Interaktiivisissa järjestelmissä maksimaalinen vasteaika pyritään saamaan mahdollisimman pieneksi, jotta käyttäjät kokevat palvelun hyvänä. Järjestelmän toiminnan on oltava ennustettavaa, niinpä vasteajan varianssin tulisi olla mahdollisimman pieni, jotta käyttäjä voisi sopeutua rytmiin [Nie96, Nie93].

Henkilökohtaisissa tietokoneissa käyttäjät ovat tottuneet hyvin vakiolliseen vasteeseen: sama toiminto vie aina saman ajan. Monenkäyttäjän koneissa käyttäjä joutuu jakamaan resursseja useiden käyttäjien kanssa ja näin vasteajat voivat vaihdella paljonkin. Viiveeseen aiheuttaa vaihtelua myös tiedonsiirtonopeudessa tapahtuvat muutokset, jotka johtuvat yleensä verkon kuormituksen vaihtelusta ja ruuhkista. Verkon vaikutus kasvaa jatkuvasti, koska yhä useampi sovellus hyödyntää verkkoa.

Verkkopalvelimessa suurin osa tapahtumista on kestoltaan varsin lyhyitä, tyypillisesti sekunnin luokkaa. Järjestelmän toimintatavasta riippuu, tuleeko pyrkiä minimoimaan vasteaika vai läpimenoaika. Jokaista tapahtumaa varten voidaan luoda oma prosessi tai yksi prosessi palvelee useita pyyntöjä, yhden kerrallaan. Yksi prosessi voidaan myös varata suorittamaan ainoastaan yhden käyttäjän kyselyjä.

Ensimmäiset HTTP- eli WWW-palvelimet palvelivat vain yhden pyynnön yhdellä prosessilla. Prosessin luominen on monissa järjestelmissä raskas toiminto, joten nykyään kukin palvelinprosessi palvelee useita pyyntöjä mutta vain yhden kerrallaan.

HTTP-protokollan versio 1.1 muuttaa järjestelmää toimivaksi käyttäjäkohtaisesti eli yksi dokumentti ja siihen liittyvät kuvat ja muut objektit haetaan yhdellä TCP-yhteydellä, mikäli ne tulevat samalta palvelimelta. Yhteys pidetään myös auki jonkin ai-



Kuva 2.2: FCFS-algoritmin saattueominaisuus [SPG91].

kaa odottaen, että seuraavatkin dokumentit haetaan pian samalta palvelimelta. Tässä säästetään yhteyden muodostukseen kuluvaa aikaa, joka on merkittävä ruuhkaisilla ja etenkin mannertenvälisillä yhteyksillä [BLFF96, NGBSP97, FGM+97].

2.2.1 Ensin tullut, ensin palveltu -algoritmi

Yksinkertaisin vuorottelualgoritmi on ”ensin tullut, ensin palveltu”[12] (FCFS). Tässä kutakin prosessia suoritetaan niin kauan kunnes se vapaaehtoisesti luopuu omasta vuorostaan suorittamalla I/O:ta tai mahdollisesti jollain muulla käyttöjärjestelmäkutsulla. Tätä ominaisuutta voidaan käyttää yhteistyöllisessä[13] eli keskeyttämättömässä[14] moniajossa, joissa prosessi voi menettää ajovuoronsa vain suorittamalla tiettyjä käyttöjärjestelmäkutsuja.

Tällä algoritmilla keskimääräinen odotusaika on pitkä ja erityisesti vasteaika on huono, koska CPU-intensiivinen prosessi voi pitää CPU:ta varattuna mielivaltaisen pitkään.

Algoritmin ominaisuutena on vaihtelu I/O- ja CPU-intensiivisen toiminnan välillä: toinen resursseista on vuoronperään vajaakäytöllä, kuten kuvassa 2.2 on esitetty.

[12] first-come, first served
[13] co-operative
[14] non-preemptive

2.2.2 Lyhyin työ ensin -algoritmi

Läpimenoajallisesti optimaalisin algoritmi on ”lyhyin työ ensin”[15] (SJF) [SPG91]. Algoritmi valitsee ensimmäiseksi prosessin, jolla on lyhyin CPU-purskeaika[16]. Kahdesta prosessista, joilla on sama purskeaika, valitaan ensimmäisenä tullut.

Algoritmin ongelma on purskeajan arvioiminen. Eräajojärjestelmissä voidaan käyttää CPU-aikarajoja, jolloin käyttäjällä on motivaatio arvioida prosessin viemä aika. Lyhyen aikavälin vuorottelussa purskeaikaa ei voida tietää mutta sitä voidaan yrittää arvioida edellisiin purskeisiin perustuen.

Arvioinnissa käytetään yleensä eksponentiaalisesti painotettua liukuvaa keskiarvoa. Merkitsemällä purskeen pituutta t_n ja arviota seuraan purskeen pituudesta τ_{n+1}, saadaan:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n, \qquad 0 \leq \alpha \leq 1 \tag{2.1}$$

Muuttamalla α:n arvoa voidaan historian painotusta muuttaa: mikäli $\alpha = 1$ niin ainoastaan viimeisimmällä purskeella on vaikutusta ($1 - \alpha = 0$). Kirjoittamalla yhtälö auki (kaava 2.2) nähdään, että mitä vanhempi purske, sitä vähemmän sillä on vaikutusta. Alkuarvona τ_0 voidaan käyttää järjestelmän keskiarvoa.

$$\tau_{n+1} = \alpha \left[t_n + (1 - \alpha)t_{n-1} + \ldots + (1 - \alpha)^j t_{n-j} + \ldots \right] + (1 - \alpha)^{n+1}\tau_0 \tag{2.2}$$

SJF-algoritmia voidaan käyttää sekä keskeyttävässä[17] että yhteistyöllisessä[18] moniajossa. Keskeyttävässä moniajossa ohjelman tullessa ajokelpoiseksi tämä otetaan heti ajoon, mikäli sen ennakoitu purskepituus on lyhyempi kuin ajossa olevan: algoritmista usein käytetäänkin nimitystä ”lyhin jäljellä oleva aika”[19].

[15] shortest job first

[16] Aika, jonka prosessi käyttää CPU:ta ennen I/O-operaatiota tai lopettamista.

[17] preemptive

[18] co-operative

[19] shortest remaining time first

2.2.3 Prioriteetti-algoritmi

Prioriteetti-algoritmissa jokaiselle työlle annetaan prioriteetti ja korkeimmalla prioriteetilla olevaa ajovalmista työtä ajetaan. Samalla prioriteetilla olevien töiden välillä voidaan käyttää esimerkiksi FCFS-algoritmia.

Prioriteetti ilmaistaan yleensä lukuarvona, joka kuuluu tiettyyn väliin, esimerkiksi 0 – 127. Prioriteetin ja lukuarvon välillä käytetään molempia mahdollisia riippuvuuksia: joko pienin tai suurin luku on korkein prioriteetti. Tässä esityksessä samoin kuin yleensä kirjallisuudessa käytetään ensimmäistä tapaa eli pienin luku on korkein prioriteetti.

SJF-algoritmi on erityistapaus prioriteettipohjaisissa vuorottelualgoritmeissa: prosessin prioriteetti (p) on käänteisarvo seuraavasta arvioidusta purskepituudesta (τ):

$$p = \frac{1}{\tau}. \tag{2.3}$$

Prioriteetteja voidaan asettaa joko sisäisesti tai ulkoisesti sekä näiden yhdistelmänä. Sisäiset prioriteetit voivat pohjautua esimerkiksi CPU-ajan käyttöön kuten SJF-algoritmissa tai johonkin muuhun kriteeriin kuten muistin käyttöön tai I/O-määrään. Reaaliaikajärjestelmissä kriteerinä käytetään kyseisen tehtävän suorittamiseen jäljellä olevaa aikaa.[20]

Ulkoiset prioriteetit voivat olla periaatteessa mitä tahansa, esimerkiksi rahoitukseen tai henkilöhierarkiaan perustuvia. Usein käytetään sisäisten ja ulkoisten prioriteettien yhdistelmää, jossa kullakin prosessilla on ulkoisesti asetettu perusprioriteetti, johon sitten lisätään tai vähennetään sisäisen prioriteetin arvo.

Prioriteetti-algoritmia voidaan käyttää sekä keskeyttävänä että ei-keskeyttävänä kuten SJF-algoritmia.

Suurin prioriteetti-algoritmeihin liittyvä ongelma on jatkuva estyminen.[21, 22] Alemman prioriteetin prosessi ei pääse koskaan ajoon, mikäli järjestelmässä on jatkuvasti

[20] earliest deadline first
[21] indefinete blocking
[22] starvation

ajokelpoisia korkeamman prioriteetin prosesseja. Tätä voidaan korjata vanhentamisella[23]: prosessin prioriteettia lisätään esimerkiksi yhdellä pykälällä jokaista minuuttia kohti, jonka prosessi on odottanut valmis-jonossa.

Toinen prioriteettipohjaisiin algoritmeihin liittyvä ongelma on prioriteetin kääntyminen.[24] Prosessin todellinen prioriteetti voi pudota alas, jos se haluaa käyttöönsä resurssin, joka on alemman prioriteetin prosessin hallussa. Tämä voi aiheuttaa jopa lukkiuman[25], jos prosessi kiireodottaa[26] resurssia, jolloin alemman prioriteetin prosessi ei pääse vapauttamaan resurssia. Tätä ongelmaa voidaan korjata prioriteetin perinnällä eli prosessi luovuttaa tilapäisesti oman prioriteettinsa prosessille, jonka hallussa resurssi on [SG94, Tan92].

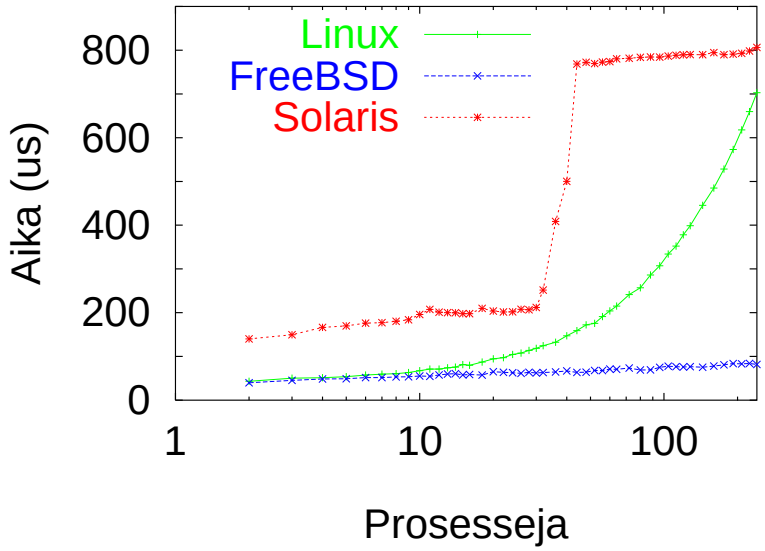
2.2.4 Kiertävä vuoro -algoritmi

Kiertävä vuoro[27] (RR) on suunniteltu erityisesti monikäyttäjäympäristöön. Ajokelpoinen prosessi otetaan valmis-jonosta ja sitä ajetaan siihen asti kunnes se joutuu odottamaan esimerkiksi I/O operaation seurauksena tai kunnes se käyttää aikayksikön[28] (tyypillisesti 10 – 100 ms) prosessoriaikaa. Prosessin suoritus keskeytetään ja se siirretään valmis-jonon viimeiseksi, mikäli prosessi käyttää aikayksikön loppuun.

Algoritmin toiminta on siis keskeyttävä[29]. Merkitsemällä ajokelpoisten prosessien määrää n ja aikayksikköä on q, silloin jokainen prosessi saa $1/n$ CPU-ajasta ja maksimiodotus prosessille on $(n-1) \times q$. Jokainen prosessi saa CPU-aikaa 100 ms kerran sekunnissa, mikäli aikayksikkönä on $q = 100$ ms ja järjestelmässä on $n = 10$ ajokelpoista prosessia

Äärimmäinen esimerkki kiertävän vuoron käytöstä on prosessorin jakaminen[30], jossa prosessia vuorotellaan hyvin lyhyin ($< 1\mu$s) aikavälein. Toiminta vastaa useam-

[23] aging
[24] priority inversion
[25] deadlock
[26] busy wait
[27] round-robin
[28] quantum
[29] preemptive
[30] processor sharing



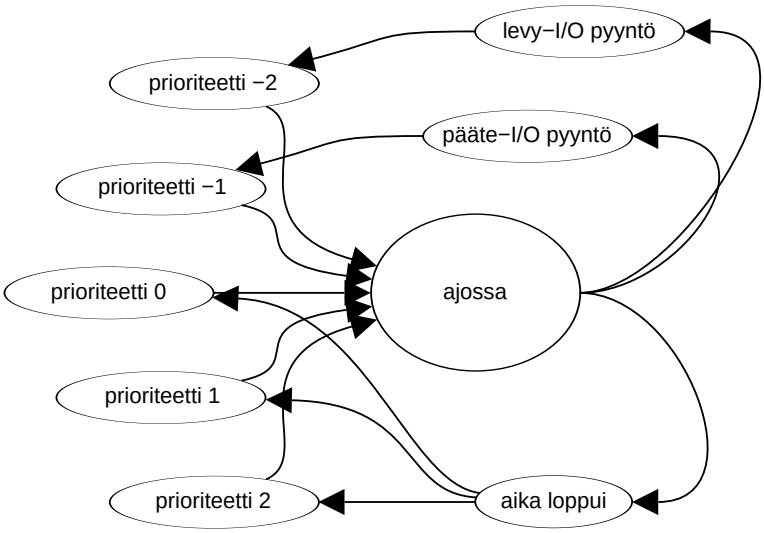
Kuva 2.3: Yhteyden vaihtamiseen kuluvat ajat eräissä käyttöjärjestelmissä 100 MHz:n Pentium prosessorilla [LB96].

paa vastaavasti hitaampaa prosessoria. Tätä on aikaisemmin käytetty menestyksellisesti erikoisrakenteisissa prosessoreissa, joissa on esimerkiksi 10 rekisterisarjaa mutta vain yksi aritmetiikkayksikkö (ALU) [Tan92].

Prosessin yhteyden vaihtamiseen kuluva aika riippuu prosessorista, laitteistototeutuksesta, käyttöjärjestelmästä ja prosessien määrästä. Samallakin laitteistolla aika vaihtelee 50 – 800 μs:n välillä, kuten kuvasta 2.3 käy ilmi kolmen käyttöjärjestelmän suhteen. Vaihtamiseen kuluvasta ajasta aiheutuva tehohäviö rajoittaa lyhintä käyttökelpoista vuorotteluväliä.

Monet nykyiset käyttöjärjestelmät käyttävät kiertävän vuoron järjestelmää monitasoisilla jonoilla tai toiminnallisesti vastaavaa järjestelyä. Monitasoisilla jonoilla prosessi siirretään aina kutakin tilannetta vastaavaan jonoon odottamaan suoritusta. Eräs järjestely on esitetty kuvassa 2.4, jossa on viisi prioriteettijonoa, joista korkeimmat (-2 – -1) on varattu I/O:ta suorittavalle prosessille ja normaaliprioriteetit (0 – 2) CPU:n käyttövuoroa odottaville prosesseille.

Asettamalla I/O-toimintoja suorittavat prosessit korkeammalle prioriteetille, saadaan sekä I/O-järjestelmän että CPU:n käyttöaste korkealle: on todennäköistä, että I/O-toiminnon suorittanut prosessi suorittaa kohta uudestaan I/O-toiminnon. Muussa tapauksessa prosessi siirtyy käytettyään oman aikaviipaleensa johonkin normaalijonoista 0 – 2.



Kuva 2.4: Prioriteettijonoihin pohjautuva järjestelmä.

2.3 TCP-yhteydet

2.3.1 IP protokollaperhe

Useat organisaatiot, mm. ISO, ovat määritelleet omia internet-protokolliaan. Tässä työssä ”internet-protokollilla” tarkoitetaan DoD:n tai nykyisin IETF:n internet-protokollaperheeseen kuuluvia protokollia, jotka ovat ylivoimaisesti suosituimmat ja merkittävimmät.

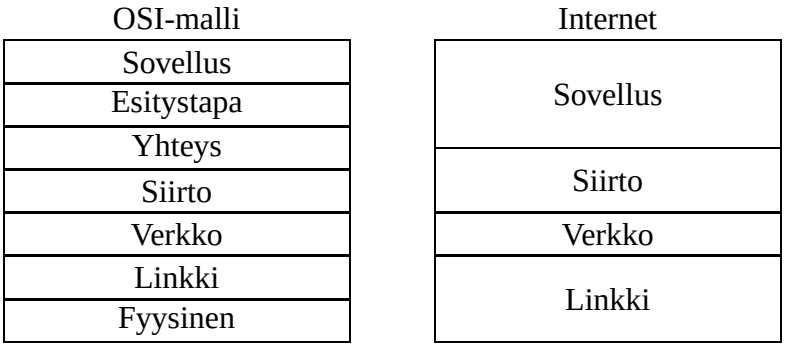
Tässä esityksessä käytetään BSD Socket -sovellusohjelmointirajapintaa (API) ja -käsitteistöä käsiteltäessä tiedonsiirtoa ohjelmistolliselta puolelta. Valtaosa nykyisin käytössä olevista TCP/IP-toteutuksista tarjoaa tämän rajapinnan [MBKQ96].

Internet-protokollat voidaan esittää 4-tasoisena mallina erotuksena OSI-mallissa esitetylle 7-kerroksisille mallille [ISO94, Bra89b]. Kerrosjako on esitetty kuvassa 2.5.

sovelluskerros Ylintä kerrosta ei internet-mallissa jaeta erikseen alikerroksiin, kuten OSI-mallissa, vaikkakin protokollien sisällä voi olla alikerroksia. Tämä kerros vastaa OSI-mallin sovellus- ja esitystapakerroksia.

Sovelluskerroksen protokollat voidaan jakaa kahteen luokkaan:

1. Suoraan käyttäjille tarjottavat palvelut, esimerkiksi pääteyhteydet Telnet



Kuva 2.5: OSI- ja Internet-mallien kerrosrakenteen vertailua [ISO94, Bra89b].

[PR83b], sähköposti SMTP [Pos82, Cro82] sekä tiedostojen siirto FTP [PR85] ja HTTP [BLFF96, FGM+97].

2. Tukipalvelut, esimerkiksi nimipalvelu DNS [Moc87a, Moc87b] ja verkonhallinta SNMP [SFDC90].

siirtokerros Siirtokerros vastaa toiminnallisesti kutakuinkin OSI-mallin siirto- ja yhteysjaksokerrosta. Tärkeimmät siirtokerroksen protokollat ovat Transmission Control Protocol (TCP) [Pos81c], joka tarjoaa kaksisuuntaisen virheenkorjaavan virtuaaliyhteyden, ja User Datagram Protocol (UDP) [Pos80], joka tarjoaa yhteydettömän epäluotettavan sähkepalvelun. TCP ei tarjoa tiedonsiirron osien erottamista eli tietueita: yhteys muodostuu kahdeksan bittiä pitkistä okteteista. TCP-yhteyden jakaminen loogisiin osiin on hoidettava sovelluskerroksella. UDP sen sijaan säilyttää tietueiden (sähkeiden) välit sovellusohjelmille saakka.

verkkokerros Kaikki internetin siirtokerroksen protokollat käyttävät Internet Protocol'laa eli IP:tä tiedon kuljettamiseen lähteestä kohteeseen. Se on yhteydetön eikä takaa viestien perillemenoa, eli ylempien protokollien on huolehdittava mahdollisista katoamisista, järjestyksen muuttumisesta ja kahdentumisista toipumisesta, mikäli tarpeen. Yhteydettömyys tekee välissä olevista yhdyskäytävistä eli IP-reitittimistä yksinkertaisia.

IP:aan kuuluu oleellisena osana Internet Control Message Protocol (ICMP) [Pos81a], joka huolehtii pääasiassa virheiden ja ruuhkien ilmaisusta. Toinen IP:n apuprotokolla Internet Group Management Protocol (IGMP) huolehtii jakeluryhmien muodostamisesta. Seuraavassa IP:n versiossa (versio 6 [DH96]) se on integroitu ICMP:hen [CD96].

Taulukko 2.1: Porttinumeroiden käytön jako.

porttinumero	käyttö
0–1023	Hyvin tunnetut porttinumerot
1024–49151	Rekisteröidyt porttinumerot
49152–65535	Dynamiset ja/tai yksityiset porttinumerot

linkkikerros Käytettävälle siirtotielle ja -menetelmälle tulee määritellä, kuinka IP-sähkeet kuljetetaan kyseisessä mediassa. IP ei aseta suuria vaatimuksia linkkikerroksille, niinpä IP-sähkeiden siirto on määritelty käyttäen sekä kirjekyyhkyjä että morsemerkkejä [Wai90, Eri96].

2.3.2 Yhteyden muodostus

Sovellusohjelmat jaetaan yleisesti asiakas- ja palvelinohjelmiin. Palvelinohjelma tai varsinaisten palvelinohjelmien edustaohjelma, joka on UNIX-tyyppisissä koneissa inetd [ine91], odottaa yhteydenmuodostuksia asiakasohjelmilta.

2.3.2.1 Palvelinohjelma

Palvelinohjelma luo ensin liikennöintipisteen[31] `socket()`-käyttöjärjestelmäkutsulla, jolle parametreina annetaan haluttu protokollaperhe, tyyppi ja protokolla. Kutsu luo käyttöjärjestelmään tietorakenteen, ja palauttaa tunnuksen, joka on tyypillisesti ”pieni positiivinen kokonaisluku” eli useimmissa toteutuksissa indeksi käyttöjärjestelmän taulukkoon. Tunnuksen lukuarvolla ei yleensä ole merkitystä sovellusohjelman kannalta.

Palvelinohjelma sitoo liikennöintipisteen tiettyyn TCP-porttinumeroon, joka on varattu kyseiselle palvelulle joko globaalisti [RP94] tai paikallisesti. Porttinumeroiden jako on esitetty taulukossa 2.1. Suurin osa Internetissä käytössä olevista palveluista on porttinumeroilla 0 – 1023.

Sitominen tapahtuu `bind()`-käyttöjärjestelmäkutsulla, jolle annetaan parametrina lii-

[31] socket

kennöintipisteen osoitteen määräävä tietorakenne. Tämä sisältää TCP-porttinumeron ja IP-osoitteen, joka voi olla joko määrätty tai avoin. Jälkimmäisessä tapauksessa mihin tahansa verkkoliitäntään tullut kutsu hyväksytään.

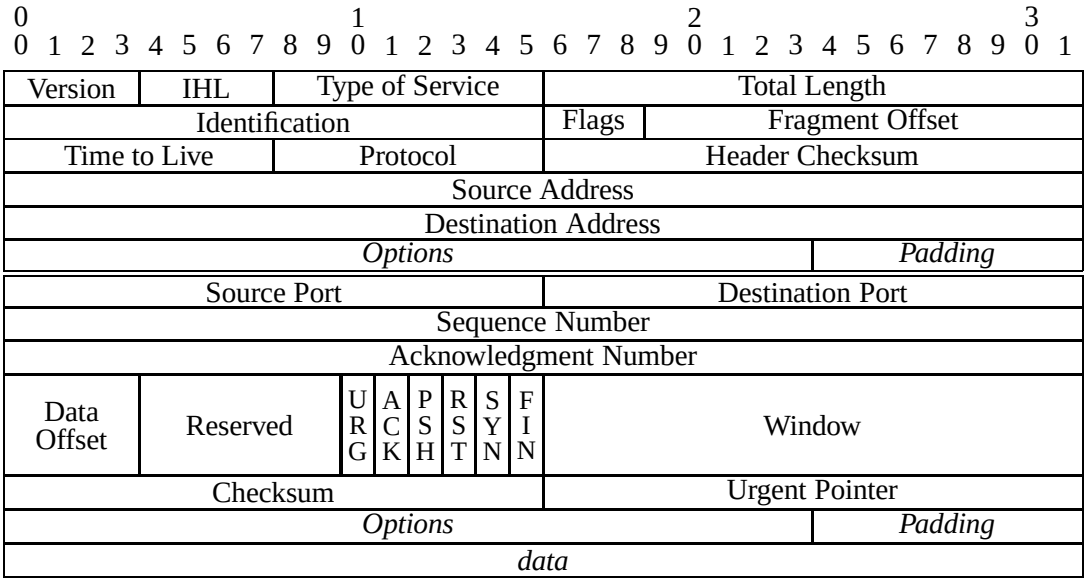
Liikennöintipisteen luominen ja sitominen on erotettu toisistaan, koska ohjelmointirajapinnan halutaan olevan mahdollisimman avoin. Käytetystä protokollasta riippuen sitomisen yhteydessä voidaan antaa lisämääreitä. Eräillä protokollaperheillä sitominen ei ole välttämätöntä ja sitomisessa käytetyt osoitteet poikkeavat toisistaan [MBKQ96].

Palvelinprosessi ilmoittaa tämän jälkeen olevansa valmis ottamaan yhteyksiä vastaan `listen()`-käyttöjärjestelmäkutsulla, jolle annetaan parametrina liikennöintipisteen tunnuksen lisäksi suurin sallittu jononpituus. Ellei sovellus ei ehdi käsitellä yhteyspyyntöjä riittävän nopeasti ja jono täyttyy, yhteyspyynnöt hylätään suoraan. Useimmissa käyttöjärjestelmissä jonon pituudella on maksimiarvo: joissakin vain 5, joissakin jopa 128 tai suurempi. Yhteyksien hyväksyminen tapahtuu `accept()`-kutsulla, joka palauttaa asiakkaan osoitteen lisäksi tunnuksen uuteen liikennöintipisteen, jonka kautta asiakas ja palvelin kommunikoivat. Alkuperäinen liikennöintipiste jää vastaanottamaan yhteyksiä.

2.3.2.2 Asiakasohjelma

Asiakasohjelma luo liikennöintipisteen ja sitoo sen tiettyyn osoitteeseen kuten palvelinohjelma käyttäen `socket()`- ja `bind()`-kutsuja. Paikallisena porttinumerona sovellus käyttää jotain vapaana olevaa porttia, joka on tyypillisesti ensimmäinen vapaana oleva numeroltaan suurempi kuin 1 023. Eräitä ohjelmia ajetaan pääkäyttäjän oikeuksin, jolloin se voi varata portteja numeroiltaan alle 1 024:n ja näin koettaa vakuuttaa vastapäälle olevansa erikoisoikeutettu prosessi, esimerkiksi käyttäjän tunnistamista varten. TCP-yhteys tunnistetaan neliköstä <*Lähdeosoite, Kohdeosoite, Lähdeportti, Kohdeportti*>.

Yhteys luodaan `connect()`-kutsulla, jolle annetaan parametrina liikennöintipisteen tunnus ja palvelimen osoitetietorakenne. Tietorakenne sisältää palvelimen IP-osoitteen ja porttinumeron. Koneen IP-osoite on selvitetty yleensä `gethostbyname()`-kutsulla, joka hakee tiedon joko paikallisesta tietokannasta tai DNS-järjestelmältä



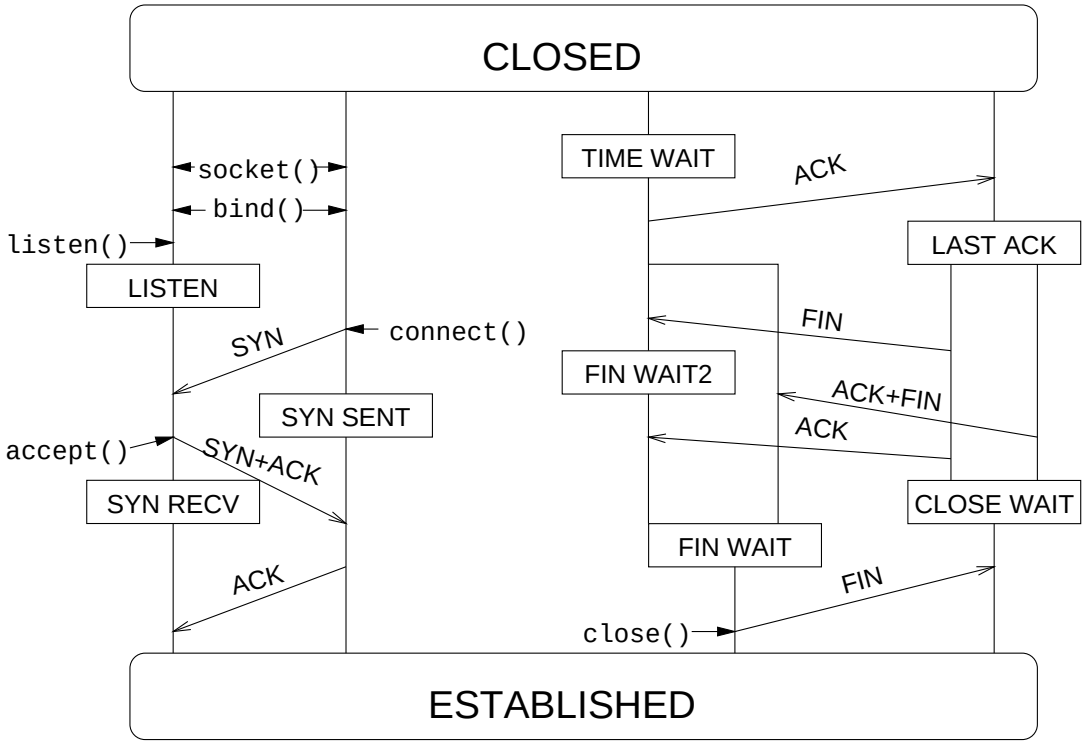
Kuva 2.6: IP- ja TCP-otsikot [Pos81b, Pos81c].

[Moc87a, Moc87b].

Yhteyden luonnissa käytetään kolmivaiheista kättelyä[32]. Yhteyttä luova osapuoli lähettää IP-TCP-tietosähkeen (kuva 2.6), johon on täytetty oikea porttinumerot, sarjanumero ja asetettu SYN-lippu päälle. Hyväksyttyään yhteyden `accept()`-kutsulla vastaanottaja vastaa tähän tietosähkeellä, jossa lähde- ja kohdeosoitteet on vaihdettu, sarjanumerona on toisesta riippumaton numero, kuittausnumerona on lähetetty toisen sarjanumero lisättynä yhdellä ja sekä SYN- että ACK-liput päällä. Yhteys on näin asiakkaan kannalta luotu ja asiakas voi lähettää ensimmäisen datansa kuittauksen mukana palvelimelle, jolloin yhteys on myös palvelimen kannalta luotu. Useimmat toteutukset kuitenkin lähettävät kuittauksen erillisessä kuittaussähkeessä. Tilasiirtymät ovat esitetty kuvassa 2.7.

Alkuperäisen määritelmän mukaan yhteyksien alkusarjanumerot tulee ottaa noin 250 kilotavua sekunnissa kasvavasta laskurista, jotta edellisistä yhteyksistä verkkoon jääneet segmentit eivät voisi aiheuttaa ongelmia. Suoja-aika on noin 4,5 tuntia 2 Mbit/s nopeudella mutta vain 5,4 minuuttia 100 Mbit/s nopeudella. Nykyiset toteutukset lisäävät sarjanumeroon satunnaisen komponentin, jotta sarjanumeron arvaamiseen perustuvat yhteyden kaappaamiset eivät onnistuisi [Mor85, Cen95].

[32] three-way handshake



Kuva 2.7: TCP-tilakone normaalitapauksessa [Pos81c].

Yhteydenmuodostus voi tapahtua myös samanaikaisesti: kumpikin osapuoli lähettää SYN-lipullisen sähkeen ja vastaa toisen lähettämään pelkällä ACK-lipulla. Tällöin yhteyden muodostamiseen tarvitaan neljä viestiä. Tätä menetelmää ei käytännössä sovelleta.

Eräissä sovelluksissa, esimerkiksi FTP-siirrossa, asiakkaan ja palvelimen roolit ovat päinvastoin: asiakas ilmoittaa palvelimelle porttinumeron, johon palvelin luo yhteyden datasiirtoa varten. Tämä käänteinen toiminta aiheuttaa ongelmia palomuurijärjestelmille, jotka estävät kaikki ulkoa sisääntulevat yhteydenmuodostusyritykset. Ongelma voidaan kiertää myöhemmällä FTP-protokollan laajennuksella, jossa asiakasohjelma luo yhteyden eli asiakas-palvelinsuhde on normaali [Bra89a].

2.3.3 Liikennöinti

Yhteyden luomisen jälkeen liikennöinti on samanlaista molemmista päistä eli TCP-siirron kannalta palvelin- ja asiakaspäillä ei ole merkitystä.

Ikkunan koko on tärkeää siirron tehokkuuden kannalta. Se ilmoittaa, kuinka paljon lä-

hettävä osapuoli voi lähettää tietoa ilman kuittausta eli kuinka paljon tietoa saa olla ”ilmassa”. Ikkunan koko ei voi olla suurempi kuin vastaanottajalla on puskuritilaa datan varastoimiseen. Maksimi ikkunan koko (65 535 tavua) on osoittautunut liian pieneksi yhteyksillä, joilla on suuri $kaistanleveys \times viive$ -tulo. Esimerkiksi maksimisiirtonopeus 5 000 kilometrin yhteysvälillä on 10 Mbit/s jo pelkästään valon fyysisen etenemisnopeuden vuoksi — verkkoelementtien aiheuttamat viiveet rajoittavat nopeutta edelleen [BBJ92].

Ikkunakoon kasvattamiseksi voidaan yhteyttä luotaessa neuvotella optiolla ikkunan skaalausparametrista N, jolloin ikkunan koko on $2^N \times window$. Skaalaus tulee käyttöön yhteydellä yhteensopivuussyistä vain mikäli molemmat ovat asettaneet tämän option yhteyttä luotaessa [BBJ92]. Muita optioilla neuvoteltavia ominaisuuksia ovat segmentin maksimikoko, jonka lähettäjä on valmis hyväksymään, sekä aikaleimaoptio, jota voidaan käyttää siirtoviiveen mittaamiseen. Aikaleima toimii myöskin sarjanumeron jatkeena nopeilla ja viiveisillä yhteyksillä suojaten laskurin pyörähtämiseltä.

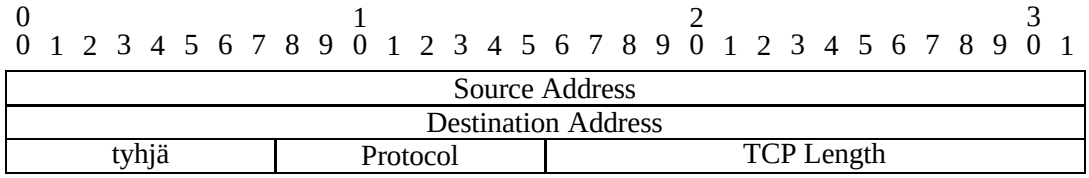
Yhteydelle varataan ikkunakokoa vastaava määrä vastaanottopuskuria. Puskurin koko on joko järjestelmäkohtainen vakio tai aseteltavissa yhteyskohtaisesti.

2.3.3.1 Datan vastaanotto

Linkkikerroksen kehyksen saapuminen järjestelmään, esimerkiksi Ethernet-verkkokortille, aiheuttaa keskeytyksen, jolloin käyttöjärjestelmän rutiini siirtää kehyksen tai antaa verkkokortille komennon kehyksen siirtämiseksi käyttöjärjestelmän puskureihin. Kehys hylätään, mikäli puskuritilaa ei ole.

Linkkikerroksen rutiini tunnistaa verkkokerroksen protokollan ja siirtää kehyksen sisällön tämän protokollan käsiteltäväksi. IP-kerros käsittelee IP-sähkeen seuraavasti [Bra89b]:

1. tarkistaa, että sähke on oikean muotoinen (pituus ja tarkistussumma täsmäävät), ja
2. tarkistaa, että se on osoitettu tähän järjestelmään ja



Kuva 2.8: TCP-lumeotsikko [Pos81c].

Taulukko 2.2: TCP:n tilamuuttujat [Pos81c].

Muuttuja	Selitys
SND.UNA	ensimmäinen lähetetty mutta kuittaamaton (send unacknowledged)
SND.NXT	seuraavana lähetettävä (send next)
SND.WND	lähetysikkuna (send window)
SND.UP	lähetettävän kiirellisen datan osoitin (send urgent pointer)
SND.WL1	viimeisimmän ikkunan päivityksen sarjanumero (segment sequence number used for last window update)
SND.WL2	viimeisimmän ikkunan päivityksen kuittausnumero (segment acknowledgment number used for last window update)
ISS	lähetyksen alkusarjanumero (initial send sequence number)
RCV.NXT	seuraava vastaanotettava (receive next)
RCV.WND	vastaanottoikkuna (receive window)
RCV.UP	vastaanotettavan kiirellisen datan osoitin (receive urgent pointer)
IRS	vastaanoton alkusarjanumero (initial receive sequence number)
SEG.SEQ	segmentin sarjanumero (segment sequence number)
SEG.ACK	segmentin kuittausnumero (segment acknowledgment number)
SEG.LEN	segmentin pituus (segment length)
SEG.WND	segmentissa oleva ikkunan koko (segment window)
SEG.UP	segmentissa oleva kiireellisen datan osoitin (segment urgent pointer)
SEG.PRC	segmentin suosituimmuus (segment precedence value)

3. käsittelee optiot ja

4. kokoaa sähkeen, mikäli tarpeen, ja

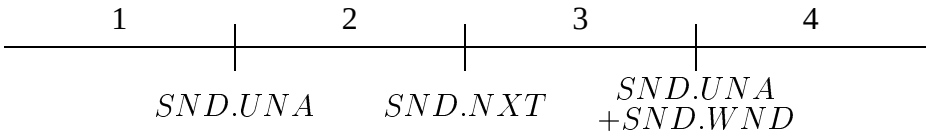
5. antaa viestin oikealle siirtokerroksen protokollalle.

Saatuaan IP-sähkeen sisällön eli TCP-segmentin otsikoineen IP-kerrokselta, TCP-kerros laskee tarkistussumman. Siihen lasketaan varsinaisen TCP-otsikon ja -segmentin lisäksi 96 bittiä pitkä kuvan 2.8 mukainen lumeotsikko, joka muodostuu lähde- ja kohdeosoitteista sekä protokolla- ja pituuskentästä. Pituuskentän arvo on TCP-otsikon ja datan pituuden summa.

Segmentti hylätään mikäli tarkistussumma on virheellinen. Samoin tarkistetaan, että

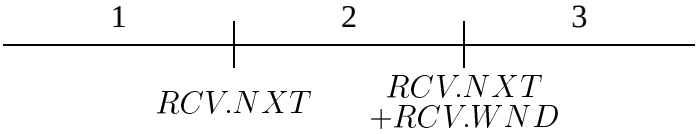
Taulukko 2.3: Laajennetut lähettäjän TCP:n tilamuuttujat [Jac88].

Muuttuja	Selitys
CWND	ruuhkaikkuna (congestion window)
SSTRESH	hitaan käynnistyksen kynnysarvo (slow-start threshold size)
RTO	uudelleenlähetysajastin (retransmit timer)



1 kuitatut sarjanumerot
2 lähetetyt mutta kuittaamattomat sarjanumerot (lähettäjän täytyy puskuroida)
3 sallitut sarjanumerot datan lähettämiseen
4 ei vielä käytettävissä olevat

(a) Lähetysavaruus



1 kuitatut sarjanumerot
2 sallitut sarjanumerot datan vastaanottoon
3 ei vielä käytettävissä olevat

(b) Vastaanottoavaruus

Kuva 2.9: TCP lähetys- ja vastaanottoavaruudet [Pos81c, s. 69].

segmentin sarjanumero on laillinen (kuva 2.9); ellei näin ole, lähetetään kuittaus takaisin laillisilla arvoilla ja segmentti hylätään. Loppuosa käsittelystä tapahtuu segmenttien mukaisessa järjestyksessä.

Seuraavaksi tarkistetaan lippujen sekä laskureiden arvot (taulukko 2.2) tässä järjestyksessä:

RST Yhteys lopetetaan eli kaikki puskuroitu tieto poistetaan ja sovellusohjelmalle tiedotetaan.

SYN Synkronointilipun havaitseminen tässä vaiheessa on virhe (pitäisi olla havaittu virheellisellä sarjanumerolla), joten yhteys suljetaan aivan kuin RST-lipun tapauksessa.

ACK Kuittauslaskuri ja lähetysikkuna päivitetään ($SND.UNA = SEG.ACK$) mikäli $SND.UNA < SEG.ACK \leq SND.NXT$. Lähettäjälle lähetetään kuittaus, ellei sitä ole aikaisemmin lähetetty.

URG Segmentissä on mukana kiireellistä dataa, esimerkiksi keskeytyskomento; ”Urgent Pointer” osoittaa kiireellisen datan alun. Tästä tiedotetaan sovellusohjelmalle, mikäli se on pyytänyt tiedon.

Segmentin data voidaan siirtää käyttäjän puskureihin, kunnes ne ovat täynnä tai segmentti on tyhjä. Käyttäjä voi saada vajaan puskurin, mikäli lähettäjä oli asettanut PSH-lipun.

Lähettävällä päällä ei ole enää lähetettävää, mikäli FIN-lippu on asetettu. Tällöin annetaan sovellusohjelmalle kaikki puskureissa oleva tieto ja ilmoitetaan yhteyden loppumisesta. Kaikki puskureissa oleva tieto lähetetään vastaanottajalle, kunnes sovellus sulkee yhteyden omaltakin puoleltaan. Yleisten ohjelmistorajapintojen ja TCP:n semantiikassa on eroavaisuuksia, mikä joudutaan huomioimaan sovellusten suunnittelussa. Tätä on selitetty tarkemmin kappaleessa 2.3.4 sivulla 26.

Vastaanottaja lähettää pelkkiä kuittaussähkeitä takaisin tietyn viiveen tai datamäärän jälkeen ellei sillä itsellään ole lähetettävää. Kuittaussähkeissä on ainoastaan päivitetty kuittausnumero ja datakenttä on tyhjä.

2.3.3.2 Datan lähettäminen

Sovellusohjelman antaessa lähetyskomennon `write()`-, `sendto()`- tai `sendmsg()`-käyttöjärjestelmäkutsulla, järjestelmä segmentoi viestin, lisää sarja- ($SEG.SEQ = SND.NXT$) ja kuittausnumeron ($SEG.ACK = RCV.NXT$) sekä vastaanottoikkunan ($SEG.WND = RCV.WND$) koon. Kutsun suoritus pysähtyy tai keskeytyy mikäli järjestelmä ei pysty tarjoamaan riittävää puskuria uudelleenlähetysten varalta: lähettäjän täytyy säilyttää lähetetty mutta kuittaamaton tieto puskureissaan ($SND.NXT - SND.UNA$, kuva 2.9(a)).

Lähetettäessä tarkistetaan, voidaanko segmentti lähettää eli onko $SEG.SEQ + SEG.LEN - 1 \leq SND.WND + SND.UNA$ [Pos81c]. Nykyisin useissa toteutuksissa käytetään lisäksi ruuhkaikkunaa ($CWND$), jolloin lähetysehto on kaavan 2.4 mukainen.

$$SEG.SEQ + SEG.LEN - 1 \leq min(SND.WND, CWND) + SND.UNA \quad (2.4)$$

Ruuhkaikkuna on aluksi yhden segmentin kokoinen ($SEG.LEN$) ja sitä kasvatetaan segmentin koolla jokaista saatua kuittausta vastaan, eli ruuhkaikkuna kasvaa eksponentiaalisesti. Saavutettuaan hitaan käynnistyksen[33] kynnysarvon ($SSTRESH$, aluksi 65 535) yhteys siirtyy ruuhkan välttämiseen ja kasvattaa ruuhkaikkunaa $SEG.LEN^2/CWND$:llä jokaista vastaanotettua kuittausta kohti eli ikkuna kasvaa lineaarisesti.

Ruuhkaikkuna asetetaan uudelleen yhdeksi segmentiksi, mikäli yhteys ei ole lähettänyt uudelleenlähetysajan (RTO) kuluessa. Yhteys ei tiedä nykyistä, mahdollisesti muuttunutta ruuhkatilannetta, joten sen tulee käyttäytyä kuin uusi yhteys [Jac88].

Uudelleenlähetysaika on kiertoviiveen (datan lähetyksen ja sen kuittauksen vastaanoton välinen aika) eksponentiaalisesti painotettu liukuva keskiarvo lisättynä nelinkertaisella keskihajonnalla. Näin RTO:n laskenta sopeutuu nopeasti kasvavaan ruuhkaan.

Yhteyden todetaan olevan ruuhkautunut, mikäli yksi tai useampi lähetetty segmentti

[33] slow start

katoaa. Segmenttien vaurioituminen on nykyisissä verkoissa hyvin epätodennäköistä ($\ll 1\%$), joten ruuhkatilanteen aiheuttama sähkeen hylkääminen on todennäköisin syy segmentin katoamiseen. Segmenttien katoaminen todetaan joko uudelleenlähetysajastimen laukeamisesta tai moninkertaisena kuittauksena. Ensimmäisessä tapauksessa suoritetaan hidas käynnistys. Jälkimmäisessä tapauksessa välistä on jäänyt segmentti tai useampi pois, jolloin hitaan käynnistyksen kynnysarvoksi asetetaan puolet käytössä olevasta ikkunakoosta $min(SND.WIN, CWND)$, kuitenkin vähintään kaksi segmenttiä ($2 \times SEG.LEN$). Kadonnut segmentti lähetetään ja ikkunakokoa suurennetaan saatujen kuittausten mukaan. Tällä nopealla uudelleenlähetyksellä[34] ja toipumisella[35] vältetään TCP-vuon katkeaminen [Ste97].

Nykyisen suosituksen [Bra89b] mukaan TCP-toteutus voi lähettää dataa, mikäli vähintään yksi seuraavista ehdoista on voimassa.

- Maksimikokoinen segmentti voidaan lähettää.
- Sovellus työntää[36] dataa ja kaikki jonossa oleva data voidaan lähettää.
- Vähintään Fs:n suuruinen osa suurimmasta ikkunakoosta voidaan lähettää.[37]
- Sovellus työntää dataa ja aikavalvonta (0,1 – 1,0 s) laukeaa.

Jonkun edellä olevista ehdoista täytyttyä TCP-kerros antaa segmentin IP-kerrokselle, joka [Bra89b]:

1. täyttää kentät, joita siirtokerros ei asettanut, ja
2. valitsee, mille linkille se lähetetään (reititys) ja
3. tarvittaessa jakaa viestin osiin ja
4. antaa paketit linkkikerroksen ajureille.

[34] fast retransmit
[35] fast recovery
[36] push
[37] Fs:n suositeltu arvo on 1/2.

Linkkikerroksen protokolla liittää omat otsikkotietonsa alkuun ja loppuun sekä mahdollisesti laskee tarkistussumman ja lähettää kehyksen verkkoon. Yleensä linkkikerros ei pysty mitenkään varmistamaan, menikö kehys perille, eikä se ole IP-verkoissa tarpeenkaan.

2.3.4 Yhteyden purku

Yhteyden voi purkaa kumpi osapuoli tahansa. Yhteyden purkamiseen liittyy useissa järjestelmissä semanttinen ero yleisen `close()`-kutsun ja TCP:n toiminnan kannalta. Normaalisti `close()`-kutsun jälkeen ei voida enää lukea eikä kirjoittaa tiedostoon tai yhteydelle. TCP:ssä yhteyden sulkeminen sulkee vain lähetyksen, vastaanotto on yhä mahdollista. Tämä vaatii ylempien kerrosten sovelluksia ottamaan tämän huomioon siten, että viimeksi lähettävä osapuoli sulkee yhteyden.

Vastaanottaessaan FIN-lipulla varustetun segmentin, TCP-kerros toimittaa puskuroidun datan sovellukselle kuten PSH-lipulla ja ilmoittaa sovellukselle yhteyden sulkeutumisesta sekä lähettää kuittauksen takaisin. Kun yhteyden sulkemisen vastaanottanut sovellus päättää sulkea yhteyden omaltakin puoleltaan, TCP-kerros lähettää FIN-lipulla varustetun segmentin vastapäälle ja odottaa kuittausta. Sulkemista pyytänyt osapuoli pitää yhteyttä kuittauksen lähettämisen jälkeen varattuna vielä kaksi kertaa segmentin suurimman elinajan verran ($2 \times MSL$) eli 4 minuuttia viimeisen kuittauksen lähettämisen jälkeen kuittauksen katoamisen varalta.

2.4 Levyoperaatiot

Levyjärjestelmä samoin kuin verkkojärjestelmä muodostuu useista kerroksista. Kerroksien tarkoituksena on peittää alemman tason erot ylemmiltä kerroksilta, jolloin osien toimintaa voidaan muuttaa ilman että se aiheuttaisi muutostarvetta kaikille kerroksille.

Kerroksellisuutta on havainnollistettu kuvassa 2.10. Fyysiset laitteet suorittavat varsinaisen tallennuksen ja muodostuvat yleensä mekaniikasta, elektroniikasta ja omasta

Kuva 2.10: Levy I/O-järjestelmän kerroksellisuus [SG94, s. 384].

ohjelmistostaan. I/O-ohjaus muuntaa loogiset komennot ”lue lohko numero 123” laitteiston ymmärtämään muotoon ja huolehtii tähän liittyvistä keskeytyksistä samoin kuin piilottaa laitteiston epäideaalisuudet [Tan92].

Tiedostojen järjestelymoduli tietää tiedostojen sijoittelun levyllä ja perustiedostojärjestelmän kautta ohjaa laitteistoa. Looginen tiedostojärjestelmä huolehtii linkityksestä nimien ja järjestelymodulin käyttämien numeroiden välillä huolehtien samalla myös tiedostojen suojauksesta.

Kirjastojen ja systeemikutsujen kautta sovellusohjelmille näkyy yhtenäinen rajapinta, joka mahdollistaa sovellusohjelmien siirron järjestelmästä toiseen.

Levyoperaatiot ovat usein aikaa vieviä mekaanisesta toiminnasta johtuen. Levyltä lukemisessa kuluu aikaa seuraavasti:

- Lukupään kohdistaminen oikealle uralle: nykyaikaisissa levyissä keskimääräinen aika on alle 10 ms.
- Oikean kohdan tulo lukupään alle: tämä riippuu levyn pyörimisnopeudesta. Nykyiset levyt pyörivät 3 600 – 10 000 kierrosta minuutissa eli keskimäärin aikaa kuluu 16 – 6 ms.
- Luku levyltä: lukunopeus riippuu tallennustiheydestä ja pyörimisnopeudesta, tyypillisesti yli 12 megatavua sekunnissa.

- Siirto levyltä ohjaimen muistiin. Levyväylien siirtonopeudet ovat nykyään vähintään 10 megatavua sekunnissa.
- Siirto ohjaimelta käyttöjärjestelmän puskureihin, nykyisten koneiden sisäiset väylät ovat nopeudeltaan noin 133 megatavua sekunnissa.
- Siirto käyttöjärjestelmältä käyttäjän puskureihin: 180 MHz:n Pentium Pro-prosessori siirtää noin 50 megatavua sekunnissa kolmen megatavun suuruisen lohkon välillä. Siirtonopeus on 160 megatavua sekunnissa 64 kilotavun lohkojen välillä, mikä osoittaa välimuistin merkityksen. Välimuistista ei kuitenkaan ole hyötyä laitteilta kopioitaessa.

Levyjärjestelmän suorituskykyä pyritään parantamaan puskuroinnilla eri vaiheissa: on todennäköistä, että seuraava luku kohdistuu seuraavana olevaan tietoon. Siirto on nopeata verrattuna hakuun, niinpä kannattaa lukea hiukan enemmän kuin sovellus pyysi. Puskureita on tyypillisesti levyn omassa ohjaimessa, levyohjaimella ja käyttöjärjestelmässä.

Suorituskykyä voidaan parantaa myös jättämällä siirtovaiheita pois ja lisäksi käyttämällä suoraa muistiinsiirtoa (DMA), jolloin prosessorin kuormitus pienenee, mikä on hyödyllistä moniajojärjestelmissä. Optimaalisessa järjestelmässä levyohjain siirtää tietoa suoraan levyltä käyttäjän puskuriin. Virtuaalimuisti ja monet laitteiston epäideaalisuudet tekevät tästä kuitenkin usein vaikeasti toteutettavan, niinpä DMA-siirtoa käytetään yleensä vain käyttöjärjestelmän puskureihin.

Puskurointia vaatii myös se, että sovellusohjelmat voivat lukea minkä suuruisia tietomääriä mistä kohtaa tahansa. Levyjärjestelmä pystyy kuitenkin siirtämään vain tietyn suuruisia määriä: ainoastaan kokonaisia lohkoja, yleensä rajallisen määrän.

Levyoperaatio ei välttämättä aiheuta prosessin pysäyttämistä odottamaan, mikäli tarvittava tieto löytyy suoraan järjestelmän puskureista, jolloin se voidaan palauttaa välittömästi. Prosessia voidaan palvella hyvinkin nopeasti, mikäli levyjärjestelmästä ei ole useita kilpailevia hakuja.

2.5 Liikenteen karakteristen ominaisuuksien määrittäminen

Dataliikenteen verkolle aiheuttama kuormitus ei riipu pelkästään liikenteen keskimääräisestä nopeudesta tai huippunopeudesta. Perinteisissä puhelinverkoissa kaistanvaraus on ollut kiinteä, halutun huippunopeuden suuruinen. Tämä on kuitenkin resurssien vajaakäyttöä, mikäli liikenteen keskinopeus on pienempi kuin huippunopeus. Ollaankin siirtymässä jaetun kaistan käyttöön, jossa voidaan tilastollisen kanavoinnin avulla välittää enemmän liikennettä kuin eri liikennelähdeiden huippunopeuksen perusteella väylälle mahtuu.

Tilastollisen kanavoinnin seurauksena liikennettä joudutaan kuitenkin joissain tapauksissa hylkäämään. Hylättävän liikenteen osuus kokonaisliikenteestä olisi kuitenkin pidettävä mahdollisimman pienenä, koska häviöt aiheuttavat häiriöitä joko käyttäjän kokemassa palvelunlaadussa tai suorituskyvyssä sovelluksesta riippuen [IT93b, KS94, LST95].

Dataliikenteen osalta merkittävä tekijä on liikenteen purskeisuus. Dataliikenne yleensä on tyypiltään on-off-liikennettä: liikennettä on jonkin aikaa maksiminopeudella ja tämän jälkeen on taas aika, jona liikennettä ei ole. Tässä työssä käytetään kahta menetelmää purskeisuuden arviointiin: ATM-verkossa käytössä olevia liikenneparametreja sekä saapumisten lukumäärän hajontaindeksiä.

2.5.1 ATM-verkon liikenneparametrit

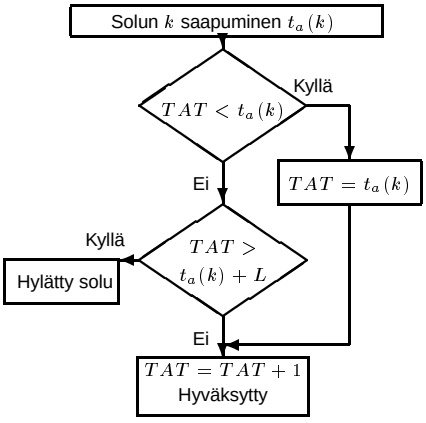
Liikenneparametrit (taulukko 2.4) määräävät liikenteen ominaisuuksia ja soveltuvat erityisesti vaihtuvanopeuksisen liikenteen (VBR) kuvaamiseen.

Liikenneparametrien tulkinta ja valvontaprosessi perustuu vuotavan ämpärin nimellä tunnettuun GCRA-algoritmiin [IT93a, IT94]. Algoritmi on toiminnallisesti liukuvan ikkunan laskenta-algoritmi.

Algoritmin toiminta on esitetty kuvassa 2.11. Vertaamalla solun tuloaikaa $t_a(k)$ teo-

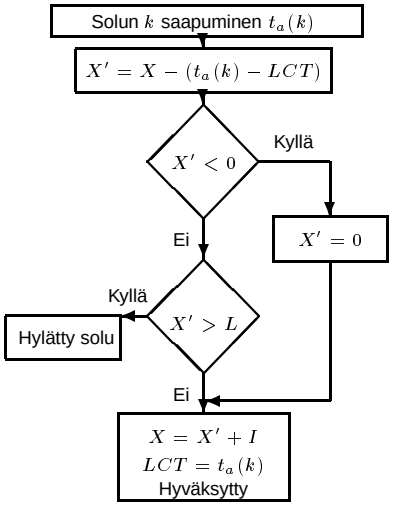
Taulukko 2.4: ATM-verkon liikenneparametrit [IT93a, IT94].

Lyhenne	Yksikkö	Määrittää	Selitys
PCR	solua/s	$T_p = \frac{1}{PCR}$	huippusolunopeus (Peak Cell Rate)
SCR	solua/s	$T_s = \frac{1}{SCR}$	keskimääräinen solunopeus (Sustainable Cell Rate)
BT	s	$\tau_s = BT$	pursketoleranssi (Burst Tolerance)
CDVT	s	$\tau = CDVT$	solun siirtoviiveen varianssi (Cell Delay Variation Tolerance)



TAT teoreettinen saapumisaika
Alussa $TAT = 0$.

(a) Virtuaalijärjestely

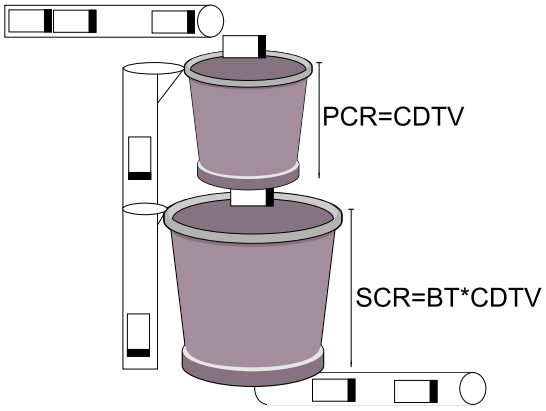


X solun saapumisaika
X' ajan kasvatusarvo
LCT raja-arvo
Alussa $X = 0$ ja $LCT = t_a(k)$.

(b) Vuotavan ämpärin algoritmi

$t_a(k)$ solun saapumisaika
I ajan kasvatusarvo
L raja-arvo

Kuva 2.11: GCRA-algoritmin toiminta [IT93a, IT94].



Kuva 2.12: Kahden vuotavan ämpärin järjestelmä.

reettiseen tuloaikaan TAT päätetään, onko kyseinen solu liikennesopimuksen mukainen [IT94].

Tarkastelu on kaksivaiheinen, mikäli yhteydellä sallitaan purskeita eli keskimääräistä suurempaa solunopeutta ($PCR > SCR$). Ensin tarkastellaan huippunopeuden sopimuksenmukaisuutta algoritmilla GCRA(T_p, τ) eli solujen saapumishetkille sallitaan $CDVT$:n mukainen soluviiveen vaihtelu. Ensimmäisen ehdon täytyttyä suoritetaan vertailu keskinopeuden ja pursketoleranssin suhteen arvolla GCRA(T_s, $\tau_s + \tau$).

Tätä toimintaa voidaan havainnollistaa kuvassa 2.12 esitetyllä kahden vuotavan ämpärin mallilla. Ylemmän ämpärin tilavuus vastaa suurinta sallittua soluviiveen vaihtelua ja ämpäri tyhjenee suurimmalla sallitulla huippunopeudella. Alemman ämpärin tilavuus vastaa suurinta purskekokoa ja se tyhjenee sallitulla keskinopeudella.

2.5.2 Saapumisten lukumäärän hajontaindeksi

Purskeisuuden määrittämiseen voidaan käyttää saapumisten lukumäärän hajontaindeksiä[38] (IDC) [HL86, RMV96]. Tämä määritellään saapumisten määrän (N_t) varianssin ja keskiarvon suhteena aikavälillä t eli

$$I(t) = \frac{Var(N_t)}{E(N_t)}. \tag{2.5}$$

[38]Index of Dispersion for Counts

Tasaiselle liikenteelle pätee $I(t) = 0$, koska saapumisten määrä on vakio (varianssi nolla) kaikilla aikaväleillä. Poissoniselle liikenteelle $I(t)$ on joko vakio tai lähestyy vakioarvoa nopeasti [Vid96]. Itsesimilaariselle liikenteelle $I(t)$ on monotonisesti kasvava [LTWW94].

Saapumisten lukumäärän hajontaindeksiä havainnollistetaan usein piirtämällä käyrä $I(t)$ eri aikaväleille t. Kirjallisuudessa suositellaan, että t on enintään 20 % kokonaisajasta [Lin96]. Tulokset vääristyvät, jos aika t on suuri verrattuna kokonaisaikaan. Menetelmää voidaan pitää ”insinöörimäisenä”: käyrästä nähdään helposti liikenteen vaihtelevuus eri aikatasoilla [LTWW94].

Luku 3

Käytännön käyttöjärjestelmätoteutukset

Tässä työssä tutustutaan tarkemmin kahteen UNIX-murteeseen, joiden lähdekoodi on kokonaan tai lähes kokonaan[1] julkisesti saatavana, mikä mahdollistaa niiden tutkimisen ja muokkaamisen.

LINUX-käyttöjärjestelmän osalta pääasiallisena lähteenä on ollut version 2.0.25 lähdekoodi. Lisäksi käytetään lähdettä [BBD+96], joka perustuu versioon 1.2; tämän työn kannalta mielenkiintoiset osat ovat kuitenkin muuttuneet varsin paljon versioiden välillä. 4.4BSD:n osalta pääasiallisena lähteenä on ollut [MBKQ96], jonka lisäksi on tutkittu FREEBSD-2.2-jakelun lähdekoodia.

[1] 4.4BSD:stä on kaksi versiota: 4.4BSD-Encumbered, joka sisältää lisenssin vaativaa lähdekoodia AT&T UNIX/32V Time-Sharing System Version 1.0:sta (täydellinen lähdekoodihistoria), ja 4.4BSD-Lite, joka ei vaadi lisenssiä, koska lisenssin vaativat osat ovat kirjoitettu uudelleen eikä alkuperäistä ohjelmakoodia ole mukana levityksessä.

Taulukko 3.1: 4.4BSD -järjestelmän prioriteettitasot [MBKQ96].

Taso	Arvo	Kuvaus
PSWP	0	prosessien sivutus
PVM	4	prosessi odottaa muistia vapautuvaksi
PINOD	8	prosessi odottaa tiedoston tietoja
PRIBIO	16	prosessi odottaa levytoimintoja
PVFS	20	prosessi odottaa kernel-tason tiedostolukkoa
PZERO	22	perustaso
PSOCK	24	prosessi odottaa liikennöintipistettä
PWAIT	32	prosessi odottaa lapsiprosessin loppumista
PLOCK	36	prosessi odottaa käyttäjätason tiedostolukkoa
PPAUSE	40	prosessi odottaa signaalia
PUSER	50	perustaso käyttäjätilan prosesseille

3.1 Vuorottelun toiminta

3.1.1 4.4BSD

Berkeley Software Distribution'in uusin versio UNIX-käyttöjärjestelmästä käyttää interaktiivista käyttöä suosivaa prioriteettijärjestelmää [MBKQ96]. Järjestelmässä on 128 eri prioriteettitasoa, joista 50 korkeinta on varattu taulukon 3.1 mukaan prosesseille, jotka odottavat vastetta tai resurssia.

Eri tapahtumien odottaminen on resurssista riippuen joko keskeytettävää tai keskeyttämätöntä, mikäli tapahtuma on tyypillisesti lyhyt. Käyttöjärjestelmäkutsussa keskeytettävä odottaminen voi keskeytyä signaaliin, jolloin käyttäjälle palautetaan virhe (`EINTR`) tai kutsun suoritus käynnistetään uudestaan (`ERESTART`).

4.4BSD-järjestelmän prosessien vuorottelu perustuu monitasoisiin palautejonoihin. Järjestelmä valitsee ajettavaksi korkeimman prioriteetin jonossa olevan ensimmäisen prosessin. Käytettyään aikaviipaleensa (0,1 s) loppuun, siirretään se saman ajojonon perään. Prosessia ei siirretä takaisin ajojonoihin vaan tapahtuman odotusjonoihin, mikäli prosessi jää suorituksen aikana odottamaan jotain tapahtumaa

Aikaviipaleen pituus on aikanaan löydetty kokeellisesti ja on pysynyt muuttamattomana 1980-luvun alusta lähtien. Vaikka nykyinen käyttäjä odottaa parempaa vastetta kuin suurkoneen käyttäjä viime vuosikymmenellä, on parempi suorituskyky lyhentänyt ajo-

jonoja ja siten poistanut tarpeen lyhyemmistä aikaviipaleista.

Prosesseja siirretään jonojen välillä niiden prioriteettimuutosten perusteella. Jonoja on 32 kappaletta[2] eli jokaisessa jonossa on neljä eri prioriteettitasoa; jonon sisällä nämä neljä tasoa ovat samanarvoisia. Ajettavaa prosessia vaihdetaan, mikäli joku toinen prosessi tulee ajokelpoiseksi korkeammalla prioriteetilla ja ajossa oleva prosessi on käyttäjätilassa. Prosessin vaihto tapahtuu, kun prosessi palaa käyttäjätilaan., mikäli ajettava prosessi on ydintilassa eli suorittamassa käyttöjärjestelmän koodia,

Järjestelmä suosii interaktiivisia prosesseja nostamalla prosessin prioriteettia, mikäli se on odottanut I/O:ta yli sekunnin, ja laskee vastaavasti prosessin prioriteettia, mikäli prosessi on käyttänyt paljon CPU-aikaa.

Prosessin vuorotteluprioriteetti muodostuu kahdesta muuttujasta: p_estcpu ja p_nice. Ensimmäinen antaa arvion prosessin viimeaikaisesta CPU:n käytöstä ja toinen on käyttäjän aseteltavissa oleva kohteliaisuusarvo välillä -20 – 20, nollan ollessa oletusarvo[3]. Prosessin prioriteetti lasketaan aina neljän kellokeskeytyksen eli noin 40 ms:n jälkeen seuraavasti:

$$p\_usrpri = PUSER + \left[\frac{p\_estcpu}{4}\right] + 2 \times p\_nice \tag{3.1}$$

Prioriteettiarvo asetetaan arvoon PUSER (taulukko 3.1) mikäli se on tätä alempi.[4] Vastaavasti suurimman prioriteettiarvon ylittävät arvot asetetaan tähän suurimpaan arvoon (127). CPU:n hyödyntämisastetta (p_estcpu) kasvatetaan jokaisella kellokeskeytyksen (10 ms välein) hetkellä, jolloin prosessi on ajossa. Lisäksi arvoa korjataan kerran sekunnissa eksponentiaalisesti painotetulla liukuvalla keskiarvolla, jonka toiminta riippuu järjestelmän kuormituksesta kaavan 3.2 mukaan.

$$p\_estcpu = \frac{2 \times load}{2 \times load + 1} \times p\_estcpu + p\_nice \tag{3.2}$$

[2] Alkuperäisessä laitteistossa 32-bittisen bittijonon käsittely oli tehokasta: siksi jonoja on neljännes prioriteettitasoista.

[3] Normaalikäyttäjä voi ainoastaan laskea prioriteettia eli antaa nollaa suurempia p_nice-arvoja.

[4] Prioriteetti korkeampi

Tässä $load$ on näytteistetty keskiarvo järjestelmän ajojonon pituudesta viimeisen minuutin aikana eli ajokelpoisten prosessien keskimääräinen määrä. Kaava antaa 6 sekuntia 90 % (F_p) unohdusajaksi järjestelmän kuormituksella 1,0 ja 30 sekuntia kuormituksella 6,1 eli unohdusaika noudattaa kaavaa 3.3.

$$t_{decay} = \frac{ln(1 - F_p)}{ln\left(\frac{2 \times load}{2 \times load + 1}\right)} \tag{3.3}$$

Prosessin prioriteetti lasketaan uudelleen, kun prosessin CPU:n hyödyntämisaste on 4. Ajojonoja tarkastellaan aina 100 ms välein, jolloin yksi prosessi voi saada tämän pituisen yhtenäisen ajoajan.

Järjestelmä ei laske prioriteetteja odottaville prosesseille. Tämä vähentää oleellisesti vuorottelun aiheuttamaa tehohukkaa, mikä näkyy esimerkiksi kuvasta 2.3 (s. 13). Yli sekunnin odottaneelle prosessille järjestelmä laskee uuden prioriteettiarvon kaavan 3.1 mukaan käyttäen hyödyntämisasteena kaavan 3.2 asemasta kaavaa 3.4, jossa huomioidaan prosessin odottamisaika $p\_slptime$ sekuntia.

$$p\_estcpu = \left[\frac{2 \times load}{2 \times load + 1}\right]^{p\_slptime} \times p\_estcpu \tag{3.4}$$

3.1.2 LINUX 2.0.25

LINUX tukee nykyisessä muodossaan normaalin vuorottelun lisäksi myös reaaliaikasovelluksissa hyödyllisiä kiertävän vuoron (RR) ja ”ensin tullut, ensin palveltu” FCFS -algoritmeja[5] FCFS-algoritmia käyttää muun muassa sivutuksesta huolehtiva kswapd. Prioriteetin lukuarvon tulkinta on päinvastainen kuin muuten tässä esityksessä ja 4.4BSD:ssä eli korkeinta prioriteettia vastaa suurin lukuarvo.

Aluksi siirretään mahdollisesti ajossa ollut RR-vuorottelua käyttävä prosessi listan loppuun, mikäli tämä oli käyttänyt aikaviipaleensa loppuun. Tämän jälkeen suoritetaan mahdolliset keskeytysten työjonot. Keskeytysrutiineissa tehdään vain välttämätön, koska ne täytyy suorittaa mahdollisimman nopeasti. Työjonoissa tehdään loppuosa

[5] LINUX käyttää termiä FIFO: ”First in, First Out”. Toiminta on kuitenkin täsmälleen sama.

työstä, esimerkiksi verkosta tulleiden kehysten käsittely ylemmillä protokollilla. Työjonoja suoritetaan mm. vuorottelurutiinin alussa, ja niissä suoritettavat tehtävät voivat muuttaa prosessien tilaa.

Jos viimeksi ajossa olleen prosessin tila on `TASK_INTERRUPTIBLE`, tarkistetaan myös, onko se saanut jonkun signaalin tai onko odotusaika kulunut, jolloin sen tilaksi muutetaan `TASK_RUNNING`. Muuten se poistetaan ajojonosta. Tämän jälkeen käydään ajojonoa läpi. Jokaiselle prosessille lasketaan "hyvyys". Korkeimman hyvyysarvon saanut prosessi on seuraavana ajovuorossa.

Reaaliaikaprosessit (FCFS, RR) saavat korkeimman hyvyysarvon, joka on $1000 + rt\_priority$. Jokaisella prosessilla on laskuri, joka määrää kuinka monta kellokeskeytystä (10 ms) sillä on ajoaikaa. Tämän laskurin arvo on prosessin hyvyysarvo. Viimeksi ajossa olleelle prosessille (mikäli laskuri oli suurempi kuin nolla) annetaan lisäksi hieman korkeampi hyvyysarvo, jotta samanarvoisilla prosesseilla vältettäisiin prosessin vaihtamisesta aiheutuva tehohäviö (kuva 2.3, s. 13) [FK96].

Moniprosessorijärjestelmässä hyvyysarvoon vaikuttaa lisäksi prosessori, jolla prosessia on viimeksi ajettu. Prosessi voidaan myöskin sitoa tiettyyn prosessoriryhmään, jolloin toisen ryhmän prosessorille prosessin hyvyysarvo on -1000 eli prosessia ei oteta koskaan ajoon. Sama arvo palautetaan myös toisen prosessorin ajaessa parhaillaan prosessia.

Prosessien laskurit lasketaan uudelleen kaavan 3.5 mukaan, mikäli minkään prosessin hyvyysarvo ei ole positiivinen eli kaikkien prosessien laskurit ovat nollautuneet.

$$counter = \frac{counter}{2} + priority \tag{3.5}$$

Käyttäjä voi asettaa prosessille kohteliaisuusarvon ($nice$) välille -20 – 20. Negatiiviset arvot ovat ainoastaan pääkäyttäjän käytettävissä, kuten 4.4BSD:ssä. Prosessin prioriteetti lasketaan kaavalla 3.6 ja muunnetaan tästä aikaviipaleiksi kaavalla 3.5.

$$priority = 20 - \frac{nice * 2 + 1}{2} \tag{3.6}$$

Normaalikäyttäjällä prosessin maksimi yhtäjaksoinen ajoaika on siis 200 ms, pääkäyttäjällä 400 ms.[6] Kohteliaisuusarvolla 20 prioriteetti menee nollaan, joka kuitenkin korjataan arvoon yksi.

Ydinkoodia suorittavaa prosessia ei keskeytä toinen ydinkoodia suorittava prosessi, vaan prosessin vaihto voi tapahtua ainoastaan vapaaehtoisella nukahtamisella, esimerkiksi jonkin resurssin tai tapahtuman odottamisella [Cox95]. Tämä yksinkertaistaa järjestelmän toimintaa mutta heikentää moniprosessorijärjestelmän suorituskykyä.

3.2 TCP lähetysprosessi

3.2.1 4.4BSD

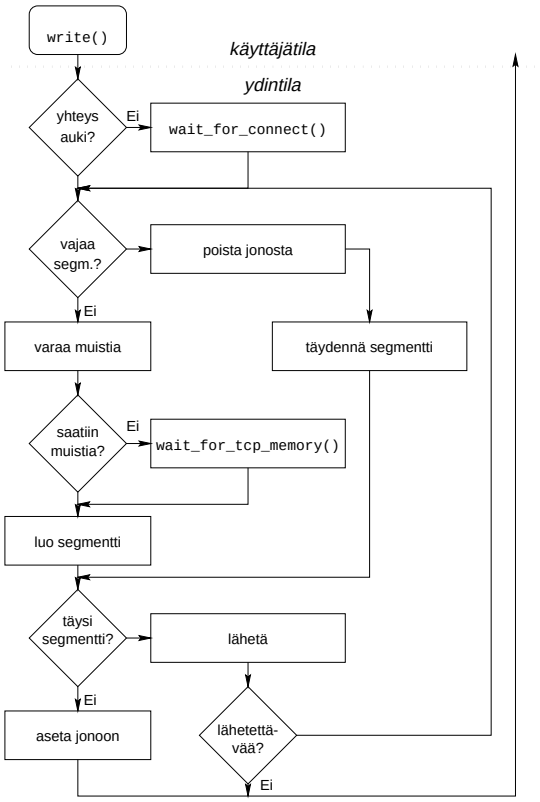
Prosessin suoritettua `write()`-, `sendto()`- tai `sendmsg()`-kutsun, kukin kutsu luo tarvittavat tietorakenteet ja kutsuu `sendit()`-rutiinia. Kaikki kolme kutsua voitaisiin toteuttaa kirjastorutiineina `sendmsg()`-kutsulla. Syynä kolmeen erilaiseen kutsuun on paras tehokkuus kussakin käyttösovelluksessa.

`sendit()`-rutiini kokoaa kaikki kutsuparametrit käyttöjärjestelmän muistiin varsinaista dataa lukuunottamatta ja kutsuu sen jälkeen `sosend()`-rutiinia. Se käsittelee useimmat liikennöintipistetason optiot ja tarkistaa liikennöintipisteen tilan: esimerkiksi virhetapauksissa se raportoi virheistä eikä yritä siirtoa. Prosessi asetetaan odottamaan, mikäli kaikki lähetettävä data ei mahdu liikennöintipisteen lähetyspuskuriin. Rutiini kopioi datan käyttäjän puskureista mbuf-rakenteisiin ja kutsuu siirtokerroksen protokollaa tekemään varsinaisen siirron.

3.2.2 LINUX 2.0.25

Prosessi kirjoittaa verkkoon (`write()`- tai `sendmsg()`-funktiolla). Tunnistettaessa kirjoituksen kohdistuvan TCP-liikennöintipisteeseen tarkistetaan ensin, että yhteys on

[6] Laskenta tapahtuu kokonaislukulaskentana hiukan eri tavalla, joten lukujen katkaisu vaikuttaa toisin kuin suoraan kaavassa 3.6.



Kuva 3.1: TCP-kirjoituksen vuokaavio LINUX'ssa.

auki. Prosessi asetetaan odottamaan kuittausta vastapäästä, mikäli yhteys ei ole vielä auki.

Lähetettävälle TCP-segmentille varataan muistia, johon kopioidaan tieto käyttäjän puskurista. Ellei lähetettävää dataa varten saada varattua muistia, jäädään odottamaan ja kutsutaan vuorottelurutiinia. Lähetyskelpoinen segmentti lähetetään IP-kerrokselle, joka liittää siihen tarvittavat IP ja MAC-otsakkeet.

Lähettäminen on mahdollista, mikäli lähetysikkunassa ja ruuhkaikkunassa on tilaa. LINUX käyttää hidasta käynnistystä, joten ruuhkaikkunan koko ($CWND$, taulukko 2.3, s. 22) on yksi mikäli yhteys on ollut lähettämättä uudelleenlähetysajastimen (RTO) ajan verran. Linux käyttää RTO:n minimiarvona aikaa 0.2 s.

Vajaa segmentti jää odottamaan yhden sekunnin ajaksi. Ellei tänä aikana tulee lisää lähetettävää, vajaa segmentti lähetetään sellaisenaan. Jos tietoa tulee lisää, vajaa segmentti poistetaan lähetysjonosta, uusi tieto liitetään segmenttiin ja sekunnin odotus käynnistetään uudelleen.

Kuva 3.2: I/O-järjestelmän kerrokset 4.4BSD-käyttöjärjestelmässä [MBKQ96, s. 194].

3.3 Levylukuprosessi

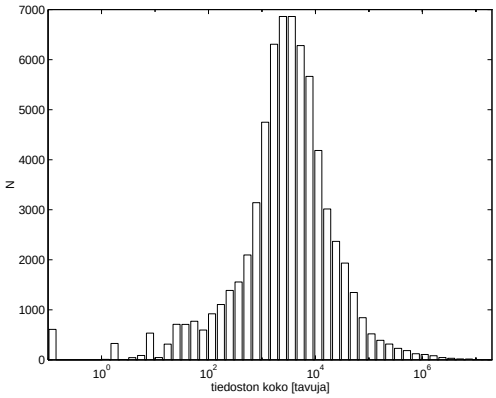
3.3.1 4.4BSD

Aikaisemmissa BSD-versioissa tiedostojärjestelmä perustui informaatiosolmuihin (inode). Nykyään kuitenkin käyttöjärjestelmän tulee tukea erilaisia tiedostojärjestelmiä, mm. verkkotiedostojärjestelmiä (kuva 3.2). Tiedoston paikallisella levyllä määrittävästä tietueesta siirryttiinkin näennäisiin informaatiosolmuihin (vnode), joissa osa tiedoista on yhteisiä ja loppu tietueesta on tiedostojärjestelmäriippuvaa. Tiedostojärjestelmän liitostietueeseen kuuluva funktiokutsutaulukkoa käytetään tiedostojärjestelmäriippuvien tietojen käsittelyyn ja tiedostotoimintoihin [MBKQ96, s. 219].

Levypuskurien määrä 4.4BSD-järjestelmässä on vakio ja määräytyy järjestelmän käynnistyksessä keskusmuistin määrän perusteella 100 – 1000 puskuriksi. Jokaiselle puskurille on varattu 64 kilotavua virtuaalimuistia. Näin suuri todellinen puskurimuisti on useimmille tiedostoille tarpeeton: UNIX-järjestelmissä tiedostokokojen mediaani on alle 2 kilotavua ja keskiarvo 22 kilotavua [Irl93]. Koneen `keskus.hut.fi` tiedostokokojen jakauma on esitetty kuvassa 3.3. Mediaani on 3128 tavua ja keskiarvo on 25,7 kilotavua, suurin tiedosto on 18,4 megatavua.

Tiedostojen kokojakaumasta johtuen kullekin puskurille varataan fyysistä muistia aluksi yksi sivu (4 kilotavua). Puskurit ovat yhdessä neljästä listasta:

lukittu Puskuri on pysyvästi varattu, esimerkiksi tiedostojärjestelmän superlohkolle.



Kuva 3.3: Tiedostokokojen jakauma koneessa `keskus.hut.fi` 5.5.1997.

LRU Käytössä olevat puskurit. Kun puskuri on varattu tai sitä on käytetty, se asetetaan listan viimeiseksi. Ensimmäisenä tässä listassa oleva puskuri otetaan, kun tarvitaan puskureita ja seuraavat kaksi listaa ovat tyhjiä.

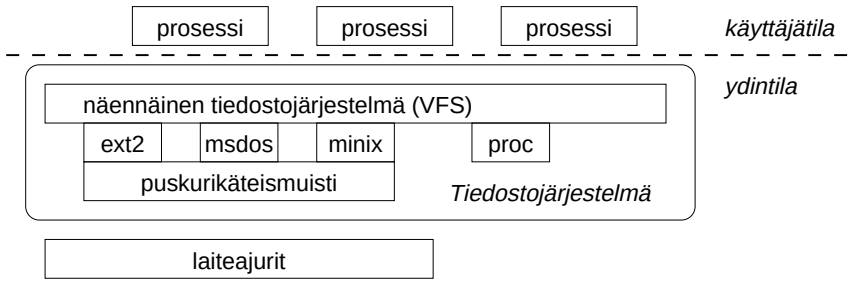
vanhat Tässä listassa ovat puskurit, jotka ei ole vielä osoittaneet olevansa käyttökelpoisia, kuten esimerkiksi etukäteisluetut puskurit. Myöskin puskurit, joiden sisältö on kokonaan luettu tai kirjoitettu, ovat tässä listassa. Puskureita voidaan liittää joko listaan alkuun, jolloin se käytetään todennäköisesti heti, tai loppuun, jolloin se säilyy listassa kauimmin.

tyhjät Puskurit, joissa ei ole yhtään fyysistä muistia, ovat tässä listassa. Nämä tulevat käyttöön silloin, kun joku käytössä oleva puskuri varataan pienemmällä muistimäärällä ja muistia vapautuu.

3.3.2 LINUX 2.0.25

LINUX-käyttöjärjestelmän tuki useille eri tiedostojärjestelmille on yksi sen suosion salaisuuksista: versiossa 2.0.29 mukana on 17 yleistä tiedostojärjestelmää. Näiden menestyksellinen käyttö on mahdollista kerrosrakenteisuuden (kuva 3.4) ansiosta: näennäinen tiedostojärjestelmä[7] (VFS) tarjoaa sovellusohjelmille yhtenäisen rajapinnan eri tiedostojärjestelmiin. Eri tiedostojärjestelmät sijoittuvat VFS:n ja puskurivälimuistin välille, jolloin kaikki tiedostojärjestelmät voivat hyödyntää samaa välimuistia. LINUX

[7] Virtual File System



Kuva 3.4: Tiedostojärjestelmän kerrokset LINUX-käyttöjärjestelmässä [BBD+96, s. 149].

käyttää kaiken vapaana olevan keskusmuistin puskurointiin: tämä tehostaa toimintaa verrattuna kiinteään puskurimuistitilaan.

Yhtenäisen laiteajurirajapinnan ansiosta jokainen tiedostojärjestelmä voi toimia käytännöllisesti katsoen millä tahansa lohkopohjaisella laitteella. Luonnollisestikaan kaikki tiedostojärjestelmät eivät ole käyttökelpoisia tai järkeviä kaikilla fyysisillä laitteilla.

Puskurimuisti toimii ”pisimpään käyttämättä ollut”-periaatteella[8] (LRU). Sovellusohjelmat yleensä lukevat dataa levyltä peräkkäisessä järjestyksessä: yhdellä lukuoperaatiolla kannattaa lukea hiukan enemmän kuin mitä sovellusohjelma pyytää. IDE-levyillä etukäteisluettava määrä on 8 lohkoa eli 4 kilotavua.

Lohkon lukeminen IDE-levyltä tapahtuu suorittamalla lukupyyntö ja asettamalla keskeytysrutiini levykeskeytykselle. Useimmat IDE-levyt siirrättävät datan prosessorilla[9] (PIO), joten keskeytysrutiini kopioi datan levyohjaimelta. Muutamat emolevy-kiintolevy-yhdistelmät osaavat käyttää myös suoraa muistiin siirtoa[10] (DMA), jolloin keskeytysrutiini huolehtii ainoastaan siirron kuittauksesta. Kun lukupyyntö halutulle lohkolle ja mahdollisesti seuraaville on asetettu, ohjelman suoritus palaa odottamaan lukupyynnön valmistumista. Ennen kuin prosessi luopuu ajovuorostaan kutsumalla `schedule()`-funktiota, se ajaa levykeskeytysten jälkityöjonon[11] ja tämän jälkeen tarkistaa, onko sen pyytämä luku jo suoritettu. Prosessi jatkaa omaa suoritustaan, mikäli luku oli suoritettu. Mahdollisten ylimääräisten lohkojen käsittely hoidetaan keskeytysrutiineissa ja niiden ajojonoissa.

[8] least resently used
[9] programmed I/O
[10] direct memory access
[11] bottom half

Luku 4

Mittaukset

4.1 Mittausten tausta

Valtaosa liikennemittauksista on ollut nimenomaan liikenteen mittauksia: on mitattu ja analysoitu liikennettä ja tämän perusteella pyritty luomaan liikennettä kuvaava malli. Päätelaitetta on käsitelty mustana laatikkona, joka tuottaa tietynlaista liikennettä. Analyysin on kenties huomioitu TCP-protokollan dynamiikka mutta jätetty koko muu osa tietokonejärjestelmää huomioimatta. Tietokonejärjestelmissä on kuitenkin sekä säännöllisiä että epäsäännöllisiä tapahtumia, jotka vaikuttavat syntyvään liikenteeseen. Tietokoneessa olevien prosessien kokoelma tuottaa omanlaisensa liikenteen. Tässä esitettävillä mittauksilla on tarkoitus saada käsitys käyttöjärjestelmän toiminnasta sekä sen ja verkkoliikenteen keskinäisestä vuorovaikutuksesta.

Täysimittainen käyttöjärjestelmä on laaja kokonaisuus: esimerkiksi LINUX-käyttöjärjestelmän versiossa 2.0.29 on kaiken kaikkiaan 740 147 riviä C- ja assembler-kielistä lähdekoodia (kommentteineen), joka on jakautunut taulukon 4.1 mukaan. Sovellusohjelmat voivat olla myös hyvin laajoja. Eräiden sovellusten koodirivimääriä on listattu taulukossa 4.2. Ominaisuuksien lisäämisen seurauksena sovellusohjelmien koko on kasvanut.

Käyttöjärjestelmän toiminnan — etenkin tapahtumien ajallisten suhteiden — selvittäminen yksinomaan lähdekoodin perusteella on vaikeaa. Aina lähdekoodia ei ole edes

Taulukko 4.1: LINUX 2.0.29:n lähdekoodin jakautuminen toiminnallisesti.

Toiminta	rivejä	%-osuus
Arkkitehtuuririippuvaa	178 600	24,1
Laiteajurit	385 287	52,1
Tiedostojärjestelmät	66 823	9,0
Verkkokoodi	56 080	7,6
Varsinainen käyttöjärjestelmä tukitoimintoineen	53 357	7,2
yhteensä	740 147	100

Taulukko 4.2: Eräiden sovellusten lähdekoodirivimääriä.

Nimi	Versio	Tarkoitus	Rivejä
Apache	1.2b8	WWW-palvelin	36 209
WU-Ftpd	2.1c	FTP-palvelin	9 310
ftp (BSD)	5.38	FTP-asiakasohjelma	5 595
Lynx	2.7	tekstipohjainen WWW-asiakasohjelma	103 321
Mosaic	2.7b5	graafinen WWW-asiakasohjelma	132 075
Emacs	18.59	tekstieditori	151 035
Emacs	19.34b	tekstieditori	544 627
Joe	2.8	tekstieditori	24 621
Microsoft Word	2.0	tekstinkäsittelyohjelma [CS95]	326 000
Microsoft Word	95	tekstinkäsittelyohjelma [Mon97]	2 700 000

saatavilla. Vuorovaikutus laitteiston ja sovellusten kanssa mutkistaa tarkastelua edelleen, koska käyttöjärjestelmän koodin suorittaminen ei ole suoraviivaista vaan sitä suoritetaan eri useista pisteistä lähtien (sovellusohjelmien käyttöjärjestelmäkutsut, laitteiston keskeytykset) ja tehtäviä asetetaan erilaisiin jonoihin suoritettavaksi myöhemmin.

4.1.1 Mittausympäristön valinta

Mittausympäristön valinnassa muutama tekijä osoittautui muita merkittävämmäksi:

- Käyttöjärjestelmän lähdekoodin on oltava saatavissa ja edelleen levitettävissä. Tutkimuksessa joudutaan tekemään muutoksia käyttöjärjestelmän ytimeen, mikä edellyttää lähdekoodin saatavuutta. Tulosten ja rakennetun ympäristön halutaan olevan julkisia, mikä on mahdollista ainoastaan vapaasti levitettävillä käyttöjärjestelmillä.
- Edulliset laitteistovaatimukset.
- Järjestelmä ei saa olla rajoittunut tietyn valmistajan laitteisiin, koska valmistus ja tuki voivat loppua hyvin nopeasti yritysostojen seurauksina.
- Hyvä ATM-tuki jatkotutkimuksia ajatellen.

Näiden vaatimusten jälkeen valinta osui LINUX-käyttöjärjestelmään, joka on PC:ssä toimiva POSIX-yhteensopiva alunperin suomalaisen Linus Torvaldsin kehittämä vapaa käyttöjärjestelmä [BBD+96, Alm95].

Jonkin kokeellisen tai tutkimuskäyttöön suunnitellun käyttöjärjestelmän, esimerkiksi X-Kernelin [PD96], ottaminen tutkimuksen kohteeksi olisi kenties tarjonnut selkeämmän ja teoreettisesti puhtaamman kohteen. Jatkotutkimuksissa on kuitenkin tarkoitus analysoida todellisia sovelluksia todellisessa ympäristössä. Sovellusten saatavuus kokeellisiin järjestelmiin on heikkoa samoin kuin näiden tuki erilaisille laitteistoille.

Valinnan LINUX'n ja toisen vapaan käyttöjärjestelmäperheen, BSD-pohjaisten järjestelmien [MBKQ96], välillä ratkaisi LINUX'n hyvä ATM-tuki.

4.1.2 Mittausten vaiheet

Mittaukset suoritettiin LINUX-käyttöjärjestelmässä kahdessa eri vaiheessa. Ensimmäinen vaihe selvitteli tutkimuksen perusasetelmaa tuottaen alustavia tuloksia. Toisessa vaiheessa voitiin tarkentaa tutkimusta kiinnostaviin yksityiskohtiin ja välttää ensimmäisessä mittauksessa ilmaantuneita ongelmia. Tarkempien mittaustulosten ansiosta myös jatkopäätelmät ja -laskelmat voitiin tehdä tarkemmin.

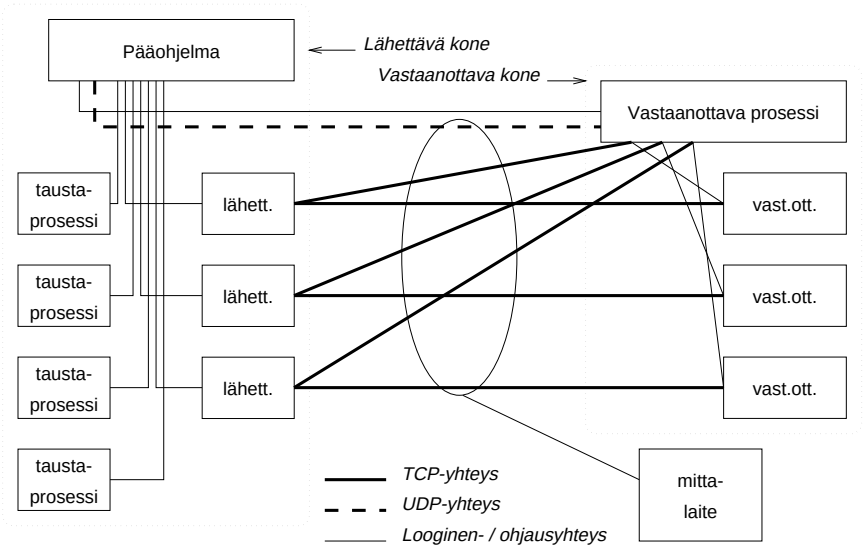
4.2 Ensimmäiset mittaukset

Mittauksien ensimmäinen vaihe suoritettiin Ethernet-verkossa, tarkemmin yhdessä 10Base2-segmentissä. Käytettävinä laitteina oli kaksi kappaletta 180 MHz:n Pentium Pro-prosessorilla varustettuja koneita, joista toinen toimi lähettävänä koneena ja toinen verkkoliikennettä mittaavana. Liikennettä vastaanottavana koneena toimi 133 MHz:n Pentium-prosessorilla varustettu kone. Sekä lähettävässä mittaavassa mittaavassa koneessa oli WD 8013 EBT -verkkokortit ja vastaanottavassa koneessa SMC Ultra -verkkokortti. Kaikkien koneiden käyttöjärjestelmänä oli RedHat 4.0 -jakelusta asennettu LINUX 2.0.18.

4.2.1 Ohjelmiston rakenne

Mittauksessa käytetty testiohjelma on yksi ohjelma, joka sisältää osat päärutiinille, lähetys-, tausta- ja vastaanottorutiineille sekä rutiinit mittaustulosten tallentamiseen. Ohjelma on kirjoitettu C-kielellä [KR88]. Mittausohjelman rakenne on esitetty kuvassa 4.1. Ohjelman suoritus muodostuu seuraavista vaiheista:

1. Ohjelma käynnistetään ja se tulkitsee syötearvot.
2. Lapsiprosessi käynnistää vastaanottoprosessin kohdekoneessa.
3. Taustaprosessit luodaan ja ne alustavat tarvitsemansa tietorakenteet.



Kuva 4.1: Mittausohjelman rakenne.

4. Pääprosessi odottaa vastaanottoprosessilta tietoa synkronointi- ja tiedonsiirtoporteista.

5. Pääprosessi ja vastaanottoprosessi mittaavat koneiden kellojen välisen eron vaihtamalla UDP-sähkeitä. Tätä tarvitaan mittausajankohtien synkronointiin.

6. Pääprosessi käynnistää tiedonsiirtoprosessit, jotka ottavat yhteyden vastaanottoprosessin tiedonsiirtoporttiin. Vastaanottoprosessi käynnistää jokaiselle yhteydelle eri prosessin.

7. Tehtyään alustustoimintonsa sekä tausta- että lähetysprosessit lähettävät signaalin pääprosessille ja jäävät odottamaan signaalia pääprosessilta.

8. Kun kaikki prosessit ovat valmiina, pääprosessi lähettää signaalin kaikille lapsiprosesseille, jotka aloittavat toimintansa.

9. Alkuviiveen kuluttua sekä tausta-, lähetys- että vastaanottoprosessit alkavat tallentaa aikaleimoja eri tapahtumista ennalta asetellun määrän.

10. Tämän jälkeen ohjelmat jatkavat toimintaansa vielä jälkiviiveen ajan ja lopuksi tallentavat aikaleimat tiedostoon.

Mittauksessa käytetään lisäksi perl-kielistä [WCS96] apuohjelmaa, joka käynnistää testiohjelman halutuilla arvoyhdistelmillä ja ohjaa myös verkkomittauksia ja lopuksi

kerää yhden mittauksen tulokset yhteen hakemistoon. Yhden mittauksen jälkeen ohjelma pitää tauon, jona aikana koneiden puskurit tyhjennetään mahdollisimman hyvin kopioimalla suuri tiedosto `/dev/null`-nollalaitteelle.

4.2.2 Mittaustapahtuma

Mittaus suoritettiin ajamalla kaikki yhdistelmät 1, 2, 5, 10, 15 ja 20 lähettävällä prosessilla ja 0, 1, 2, 5, 10, 15 ja 20 taustaprosessilla eli kaiken kaikkiaan 42 eri mittausta. Näistä tarkempaan käsittelyyn otettiin alustavien arvioiden mukaan kiinnostavimmat yhdistelmät eli (1,0), (1,1), (1,2), (1,5), (1,10), (1,20), (10,0), (10,1) ja (10,10).

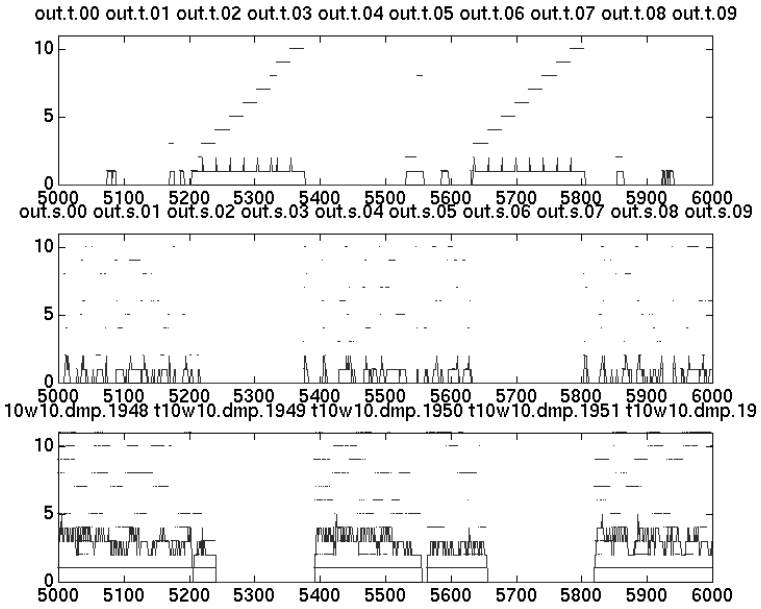
Mittauksen aikana Ethernet-segmentti ei ollut täysin eristetty, mutta muu segmentissä oleva liikenne oli vähäistä. Pääasiallisiin tuloksiin liikenteellä ei havaintojen mukaan ollut vaikutusta. Verkkomittauksiin käytettiin tcpdump-ohjelmaa [JLM94].

4.2.3 Tulosten käsittely

Tässä ensimmäisessä vaiheessa turvauduttiin pääasiassa kuvaajista visuaalisesti tehtyihin havaintoihin ja muutamien yksinkertaisten tunnuslukujen laskemiseen. Sopivia, kuvaavia analyysimenetelmiä etsittiin saadulle mittausaineistolle.

4.2.3.1 Prosessien vuorottelu

Ensimmäisenä määrättiin prosessit, jotka olivat olleet ajossa kunakin 10 ms ajanhetkenä. Tästä laadittiin kolmiosainen kuva, jossa yhteen oli merkitty taustaprosessit, toiseen lähettävät prosessit ja kolmanteen vastaanottavat prosessit. Kuvasta 4.2 näkyy selvästi ensimmäisessä mittauksessa oleva ongelma: taustaprosessit suoritetaan yhtenä joukkona peräkkäin, koska prosessit on käynnistetty ryhminä. Todellisissa sovelluksissa I/O- ja CPU-intensiiviset prosessit käynnistyvät sekajärjestyksessä.



Kuva 4.2: Eri prosessien ajohetket 10 lähettävällä ja 10 taustaprosessilla.

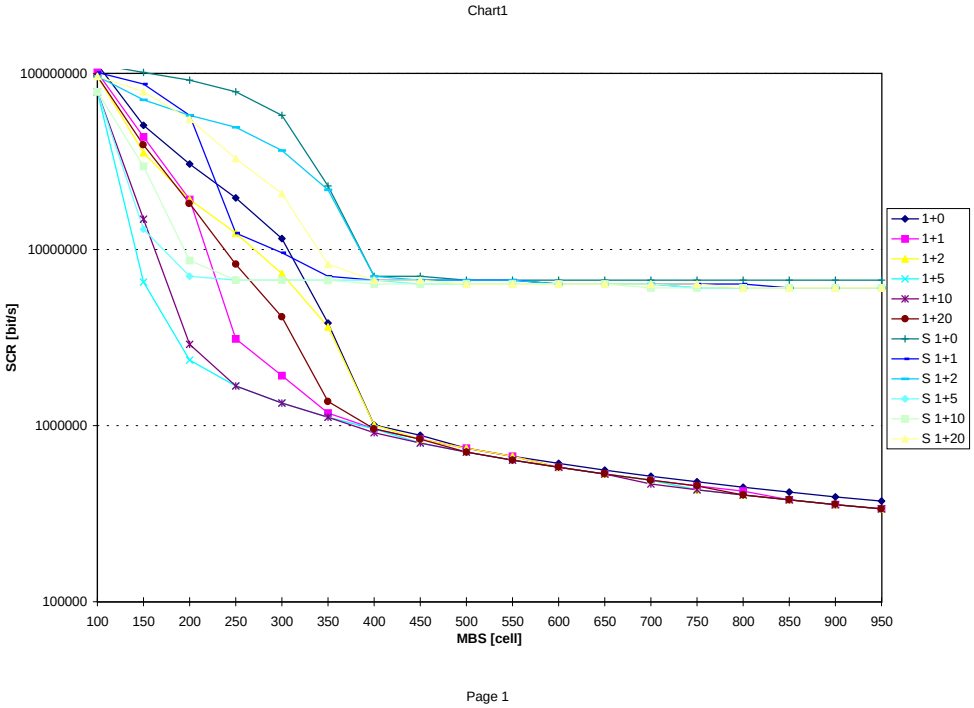
4.2.3.2 Sopiva liikennesopimus

Seuraavaksi tutkittiin lähetetyn liikenteen ominaisuuksia verkon kannalta soveltamalla ATM-verkoissa käytettyjä liikenneparametreja (kappale 2.5.1, sivu 29).

Mitattu Ethernet-liikenne muunnettiin ATM-liikenteeksi muuntamalla jokainen Ethernet-kehys vastaavaksi määräksi ATM-soluja. Solujen välinen aika oli STM-1:n mukainen aika eli yhden kehyksen solut menisivät STM-1-nopeudella (noin 155 Mbit/s) peräkkäin. Tämä vastaa tilannetta, että verkkokorttitasolla ei ole mitään kaistanleveyden rajoitusta käytössä. Nykyiset ATM-verkkokortit jakavat AAL5-kehyksen soluiksi laitteistotasolla ja siten kykenevät tähän.

Näin suurta linjanopeutta käytettiin laskuissa todellisen 10 megabitin sekuntinopeuden asemasta, koska mittauksessa oli ilmennyt epätarkkuuksia. Kehysten aikaleimat olivat mittausvirheestä johtuen tiheämmässä kuin siirtotiellä mahdollista (kappale 4.3.3, s. 53). Muunnosohjelma tarkisti, etteivät kehykset menneet lomittain.

Saatu ATM-liikenne syötettiin tämän jälkeen liikennesopimuksen valvontasimulaattoriin erilaisilla liikenne sopimuksilla. Kunkin mittaustapahtuman liikenteelle etsittiin liikennesopimus, josta liikenne menisi läpi hukkaamatta yhtään solua. Kuvassa 4.3 on laskettu vaadittava keskinopeus (SCR), jotta suurimmalla purskekoolla (MBS)



Kuva 4.3: Vaadittu keskinopeus eri purskeko'oilla. Ylempi käyräryhmä: yhteyksien summaliikenne. Alempi käyräryhmä: yksittäisten yhteyksien arvojen keskiarvo.

($BT = (MBS - 1)\left(\frac{1}{SCR} - \frac{1}{PCR}\right)$) liikenne olisi kokonaisuudessaan liikennesopimuksen mukainen.

Kuvasta 4.3 nähdään, että suurin purskekoko on noin 400 solun kokoinen eli tämä vastaa reilua 12:ta 1 500 tavun mittaista kehystä. Vaadittava keskinopeus on pienempi pienillä alle 400 solun purskeko'oilla, mikäli taustaprosesseja on ollut useita, joskaan tämä ei mene aivan suorassa suhteessa. Tähän voi olla syynä mittaustapahtuman aikana ollut ulkoinen häiriö tai mittauksen aikaleimojen epätarkkuus.

4.3 Toinen mittaus

Toiseen mittauksen perusjärjestely oli samankaltainen kuin ensimmäisessä mittauksessa. Mittausten välissä LINUX-käyttöjärjestelmä oli päivitetty versiosta 2.0.25 versioon 2.0.29, jossa ei kuitenkaan ole merkittäviä eroavaisuuksia tämän työn kannalta. Verkosta mittaava kone vaihdettiin kappaleessa 4.3.3 (s. 53) esitettävien syiden vuoksi.

4.3.1 Mittausohjelmisto

Mittausohjelman perusrakenne oli sama kuin ensimmäisessä mittauksessa (kappale 4.2.1, s. 46), joskin sitä oli havaittujen ongelmien ja myöhemmin tulleiden ideoiden pohjalta muutettu seuraavasti:

- Lähettävä prosessi huolehtii yhteyden sulkemisesta. Ensimmäisessä versiossa kumpi prosessi tahansa saattoi sulkea yhteyden taulukon tai mittausajan täytyttyä: tämä monimutkaisti lähettävän prosessin rakennetta, vrt. kappale 2.3.4 (s. 26). Vastaanottava prosessi lukee niin kauan, kuin dataa on luettavana mutta lopettaa tilastotiedon tallentamisen mikäli taulukko täyttyy.

- Tausta- ja lähetysprosessit käynnistetään vuorotellen eli ensin käynnistetään ensimmäinen taustaprosessi, sitten ensimmäinen lähetysprosessi, toinen taustaprosessi ja niin edelleen.

  Tausta- ja lähetysprosessien vuoroittainen käynnistäminen ei kuitenkaan poistanut täysin ensimmäisissä mittauksissa esiintynyttä CPU-intensiivisten ja I/O-intensiivisten prosessien ryppäitä. Tämä ominaisuus on ilmeisestikin vuorottelupolitiikan seurausta eikä mittausjärjestelyistä johtuva ilmiö.

- Taustaprosessi on muistin käytöltään enemmän normaalin ohjelman kaltainen. Ensimmäisessä versiossa ohjelma suoritti vain yksinkertaista silmukkaa, uudessa versiossa taustaprosessi laskee Eratostheneen seulalla alkulukuja halutun kokoisesta ($2^N, 1 \leq N \leq 25$ eli 2 tavua – 32 megatavua) datajoukosta. Mittauksessa käytettiin 4 096 tavun suuruista muistitilaa.

  Jokaisen kierroksen jälkeen prosessi tallensi kellonajan ja vertasi löydettyjen alkulukujen määrää sekä suurinta löydettyä alkulukua etukäteen taulukkoon laskettuun arvoon mahdollisen kääntäjän optimointivirheen havaitsemiseksi. Muutettavaa data-alueen kokoa voidaan esimerkiksi käyttää suurilla arvoilla sivutusjärjestelmän ja pienemmillä arvoilla välimuistien suorituskyvyn mittaukseen.

- Ensimmäisessä versiossa kaikki lähetysprosessit lukivat tiedoston alusta, jolloin kukin prosessi hyötyi toisten jo lukemasta tiedosta. Toisessa versiossa kukin prosessi i aloitti lukemisen kohdasta $(i \bmod N) \times tiedostokoko/N$, jolloin proses-

Taulukko 4.3: Prosesseista tallennetut tiedot.

Muuttuja	Edellinen	Seuraava
Prosessitunniste (PID)	X	X
CPU:n käyttölaskuri	X	X
Prosessin tilatieto	X	
Päällä olevat signaalit		X
Käyttäjätilan ohjelmalaskuri	X	X

sien toisistaan saama hyöty oli mahdollisimman pieni. Mittauksissa käytetyn tiedoston koko oli 130 megatavua, mikä oli riittävä näihin mittauksiin.

Kaikkien prosessien kohteliaisuusarvo ($nice$) oli oletus (0).

4.3.2 Käyttöjärjestelmän modifiointi

Tiedon keräämiseksi käyttöjärjestelmän sisältä vuorottelualgoritmin toiminnasta ja eri suureiden arvoista, esimerkiksi puskurien käyttöasteesta, oli välttämätöntä muokata käyttöjärjestelmää.

4.3.2.1 Tietojen kerääminen käyttöjärjestelmästä

Vuorottelun suorittavaan funktioon `schedule()` lisättiin kutsu tietoa tallentavaan rutiiniin juuri ennen kuin prosessi mahdollisesti vaihtui. Näin tietoa päätöksestä tallentui vaikkei ajettava prosessi olisikaan vaihtunut. Prosesseista tallennettiin taulukon 4.3 mukaiset tiedot, jonka lisäksi tallennettiin aikaleima, ajoon valitun prosessin hyvyysarvo sekä eri tehtäväjonojen pituudet. Verkkopuskureiden tiedoista tallennettiin senhetkinen käyttömäärä, verkkoajurien varaama puskurimäärä, puskureiden varauksien kokonaismäärä, epäonnistuneet puskurivaraukset, virheelliset puskurien vapautukset ja IP-fragmenteille varattu puskurikoko.

LINUX-käyttöjärjestelmä sallii keskusmuistin varaamisen maksimissaan 128 kilotavun[1] suuruisina lohkoina. Yhteen lohkoon mahtuu vain 1 560 tietuetta, joten riittä-

[1]Tämä riipuu järjestelmässä käytetystä sivukoosta: tässä 4 kilotavun sivuilla. Lisäksi lohkosta varataan alue kirjanpitoon. Tämä on 24 tavua, mikäli osoittimen pituus on 32 bittiä kuten Pentium-arkkiteh-

vän pitkän tallennusajan saamiseksi näitä varattiin 256 kappaletta eli muistia kului 32 megatavua. Kaikkiaan tilaa oli 399 360:lle tapahtumalle. Tällä järjestelyllä laite toimi kuten se olisi ollut varustettu vain 32 megatavun fyysisellä muistilla todellisen 64 megatavun asemasta.

Mittauksien aikana talletettiin enimmillään 2 311 prosessin vaihtoa, joista varsinaisen mittauksen aikana tapahtui noin 1 550. Yksikin lohko olisi siten juuri ja juuri riittänyt. Etukäteen ei voitu kuitenkaan tarkasti tietää todellista tallennustarvetta ja ketjutettujen lohkojen toteutuksessa lohkojen määrän lisääminen ei monimutkaista toteutusta. Iso muistitila oli sikäli perusteltu, että tallennuksen aiheuttaman tehohäviön mittaamisessa (kappale 4.3.5, s. 55) yksi lohko olisi täyttynyt vajaassa neljäsosasekunnissa. Ylimääräistä tilaa voidaan käyttää myöhemmissä mittauksissa esimerkiksi systeemikutsujen tallentamiseen.

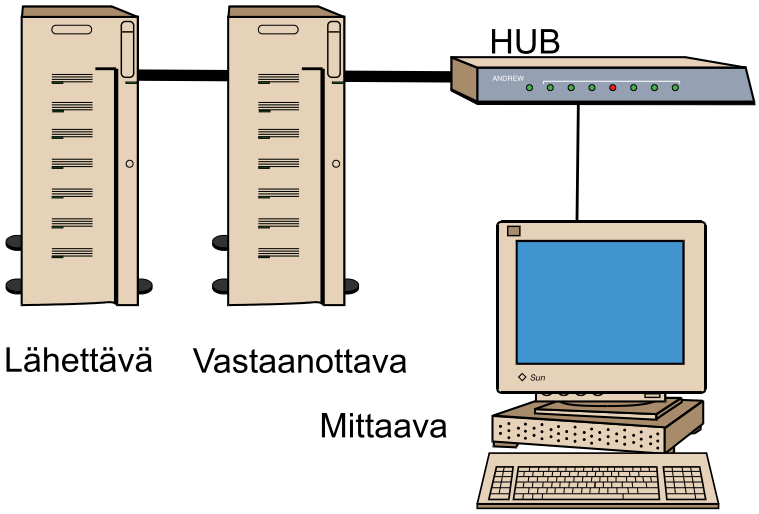
Tietojen tulostus tapahtui `/proc`-tiedostojärjestelmän kautta, joka tarjosi helpon tavan tuottaa tietoa käyttöjärjestelmästä käyttäjälle [BBD+96]. Tiedot saatiin tallennetuksi kopioimalla tiedosto `/proc/CNTX_SWTCH` levylle.

4.3.3 Verkkoliikenteen tallentaminen

Liikenteen mittaamiseen verkosta oli alunperin tarkoitus käyttää hieman modifioitua LINUX-käyttöjärjestelmää ja tcpdump-ohjelmaa. Ensimmäisen mittauksen ongelmat johtuivat kehyksen tuloajan määräämisestä vasta sovelluksessa, jolloin aika ei ole tarkka. Normaalisti LINUX asettaa vastaanottaessaan kehyksen sen tuloajan `xtime`-tietorakenteesta, jota päivitetään 10 ms välein. Tämä tarkkuus on riittämätön, koska kahden peräkkäisen täysimittaisen Ethernet-kehyksen väliaika voi olla vain 1,3 ms. Muokattu versio kutsui `do_gettimeofday()`-rutiinia, jonka pitäisi antaa tarkka aika mikrosekuntien tarkkuudella. Rutiini ei kuitenkaan toiminut odotetusti kutsuttaessa keskeytyksestä, jolloin jälkimmäinen kahdesta ajasta saattoi olla jopa 10 ms aikaisempi.

Mittauksiin käytettiin SUNOS 5.5 -käyttöjärjestelmällä varustettua Sun UltraSparc 1 -työasemaa, jossa kehysten aikaleimat olivat järkeviä kokeilujen perusteella: peräk-

tuurissa.



Kuva 4.4: Mittausjärjestelyt.

käisten kehysten väliset ajat eivät olleet liian lyhyitä. Absoluuttista tarkkuutta ei kuitenkaan pystytty mittaamaan. Mittaustuloksia analysoitaessa ei havaittu epätarkkuudesta johtuvia ilmiöitä. Käytetyn tcpdump-ohjelman versio oli 3.3 [JLM94].

Verkosta mitatut tapahtumat synkronoitiin mitattavan koneen aikaan synkronointisähkeiden avulla jälkikäsittelyssä. Jokaisesta synkronointisähkeestä tiedettiin sen lähetysaika. Tätä verrattiin verkosta mitattuun aikaan ja aikaeroista valittiin pienin. Perusteluna pienimmän aikaeron käyttöön keskiarvon tai mediaanin sijasta on lähetysprosessi: ennenkuin sähke pääsee sovellukselta verkkoon, voi olla satunnaisia eri lähteistä peräisin olevia viiveitä. Valitsemalla pienin ero saadaan varmistetuksi, että ”verkkoaika” ei ole aikaisempi kuin ”koneaika”.

4.3.4 Mittausjärjestely

Mittausta ohjasi verkkomittauksia suorittava kone. Kolme mittauksissa käytettyä konetta olivat omassa eristetyssä segmentissään kuvan 4.4 mukaan. Vastaanottava ja lähettävä kone olivat kytketyt samaan 10Base2-segmenttiin noin metrin etäisyydelle toisistaan. Verkkomittauksia suorittava kone oli kytketty noin 10 metrin päähän 10BaseT-keskittimeen, johon 10Base2-segmentti oli liitetty trancieverillä.

Jokaisen mittauksen välillä sekä lähettävä että vastaanottava kone käynnistettiin uudelleen. Näin mittauksista saatiin suoritusjärjestyksestä riippumattomia ja vältettiin muis-

Taulukko 4.4: Yhden kirjoitus-luku -silmukan aika.

	Aika (μs)
Alkuperäinen 2.0.29 ydin	28,5
Muokattu 2.0.29 ydin	31,9
Ero prosessin vaihtoa kohden	1,7

tin tyhjennys kopioimalla, mikä oli hitaampaa kuin uudelleenkäynnistys.

4.3.5 Mittausvirheiden arviointi

Käyttöjärjestelmän muokkaamisen aiheuttamaa häiriötä arvioitiin mittaamalla kahden prosessin välistä kirjoitusoperaatiota. Prosessit oli yhdistetty toisiinsa kahdella putkella ja prosessit vuorotellen lukivat ja kirjoittivat yhden merkin, jolloin prosessinvaihdon täytyi tapahtua jokaisella kirjoituksella.

Mittaamalla 100 kertaa 100 000:n kirjoitus-luku silmukan aika, saatiin taulukossa 4.4 olevat tulokset. Mittausten aikana tapahtui eri mittauksissa keskimäärin 16,3 – 41.3 kutsua sekunnissa `schedule()`-funktioon. Tehohäviö on alle $1/10\ 000$, mikäli oletetaan, että keskimäärin olisi 50 kutsua sekunnissa. Tarvitaan yli 6 000 kutsua sekunnissa, jotta tehohäviö olisi yksi prosentti.

Luku 5

Tulokset

Toisiin mittauksiin mennessä mittausjärjestelmä oli saatu toimimaan halutulla tavalla. Suurimmat mittauksen häiriötekijät oli saatu poistetuksi ja tarkkuutta parannetuksi. Häiriötekijöiden poistamisen jälkeen tulosten voitiin olettaa sisältävän tietoa järjestelmän toiminnasta eikä häiriöistä. Näin voitiin suorittaa varsinaiset mittaukset ja analysoida tulokset vuorottelun toiminnan ja syntyvän liikenteen karakteristen ominaisuuksien selvittämiseksi eri tapauksissa.

5.1 Mitä mitattiin

Mittauksia suoritettiin kaikkiaan 36 kappaletta eli kaikki yhdistelmät 1, 2, 5, 10, 15 sekä 20 lähettävällä prosessilla ja 0, 1, 2, 5, 10 sekä 20 taustaprosessilla. Mittaustapahtumassa tallennettiin seuraavat tiedot:

verkkoliikenne Synkronointisähkeistä ja varsinaisesta tiedonsiirrosta tallennettiin 64 tavua jokaisen kehyksen alusta aikaleiman lisäksi. Tämä riittää mainiosti IP+TCP-otsikkotietojen analysointiin. Tallennetuista tiedoista synkronoitiin eri koneiden kellonajat sekä määritettiin kehysten aikaleimat.

käyttöjärjestelmä Lähettävän koneen käyttöjärjestelmästä tallennettiin kokonaisuudessaan kappaleessa 4.3.2.1 (s. 52) mainitut tiedot. Tallennus alkoi hiukan ennen

mittausta ja jatkui mittauksen jälkeen siihen asti, että tiedosto saatiin kopioiduksi levylle.

Käyttöjärjestelmän tiedoista hyödynnettiin tietoa kulloinkin ajossa olevasta prosessista ja käytössä olevista verkkoliikenteen puskureista.

lähettävä prosessi Lähettävä prosessi tallensi alkuviiveen jälkeen kolme tietuetta jokaista luku-kirjoitus -tapahtumaa kohti. Luettujen ja kirjoitettujen tavujen lukumäärän lisäksi aikaleimat tallennettiin ennen levyltä lukua, levyltä luvun jälkeen sekä ennen verkkoon kirjoitusta. Verkkoon kirjoituksen jälkeinen aika on sama kuin levyltä lukua edeltävä aika.

Näistä saadaan kirjoitusten ajat (kirjoitusta edeltävä aikaleima) sekä levyltä lukuun kulunut aika (kahden ensimmäisen ajan erotus).

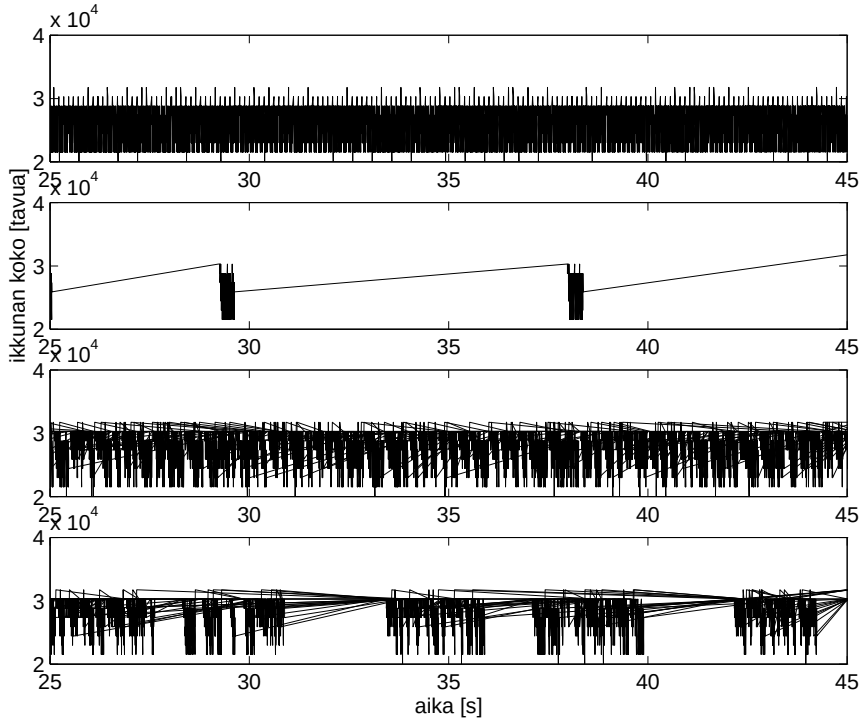
vastaanottava prosessi Vastaanottava prosessi tallensi jokaisen vastaanotetun segmentin aikaleiman ja vastaanotettujen tavujen lukumäärän, mutta tätä tietoa ei hyödynnetty.

taustaprosessi Taustaprosessi tallensi jokaisen kierroksen jälkeisen kellonajan, mutta tätä tietoa ei myöskään hyödynnetty. Aikoja voitaisiin käyttää suorituskyvyn analysoimiseen: toimiiko prosessi toisella vuorottelualgoritmilla hitaammin esimerkiksi välimuistin tyhjenemisen vuoksi.

Mittausdataa kertyi kaikkiaan 80 megatavua. Tämän perusteella voitiin lähetä tarkastelemaan tuloksia.

5.2 Perusarviointi

Ensin tarkistettiin mahdollisesti mittauksessa esiintyvät pullonkaulat ja mittausvirheet. Verkosta tallennettujen kehysten väliajat olivat oikeaa suuruusluokkaa (teoreettinen minimi 1,3 ms, pienin mitattu täysimittaisen kehyksen väliaika oli yli 1,7 ms), joten ensimmäisissä mittauksissa esiintynyttä virhettä ei tässä ollut ainakaan yhtä suurena. Toinen mahdollinen pullonkaula, vastaanottava kone, ei myöskään rajoittanut mittausta. Kuvasta 5.1 nähdään neljän mittauksen osalta, että käytettävissä oleva ikkunakoko

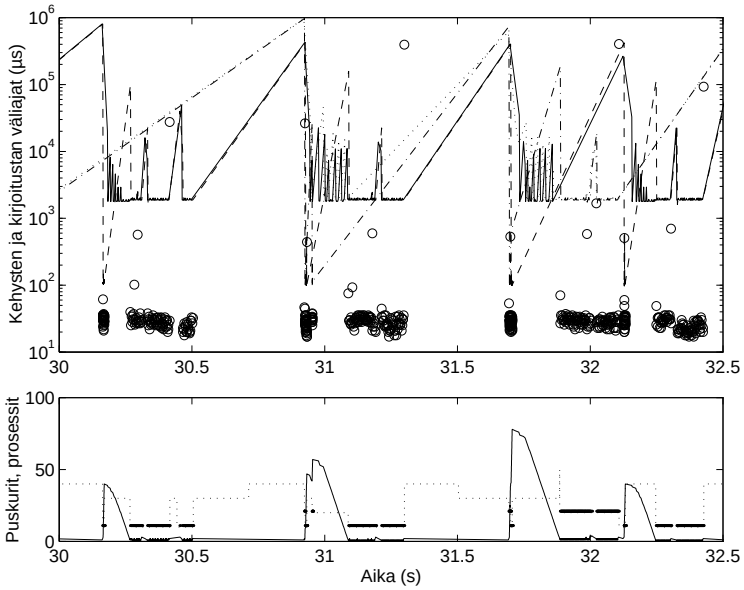


Kuva 5.1: Käytössä oleva ikkunakoko. Ylhäältä: yksi lähettävä prosessi, ei taustaprosesseja; yksi lähettävä prosessi, 20 taustaprosessia; 20 lähettävää prosessia, ei taustaprosesseja; 20 lähettävää ja taustaprosessia.

oli vähintään 20 000 tavua. Tämä pätee kaikissa mittauksissa, joten lähettävä kone pystyi TCP:n kannalta lähettämään aina.

Yleiskuvaaja on esitetty kuvassa 5.2. Lukuunottamatta joukkoa noin 100 mikrosekunnin luokkaa olevia kirjoituksien välisiä aikoja, nämä ajat korreloivat verkosta mitatut kehysten välisten aikojen kanssa. Levyltä lukemiseen kulunut aika on hyvin säännöllinen, suurimmaksi osaksi 20 – 40 μs.

Alempaan osakuvaan on merkitty käytössä olevat verkkopuskurit yhtenäisellä viivalla. Kunakin ajanhetkenä ajossa oleva prosessi ilmenee porrasmaisesta pisteviivasta: lähettävät prosessit ovat 10:n ja 20:n kohdalla ja taustaprosessit 30:n ja 40:n kohdalla. Ajanhetkenä 31,9 s ajossa on ollut käyttöjärjestelmän apuprosessi, joka näkyy piikkinä ylöspäin, arvoon 50. Kunkin prosessin jokainen kirjoitustapahtuma on merkitty pisteenä kyseisen prosessin tasolle pisteviivan päälle.



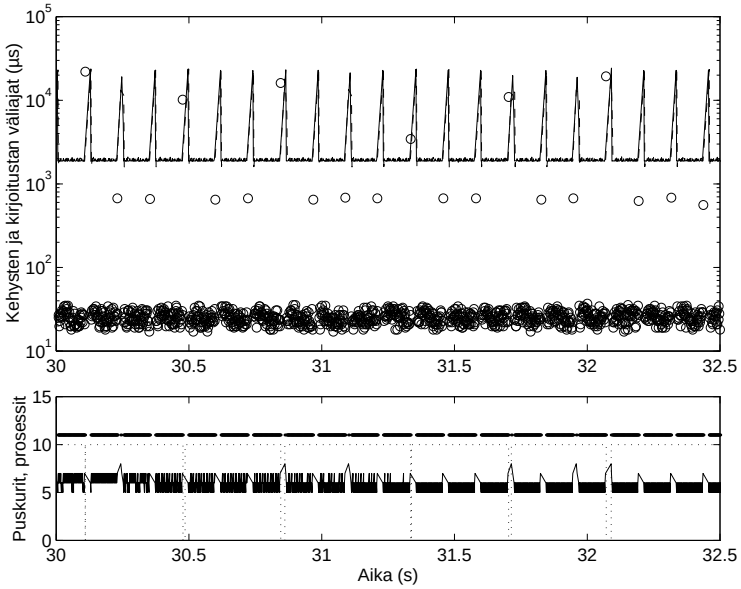
Kuva 5.2: Tapahtumakaavio: kaksi lähettävää prosessia, kaksi taustaprosessia. Kehysten väliajat: yhtenäinen ja pisteviiva. Kirjoitusten väliajat: katko- ja katkopisteviiva. Levylukuun kulunut aika: ympyrät.

5.3 Peräkkäisten kirjoitusten sekä kehysten väliset ajat

Lähettävän prosessin tallentamien peräkkäisten kirjoitusten välisistä ajoista laadittiin 50-lokeroisen histogrammin avulla jakaumat. Histogrammi oli tasavälinen ajan logaritmin suhteen ($ln(t)$). Syynä logaritmin käyttöön oli arvoalueen laaja jakauma: kirjoitusten väliajat vaihtelivat välillä 90 μs – 8,3 s. Samanlaiset histogrammit laadittiin myös verkosta mitattujen saman yhteyden peräkkäisten kehysten välisistä ajoista.

5.3.1 Yksi lähettävä prosessi, ei taustaprosesseja

Tilanne, jossa on vain yksi lähettävä prosessi eikä yhtään taustaprosessia tuottaa hyvin tasaisen liikenteen, kuten kuvasta 5.3 voidaan havaita. Lähes kaikkien (yli 97 %) peräkkäisten kehysten välinen aika on 1,9 ms, pienin väliaika on 1,811 ms ja mediaani on 1,882 ms. Noin sadan kehyksen välinen aika on kuitenkin pitempi, noin 23 ms ja pisin kehysten välinen aika on on 33 ms. Kirjoitusten osalta hajonta on hiukan suurempi: minimi on 1,566 ms, mediaani 1,878 ms ja maksimi 34 ms. Väliaikojen jakaumat on esitetty kuvissa 5.4(a) ja 5.4(b). Levyltä lukemiseen kuluva aika pysyttelee pääosin 20



Kuva 5.3: Tapahtumakaavio: yksi lähettävä prosessi, ei taustaprosesseja. Kehysten väliajat: yhtenäinen viiva. Kirjoitusten väliajat: katkoviiva. Levylukuun kulunut aika: ympyrät.

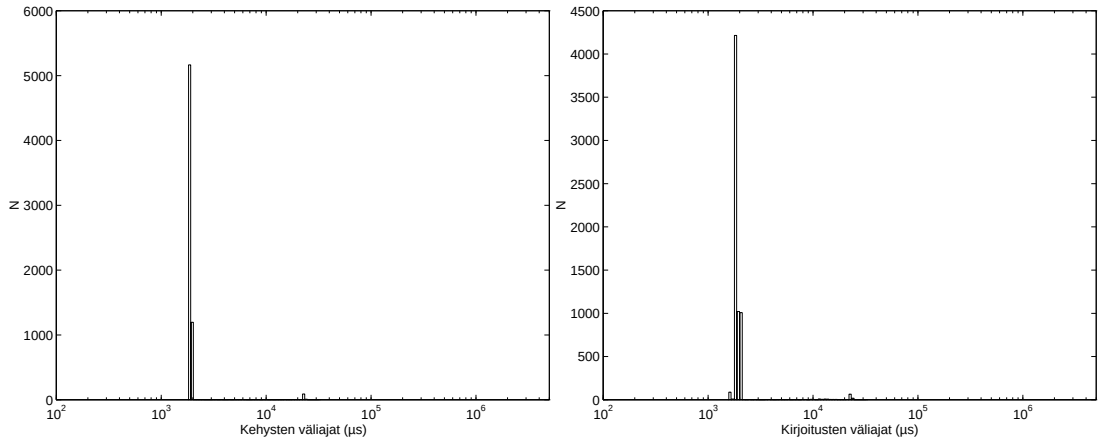
ja 40 mikrosekunnin välillä.

Sovelluksen kirjoituskäskystä segmentti lähtee välittömästi verkkoon. Välillä (10 kertaa sekunnissa) käyttöjärjestelmä tekee jotain kauemmin vieviä (noin 20 ms) toimia, jolloin prosessin suoritus keskeytyy täksi ajaksi.

5.3.2 Useita lähettäviä prosesseja, ei taustaprosesseja

Lähettävien prosessien lukumäärän lisääntyessä ilmenee kilpailu- ja vuorottelutilanteen aiheuttamia ilmiöitä. Jo kahdella lähettävällä prosessilla reilut 10 % kirjoitusten väliajoista on reilusti alle kehysten väliajan eli noin 100 mikrosekunnin luokkaa. Tämä nopeatahtinen kirjoitusrypäs ilmenee prosessin oltua jonkin aikaa lähettämättä: kirjoituksia on tällä nopealla tahdilla noin 40 kappaletta, jonka jälkeen ajettavaa prosessia vaihdettaan.

Tiheiden kirjoitusten osuus kasvaa lähettävien prosessien lukumäärän kasvaessa. Kymmenellä lähettävällä prosessilla yhteyskohtaisen väliaikojen mediaanien mediaani on 116 μs. Viidellä prosessilla kirjoitusten mediaani on vielä samaa luokkaa kuin kehys-



(a) Kehykset. (b) Kirjoitukset.

Kuva 5.4: Aikojen jakaumat: yksi lähettävä prosessi, ei taustaprosesseja.

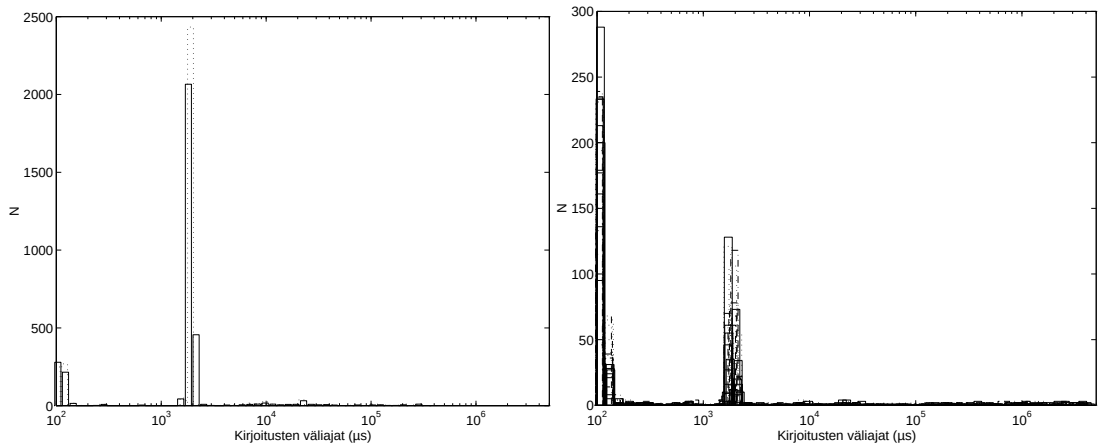
ten välinen mediaani eli 1,9 ms. Kirjoitusten välisten aikojen jakauma on esitetty kuvassa 5.5(a) kahdelle ja kuvassa 5.5(b) kahdellekymmenelle lähettävälle prosessille.

Kehysten välinen maksimiaika kasvaa prosessien lukumäärän lisääntyessä siten, että kahdella lähettävällä prosessilla se on 430 ms, viidellä 1,3 s, kymmenellä 2,7 s ja kahdellakymmenellä 4,6 s.

5.3.3 Yksi lähettävä ja yksi taustaprosessi

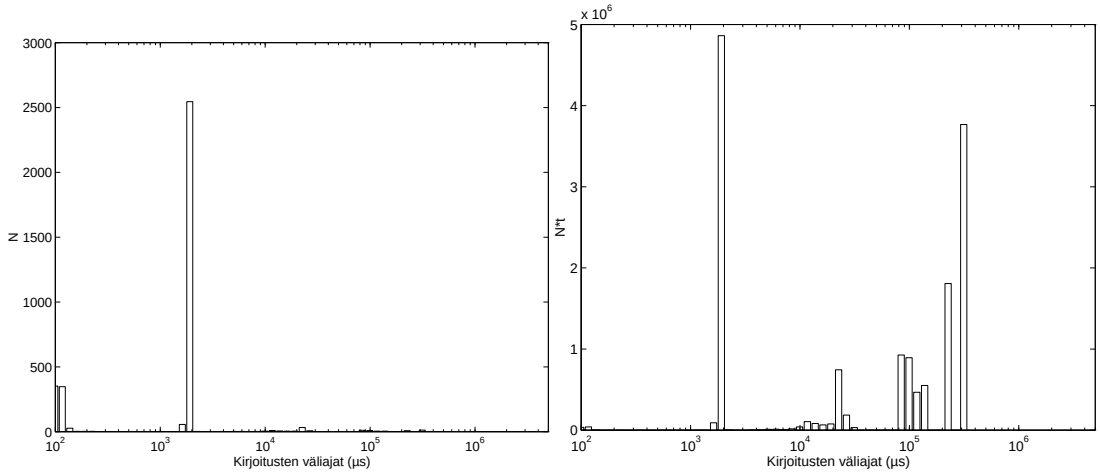
Yhdellä lähettävällä ja yhdellä taustaprosessilla arvot ovat varsin samat kuin kahdella lähettävällä prosessilla ilman taustaprosesseja. Prosessin kannalta ei ilmeisesti ole eroa sillä, onko toinen prosessi I/O- (kuva 5.5(a)) vai CPU-intensiivinen (kuva 5.6(a)).

Havainnollinen kuva prosessin ajankäytöstä saadaan, kun normaalin histogrammin kussakin luokassa oleva alkioiden lukumäärä kerrotaan k.o. luokan arvolla. Näin saadaan pylväsdiagrammi, joka suoraan antaa kussakin luokassa kuluvan kokonaisajan. Kuvista 5.6(a) ja 5.6(b) nähdään, että vaikka luokat yli 100 ms ovat normaalissa histogrammissa hyvin pienet, skaalatusta histogrammista nämä muodostavat yli puolet kokonaisajasta. Sen sijaan tiheiden kirjoitusten luokassa (noin 100 μs) prosessi on hyvin pienen osan kokonaisajasta.



(a) 2 prosessia. (b) 20 prosessia.

Kuva 5.5: Kirjoitusten välisten aikojen jakaumat: ei taustaprosesseja.



(a) Normaali jakauma. (b) Skaalattu jakauma.

Kuva 5.6: Kirjoitusten väliaikojen jakauma yhdellä lähettävällä ja yhdellä taustaprosessilla.

5.3.4 Useita lähettäviä ja taustaprosesseja

Lähettävien ja taustaprosessien lukumäärän lisääminen lisää sekä lyhyitä että pitkiä kirjoituksien väliaikoja. Väliaikojen keskiarvo kasvaa prosessien määrän lisääntyessä. Keskimääräinen väliaika yhden lähettävän prosessin tapauksessa on 2,3 ms, kun se viidellä lähettävällä ja taustaprosessilla on 20 ms.

5.4 Lähetyspuskurien käyttö

Kuvasta 5.2 nähdään tyypillinen puskurien käyttö prosessille. Kun prosessi alkaa lähettää tauon jälkeen, se kirjoittaa hyvin suuren määrän käyttäen jopa 40 puskuria (60 kilotavua) jonka jälkeen se lopettaa suorituksensa. Jokin toinen prosessi aloittaa suorituksensa ja samalla ensimmäisen prosessin lähetysjonoa aletaan tyhjentää.

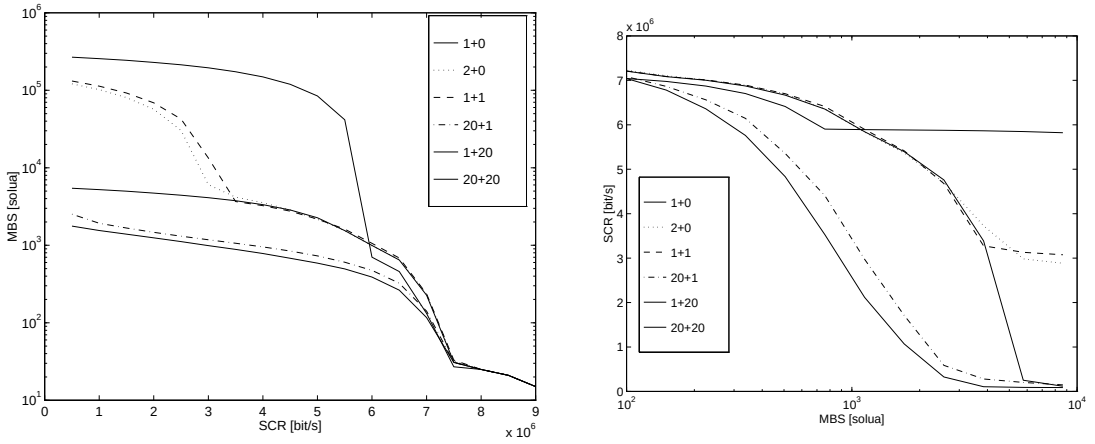
TCP:n ruuhkanhallintaan kuuluvasta hitaasta käynnistyksestä johtuen jono tyhjenee ensin hitaasti mutta sitten saavuttaa normaalin nopeutensa. Kun prosessi sitten jatkaa suoritustaan, puskureiden käyttö on hyvin tasaista ts. kirjoitus tapahtuu suoraan sovelluksesta verkkoon.

5.5 Liikenteen karakteriset ominaisuudet

5.5.1 Sopiva liikennesopimus

Vastaavasti kuin ensimmäisessä mittauksessa (kappale 4.2.3.2, s. 49), toisenkin mittauksen verkkoliikenne muunnettiin vastaavaksi ATM-liikenteeksi. Tarkemman mittauksen ansiosta kehyksen solujen lähetysnopeus voitiin asettaa todellisuutta vastaavasti 10 Mbit/s:ksi. Useat ATM-kortit voivat lähettää solut täällä nopeudella huippunopeuden asemesta mikäli muokkaimessa on näin määrätty.

Liikenneparametreista huippunopeus muutettiin vastaamaan solunopeutta eli PCR:n arvo oli 10 Mbit/s. Soluviiveen vaihtelun toleranssi ($CDVT$) oli kehyksen ajan suurui-



(a) SCR:n mukainen MBS.

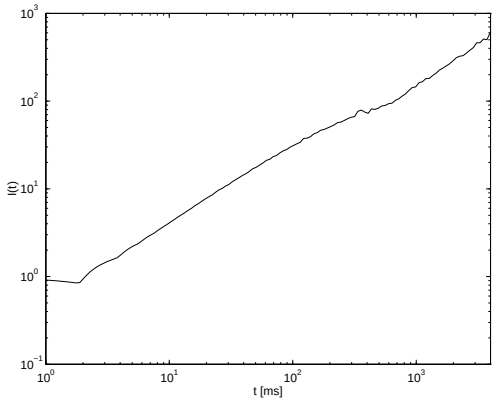
(b) MBS:n mukainen SCR.

Kuva 5.7: Tarvittavat liikennesopimuksen parametrit.

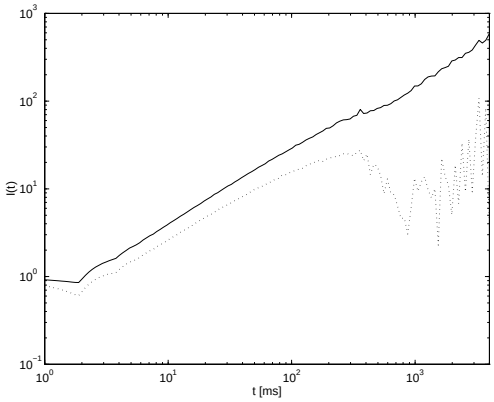
nen eli noin 1,3 ms. Kuvissa 5.7(a) ja 5.7(b) on esitetty pienin mahdollinen purskekoko tietyllä keskinopeuden arvolla ja pienin mahdollinen keskinopeus tietyllä purskekoolla, jotta tarjottu liikenne olisi liikennesopimuksen mukaista. Molemmat käyrät ilmaisevat siis samaa asiaa ja toinen saadaan toisesta kääntämällä ja peilaamalla; havainnollisuuden vuoksi molemmat on kuitenkin liitetty mukaan. Käyrät ovat keskiarvoja eri yhteyksien arvoista; jokaisen yhteyden liikenneparametrit on laskettu erikseen.

Kuvista voidaan havaita sama ilmiö kuin histogrammeista: liikenne on melko samanlaista pienillä prosessimäärillä riippumatta näiden tyypeistä. Eroavuuksia kuitenkin syntyy suurilla prosessimäärillä: 20 lähettävän ja yhden taustaprosessin tapaus vaatii pienemmän purskekoon kuin yhden lähettävän ja 20 taustaprosessin tapaus. Ensimmäisessä tapauksessa yhteyksien liikenne ilmeisesti lomittuu, mikä pienentää yhden yhteyden vaatimaa purskekokoa.

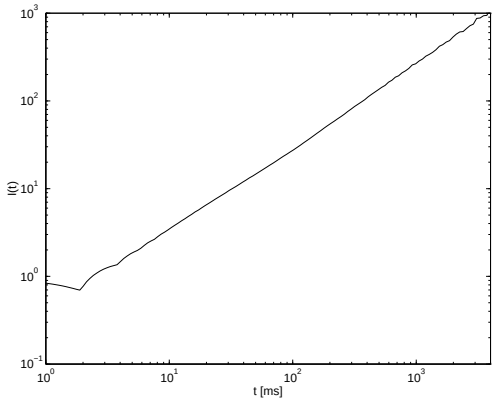
Purskekoon ja keskinopeuden riippuvuutta esittävistä kuvaajista voidaan erottaa selvä taitekohta, jota pienemmillä keskinopeuden arvoilla vaaditava purskekoko kasvaa hyvin nopeasti. Tästä kohdata voidaan määrittää purskeen koko, jota pienempiä on valtaosa yhteyden purskeista.



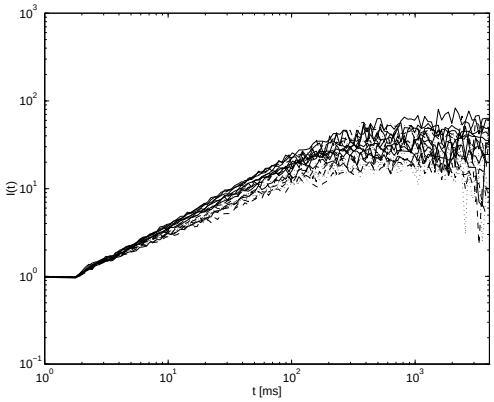
(a) Yksi lähettävä, yksi taustaprosessi.



(b) Kaksi lähettävä prosessia.



(c) Yksi lähettävä prosessi.



(d) 20 lähettävää, 20 taustaprosessia.

Kuva 5.8: Määrän hajontaindeksi.

5.5.2 Määrän hajontaindeksi

Saapumisten lukumäärän hajontaindeksikäyriä (kappale 2.5.2, s. 31) on esitetty kuvissa 5.8(a) – 5.8(d). Osa suurempien aikavälien vaihteluista on ilmeisesti peräisin lyhyestä mittausajasta, vaikka suurin aikaväli t ei ylitäkään viidennestä mittausajasta.

Kehysten normaali väliaika (1,7 ms) näkyy käyrissä pienenä notkahduksena alussa. Aikaisemminkin todettu kahden prosessin tapauksessa liikenteen ominaisuuksien riippumattomuus toisen prosessi luonteesta näkyy kuvista 5.8(a) ja 5.8(b) myös saapumisten lukumäärän hajontaindeksin perusteella. Kahden lähettävän prosessin tapauksessa toisella yhteydellä on kuitenkin selvä indeksin pieneneminen 0,5 – 1 s aikavälillä. Oletettavasti syynä on mittausjakson lyhyys, joka aiheuttaa poikkeavuuksia.

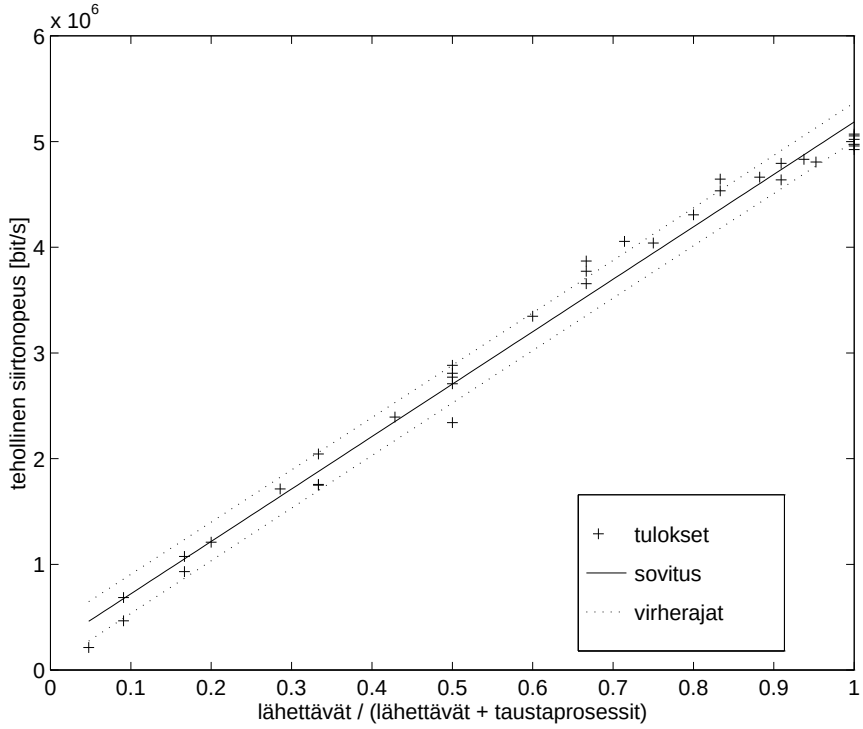
Yhden lähettävän prosessin ilman taustaprosesseja synnyttämän liikenne on selkeästi itsesimilaarista alkaen kehysten väliaikojen mediaanista. Suurilla prosessimäärillä liikenteen vaihtelevuus on suurta ja $I(t)$ näyttäisi tasoittuvan alle sadan, joten liikenne muistuttaisi Markov-moduloitua Poisson-prosessia [Vid96]. Tämä tulos voi kuitenkin olla myös lyhyen mittausjakson aiheuttama vääristymä.

5.6 Suorituskyky

5.6.1 Siirtotehokkuus

Eri prosessityyppien suhteen vaikutusta siirtonopeuteen arvioitiin laskemalla jokaisesta mittauksesta kaikilla yhteyksillä siirretty datamäärä ja laskemalla tästä tehollinen siirtonopeus. Saatua kokonaissiirtonopeutta verrattiin lähettävien prosessien suhteeseen kaikkiin prosesseihin. Osoittautui, että näillä oli selkeä riippuvuus: lineaarikorrelaatio oli 0,99. Yksittäiset tulokset ja pienimmän virheen neliösumman mukaan sovitettu suora 50 % virherajoineen on esitetty kuvassa 5.9.

Aikasarjakuvien (kuva 5.2, s. 59) perusteella näytti, että TCP:n hidas käynnistys (kappale 2.3.3.2, s. 24) heikentäisi siirron suorituskykyä aiheuttamalla viivettä purskeen alussa. Tulosten tarkempi tarkastelu kuitenkin paljasti, että tästä syntyvä tehohäviö



Kuva 5.9: Kokonaissiirtonopeus suhteessa prosessityyppien suhteeseen.

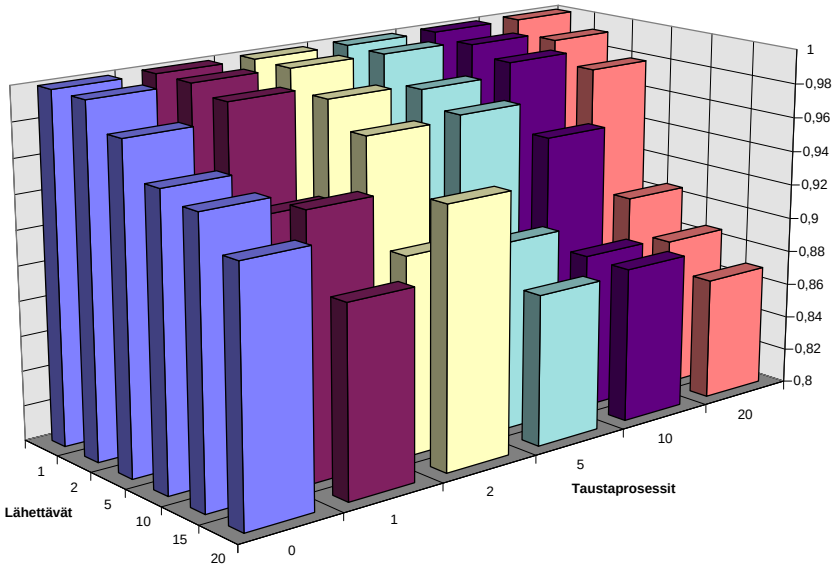
on varsin pieni: 20 lähettävällä prosessilla ilman taustaprosesseja kokonaissiirtonopeus on 97,3 % yhden prosessin tapauksesta. Tämä siitä huolimatta, että 20 lähettävän prosessin tapauksessa hidas käynnistys tapahtui aina kuin prosessi alkoi lähettää. Tehohäviöstä ainakin osa voi olla lisäksi peräisin prosessien vaihtamisen aiheuttamista viiveistä.

5.6.2 Liikenteen reiluus

Liikenteen reiluus eri yhteyksillä i määritellään aikavälillä siirrettyjen tavujen x_i perusteella kaavan 5.1 mukaan [GJK+96].

$$reiluus = \left(\sum x_i\right)^2 / \left(n \times \sum x_i^2\right) \tag{5.1}$$

Eri mittauskertojen tuloksista laskemalla (kuva 5.10) voidaan todeta, että liikenne kaikilla mittauskerroilla on reilua koko mittausajalta (20 s) laskettuna. Liikenteen reiluus vähenee (lineaarikorrelaatio -0,82) prosessien lukumäärän lisääntyessä, mikä selittyy



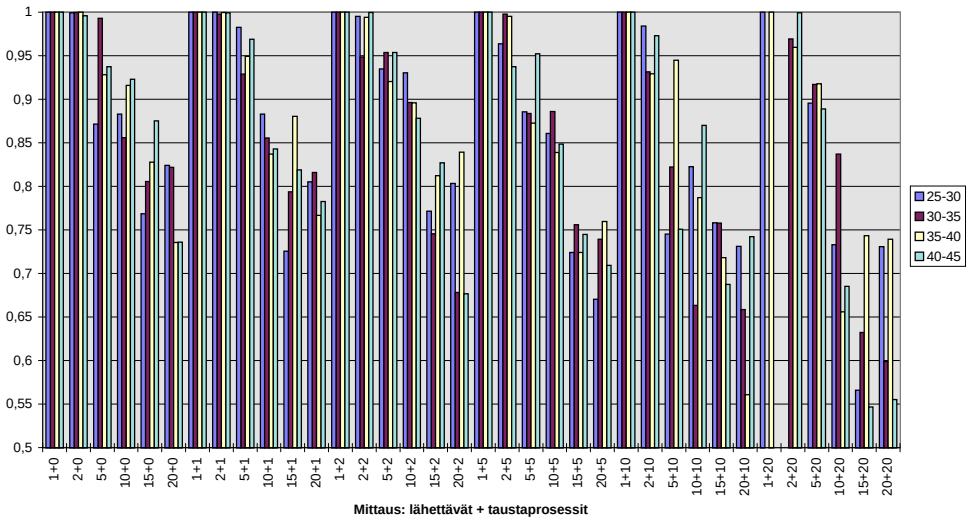
Kuva 5.10: Liikenteen reiluus 20 s ajanjaksolta.

kohtuullisen lyhyellä mittausajalla: tämä ilmiö on selvästi nähtävissä erona 5 sekunnin aikaväleiltä (kuva 5.11), missä reiluus on pienempää kuin samojen mittaustapahtumien 20 sekunnin aikaväleillä.

5.7 Vuorottelun ohjaaminen liikenteen perusteella

Syntyvän liikenteen ominaisuuksiin voi ilmeisesti vaikuttaa ohjaamalla prosessien vuorottelua. Tämä ei kuitenkaan ole aivan yksinkertainen tehtävä, koska tälle mekanismille tulee asettaa seuraavat vaatimukset:

- CPU-intensiivisten prosessien suorituskyky ei saa laskea.
- Tiedonsiirtoprosessien suorituskyky ei saa laskea.
- Mekanismin on oltava kevyt.
- Reiluus prosessien kesken on säilytettävä.



Kuva 5.11: Liikenteen reiluus 5 s ajanjaksoilta.

- Väärinkäyttö on estettävä, esimerkiksi palvelun kielto -hyökkäykset eivät saa helpottua.

Yksinkertainen ”pysäytä prosessi kun se on lähettänyt tietyn määrän” tai käytössä oleviin puskureihin perustuva järjestelmä ei tuo haluttua vaikutusta. Purskeiden väliajat säilyvät suurina, jos lähettäviä prosesseja yksinomaan rangaistaan eikä tätä hyvitetä mitenkään.

Toiminnassa tulee myös huomioida TCP:n vuonohjaus, jonka tehokkaan toimivuuden vuoksi TCP:n liikennettä ei voida mielivaltaisesti muokata. Tämä tuo eteen ongelman prosessien hallinnan sijoittamisesta: eri sijoitusvaihtoehdoilla on erilaiset edut ja haitat.

verkkokortti Verkkokortille on (ATM-verkoissa) sijoitettu liikenteenhallinta, joten täällä on paras tieto tarjolla olevasta verkkokapasiteetista samoin kuin kirjanpito siirretystä liikenteestä. Nykyisissä ATM-verkoissa samalla virtuaaliyhteydellä voi olla kuitenkin useita erillisiä, toisistaan riippumattomia yhteyksiä jopa eri kohdekoneille [Com95, Hei93].

Mekanismi joudutan implementoimaan jokaiselle verkkokortille erikseen, mikä lisää kustannuksia ja tekee toteutuksen laiteriippuvaksi.

verkkokortin ajuri Yleensä verkkokortin ajurillakin on hyvä tietämys verkon tilasta, koska se voi tiedustella sitä kortilta. Verkkokortin ajuri toimii pääprosessorin ohjaamana, joten tätä jouduttaneen käsittelemään keskeytyksistä, mikä taas lisää järjestelmän latenssia.

Eri verkkokortit eroavat palauttamansa tiedon osalta hyvinkin paljon, joten toteutuksesta melkoinen osa tulee olemaan laitteistoriippuvaa. Tietämättömyys yksittäisistä voista haittaa samoin kuin verkkokortin tapauksessa.

IP-kerros Vuotarkkuus on IP-kerroksella jo konekohtaista, mutta tietämys verkon tilasta on heikompaa kuin laiteajurilla. Samaan koneeseen voi olla suuntautuneena useampienkin prosessien liikennettä, joten ei välttämättä tiedetä, mihin prosessiin tulisi vaikuttaa.

IP-kerroksen kautta kulkee useimmiten kaikki verkkoliikenne, joten rinnakkaisia toteutuksia ei vaadita.

TCP-kerros Vuot muodostetaan TCP-kerroksella ja se on myös perinteisesti hoitanut vuonohjausta omien mekanismiensa avulla. Ongelmana on etääntyminen verkkokortilta ts. verkon tilasta ei saada tietoa ennen kuin vahinko on tapahtunut ja segmentti kadonnut. Reitityspäätös tehdään vasta IP-kerroksella, joten yhteyden liikenteenhallinnan kohdentaminen oikeaan verkkoliitäntään mutkistaa asioita, mikäli verkkoliitäntöjä on enemmän kuin yksi.

Ongelmia aiheuttaa myös kirjanpito siitä, mikä yhteys kuuluu millekin prosessille, sillä yhteyden luonut prosessi ei välttämättä ole se, joka sitä käyttää. Tämä pätee luonnollisesti kaikille alemmille kerroksille. TCP-kerroksen toteutus ei myöskään säätele liikennettä muilla siirtokerroksen protokollilla kuten UDP:llä.

liikennöintipistekerros Ylimmällä käyttöjärjestelmän tasolla, systeemikutsurajapinnassa, on luonnollisestikin paras tietämys prosesseista. Sama käyttöjärjestelmäkutsu voi kuitenkin kirjoittaa levylle, toiselle prosessille samassa koneessa tai verkkoon, joten vuon olemassaolonkin tarkistaminen pienentää koneen tehoa.

Tietämys verkon nykyisestä tilasta on tällä tasolla luonnollisesti heikointa.

Toteutusta ei ilmeisesti pystytä rakentamaan yksinomaan yhdelle kerrokselle vaan se vaatii vähintäänkin tukitoimintoja ja kirjanpitoa muilla kerroksilla. Tehtävästä tulee monimutkaisempi ja vaikeammin ylläpidettävä, koska se rikkoo kerrosrakenteen.

Mikään kerros ei ole selkeästi parempi kuin muut, mutta tässä vaiheessa näyttäisi parhaalta vaihtoehdoilta TCP- ja laiteajurikerroksille sijoittuva ratkaisu.

Yhteenveto

Työssä kehitettiin mittausmenetelmiä ja -järjestelmä käyttöjärjestelmän ja verkkoliikenteen vuorovaikutusten mittaamiseen. Kehitetty mittausympäristö saatiin käyttökelpoiseen, käytetystä verkkoarkkitehtuurista riippumattomaan muotoon. Mittauksilla saatuja tuloksia on analysoitu eri tavoin tavoitteena karakterisoida syntyvä liikenne erilaisilla prosessiyhdistelmillä. Erilaisten liikenteen karakterisointimenetelmien käyttö toisealta vahvisti yhdellä tavalla tehtyjä päätelmiä, toisealta kumosi toisia. Tämä osoittaa eri menetelmien käytön tärkeyden.

Tietokoneliikenne on purskeista, vaikkakin yhden häiriöttömän prosessin liikenne on hyvin tasaista. Prosessien lukumäärän kasvu lisäsi odotetusti liikenteen purskeisuutta pidentämällä purskeiden välistä ajanjaksoa. Purskeiden pituus vastaavasti lyheni suurilla prosessimäärillä mutta säilyi kuitenkin suuruusluokaltaan samanlaisena: useimmiten oli vain yhden yhteyden liikennettä 10 ms ajan.

Tätä säännönmukaisuutta kuitenkin sekoitti TCP-yhteyden hidas käynnistys kun yhteys oli ollut hiljaa 0,2 s. TCP-segmenttien lähettämistä viivästettiin hitaan käynnistyksen algoritmin mukaan. Lähettävää prosessia ei kuitenkaan pysäytetty vaan se saattoi jatkaa levylukuja ja kirjoituksia verkkopuskureihin, kunnes se käytti noin 60 kilotavua puskurimuistia. Vastoin aikasarjakuvasta tehtyä intuitiivistä tulkintaa, hidas käynnistys ei heikentänyt juurikaan kokonaissuorituskykyä.

LINUX ei suosi I/O-prosesseja, vaan kohtelee samalla tavalla kuin CPU-intensiivisiä prosesseja. Kokonaissiirtonopeus riippuu ainoastaan lähettävien prosessien suhteesta kokonaismäärään. Rajoittava resurssi, joka oli mittausjärjestelyssä verkon siirtonopeus, oli monissa tapauksissa huomattavalla vajaakäytöllä. LINUX:in vuorottelualgoritmi ei osaa riittävästi lomitella prosessien suoritusta, jotta kaikkia osajärjestelmiä

hyödynnettäisiin täysimääräisesti. Osasyynä voi olla käytetty PC-arkkitehtuurin epäideaalisuudet: esimerkiksi levy-DMA:n käyttö voisi helpottaa prosessien vuorottelua.

Jatkotutkimuksissa tulee testata mitatut ominaisuudet nopeammissa verkoissa. Mittausten suorittaminen nopeammassa verkossa todennäköisesti muuttaa lähettävien prosessien suhdetta, mutta perusilmiö todennäköisesti säilyy ennallaan vuorottelualgoritmin ominaisuuksista johtuen. Mittauksen kesto on saatava pitemmäksi. Lyhyen mittausajan aiheuttamaa vääristymää oli useissa tuloksissa, esimerkiksi saapumisien määrän hajontaindeksissä ja reiluudessa. Erityisen suuri ongelma oli suurilla prosessimäärillä.

Eric Lin mittauksissa [Lin96] yhteyden käynnistyspurskeen mukaanotto vaikutti huomattavasti tuloksiin. Mielenkiintoista olisi tutkia, onko ilmiö poistunut uusimmista TCP-toteutuksista. Mittausten suorittaminen todellisilla sovelluksilla auttaisi kehittämään mittausohjelmaa vastaamaan enemmän todellista sovellusohjelmaa.

Purskeisuutta on myös mahdollista vähentää puskuroimalla lähetettävää dataa käyttöjärjestelmän puskureissa. Suurien puskureiden käyttö on kuitenkin tehotonta kokonaisresurssien kannalta, joten muita ratkaisuja on etsittävä. Liikenteen ominaisuuksiin vaikuttaminen rakentamalla vaikutusmekanismi liikenteestä prosessien vuorotteluun on haaste. Sen sijoittaminen oikealle tasolle vaatii huolellista harkintaa. Onnistuessaan järjestelmä helpottaa tiedonsiirtoverkon liikenteenhallinnan tehtäviä.

Lähteet

[Alm95] Werner Almesberger. High-speed atm networking on low-end computer systems. DI-EPFL Technical Report 95/147, Laboratoire de Réseaux de Communication (LRC), EPFL, CH-1015 Lausanne, Switzerland, August 1995. URL:ftp://lrcftp.epfl.ch/pub/linux/atm/papers/atm_on_lowend.ps.gz.

[ANS96] ANSI/IEEE. Information technology–telecommunications and information exchange between systems–local and metropolitan area networks–lan/man-type specific requirements, part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications and 802.3u-1995 supplement to ieee std 8802-3 : 1996: Media access control (mac) parameters, physical layer medium attachment units, and repeater for 100 mb/s operation, type 100baset (clause 21-30). Iso/iec standard, ISO/IEC, 1996.

[BBD+96] M Beck, H Böhme, M Dziadzka, U Kunitz, R Magnus, and D Verworner. *Linux Kernel Internals*. Addison-Wesley Publishing Company, Inc., 1996.

[BBJ92] D. Borman, R. Braden, and V. Jacobson. TCP extensions for high performance. Request for Comments (Proposed Standard) RFC 1323, Internet Engineering Task Force, May 1992. (Obsoletes RFC1185). URL:ftp://ds.internic.net/rfc/rfc1323.txt.

[BLC95] T. Berners-Lee and D. Connolly. Hypertext markup language - 2.0. Request for Comments (Proposed Standard) RFC 1866, Internet Engineering Task Force, November 1995. URL:ftp://ds.internic.net/rfc/rfc1866.txt.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyck. Hypertext transfer protocol – http/1.0. Request for Comments (Informational) RFC 1945, Internet Engineering Task Force, May 1996. URL:http://www.internic.net/rfc/rfc1945.txt.

[Bra89a] R. Braden. Requirements for internet hosts - application and support. Request for Comments (Standard) STD 3, RFC 1123, Internet Engineering Task Force, October 1989. URL:ftp://ds.internic.net/rfc/rfc1123.txt.

[Bra89b] R. Braden. Requirements for internet hosts - communication layers. Request for Comments (Standard) STD 3, RFC 1122, Internet Engineering Task Force, October 1989. URL:ftp://ds.internic.net/rfc/rfc1122.txt.

[CD96] A. Conta and S. Deering. Internet control message protocol (ICMPv6) for the internet protocol version 6 (ipv6). Request for Comments (Proposed Standard) RFC 1885, Internet Engineering Task Force, January 1996. URL:ftp://ds.internic.net/rfc/rfc1885.txt.

[Cen95] CERT Coordination Center. Ip spoofing attacks and hijacked terminal connections. CERT(sm) Advisory CA-95:1, CERT Coordination Center, January 1995.

[Com95] Technical Committee. Lan emulation over atm version 1.0. Technical Report af-lane-0021.000, The ATM Forum, Jan 1995.

[Cox95] Alan Cox. An implementation of multiprocessor linux. Technical report, Linux community, 1995.

[Cro82] D. Crocker. Standard for the format of ARPA internet text messages. Request for Comments (Standard) STD 11, RFC 822, Internet Engineering Task Force, August 1982. (Obsoletes RFC0733); (Updated by RFC1327, RFC0987). URL:ftp://ds.internic.net/rfc/rfc822.txt.

[CS95] Michael A. Cusumano and Richard W. Selby. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People.* Free Pr, October 1995.

[Dee89] S. Deering. Host extensions for IP multicasting. Request for Comments (Standard) STD 5, RFC 1112, Internet Engineering Task Force, August 1989. (Obsoletes RFC0988). URL:ftp://ds.internic.net/rfc/rfc1112.txt.

[DH96] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. Request for Comments (Proposed Standard) RFC 1883, Internet Engineering Task Force, January 1996. URL:ftp://ds.internic.net/rfc/rfc1883.txt.

[Eri96] J. Eriksson. An experimental encapsulation of ip datagrams on top of atm. RFC 1926, Internet Engineering Task Force, April 1996. URL:ftp://ds.internic.net/rfc/rfc1926.txt.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. Request for Comments (Standards Track) RFC 2068, Internet Engineering Task Force, January 1997. URL:http://www.internic.net/rfc/rfc2068.txt.

[FK96] John H. Fraser and David R. Kaeli. Operating system impact on cache performance. Submitted to HPCA-3, 1996.

[FS96] Bryan Ford and Sai Susarla. Cpu inheritance scheduling. In *Proceedings of the USENIX 2nd Symposium on Operating Systems Design and Implementation*, page 15. USENIX Association, October 1996.

[GGV96] Pawan Goyal, Xingang Guo, and Harrick M. Vin. A hierarchical cpu scheduler for multimedia operating systems. In *Proceedings of the USENIX 2nd Symposium on Operating Systems Design and Implementation*, page 15. USENIX Association, October 1996.

[GJK+96] Rohit Goyal, Raj Jain, Shiv Kalyanaraman, Sonia Fahmy, and Seong-Cheol Kim. Performance of tcp over ubr+. Technical Report 96-1269, ATM Forum, October 1996.

[Hei93] J. Heinanen. Multiprotocol encapsulation over ATM adaptation layer 5. Request for Comments (Proposed Standard) RFC 1483, Internet Engineering Task Force, July 1993. URL:ftp://ds.internic.net/rfc/rfc1483.txt.

[HL86] Harry Heffes and David M. Lucantoni. A markov modulated characterization of paketized voice and data traffic and related statistical multiplexer performance. *IEEE Journal of Selected Areas in Communications*, SAC-4(6):pp. 856 – 868, Sept 1986.

[ine91] *inetd – internet "super-server"*, 4.3 berkeley distribution edition, March 1991. Manual Page.

[Irl93] G. Irlam. Unix file survey – 1993. Technical report, Usenet community trust, November 1993. URL:http://www.base.com/gordoni/ufs93.html.

[ISO86] ISO. Iso 8879:1986 information processing – text and office systems – standard generalized markup language (sgml). ISO standard ISO 8879, ISO / JTC 1 / SC 18, 1986.

[ISO94] ISO. Iso/iec 7498-1:1994 information technology – open systems interconnection – basic reference model: The basic model. ISO standard ISO/IEC 7498-1, ISO / JTC 1 / SC 21, 1994.

[IT93a] ITU-T. B-isdn atm layer cell transfer performance. ITU-T Recommendation I.356, International Telecommunication Union, 1993.

[IT93b] ITU-T. B-isdn service aspects. ITU-T Recommendation I.211, International Telecommunication Union, 1993.

[IT93c] ITU-T. General aspects of quality of service and network performance in digital networks, including isdns. ITU-T Recommendation I.350, International Telecommunication Union, 1993.

[IT94] ITU-T. Traffic control and congestion control in b-isdn. ITU-T Recommendation I.371, International Telecommunication Union, 1994.

[Jac88] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM Conference*, pages 314–329, August 1988.

[Jav96] Java security – denial of service. On WWW, May 1996. URL:http://java.sun.com/sfaq/denialOfService.html.

[JLM94] Van Jacobson, Craig Leres, and Steven McCanne. *tcpdump – dump traffic on a network*, Jun 1994. A Program Manual Page. URL:ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

[Kil97] Kalevi Kilkki. Simple integrated media access (sima). Internet-Draft 00, Nokia Research Center, March 1997. Expire in 20th September 1997. URL:http://www-nrc.nokia.com/sima/.

[Kos96] Kimmo Koski. *Development of Metacomputing Technology*. PhD thesis, Helsinki University of Technology, Jan 1996.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1988.

[KS94] Stephen Keung and Kai-Yeung Siu. Degradation in tcp performance under cell loss. Technical Report 94-0490, ATM FORUM: Traffic Management Subworking Group, Apr 1994.

[LB96] Kevin Lai and Mary Baker. A performance comparison of unix operating systems on the pentium. In *Proceedings of the 1996 USENIX Conference*, Jan 1996.

[Lin96] Eric C. Lin. *The Effects of Scheduling Contention on Workstation Traffic*. Licentiate thesis, Kungliga Tekniska Högskolan, May 1996.

[LST95] Hongqing Angela Li, Kai-Yeung Sunny Sir, and Hong-Yi Henry Tzeng. Ipx and tcp performance over atm networks with cell loss. Technical Report 95-0151, ATM Forum, Feb 1995.

[LTWW94] Will E. Leland, Murad S Taqqu, Walter Willinger, and Dalinel V Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1), 1 1994.

[MBKQ96] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley Publishing Company, Inc., 1996.

[Moc87a] P. Mockapetris. Domain names - concepts and facilities. Request for Comments (Standard) STD 13, RFC 1034, Internet Engineering Task Force, November 1987. (Obsoletes RFC0973); (Updated by RFC1101). URL:ftp://ds.internic.net/rfc/rfc1034.txt.

[Moc87b] P. Mockapetris. Domain names - implementation and specification. Request for Comments (Standard) STD 13, RFC 1035, Internet Engineering Task Force, November 1987. (Obsoletes RFC0973); (Updated by RFC1348). URL:ftp://ds.internic.net/rfc/rfc1035.txt.

[Mog84a] J. Mogul. Broadcasting internet datagrams. Request for Comments (Standard) STD 5, RFC 919, Internet Engineering Task Force, October 1984. URL:ftp://ds.internic.net/rfc/rfc919.txt.

[Mog84b] J. Mogul. Broadcasting internet datagrams in the presence of subnets. Request for Comments (Standard) STD 5, RFC 922, Internet Engineering Task Force, October 1984. URL:ftp://ds.internic.net/rfc/rfc922.txt.

[Mon97] Jussi Mononen. Utopiamaailman porteilla. *MikroPC*, 1997(4):52–60, April 1997.

[Mor85] Robert T. Morris. A weakness in the 4.2bsd unix™ tcp/ip software. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, February 1985. URL:ftp://ftp.research.att.com/dist/internet_security/117.ps.Z.

[MP85] J. Mogul and J. Postel. Internet standard subnetting procedure. Request for Comments (Standard) STD 5, RFC 950, Internet Engineering Task Force, August 1985. URL:ftp://ds.internic.net/rfc/rfc950.txt.

[NGBSP97] Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, and Eric Prud'hommeaux. Initial http/1.1 performance tests. Technical Report NOTE-pipelining-970112, W3 Consortium, Jan 1997.

URL:http://www.w3.org/pub/WWW/Protocols/HTTP/Performance/Pipeline.html.

[Nie93] Jakob Nielsen. *Usability Engineering*. AP Professional, 1993.

[Nie96] Jakob Nielsen. Top ten mistakes in web design. Technical report, SunSoft, May 1996. URL:http://www.sun.com/columns/alertbox/9605.html.

[PD96] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufman Publishers, Inc., 1996.

[Pos80] J. Postel. User datagram protocol. Request for Comments (Standard) STD 6, RFC 768, Internet Engineering Task Force, August 1980. URL:ftp://ds.internic.net/rfc/rfc768.txt.

[Pos81a] J. Postel. Internet control message protocol. Request for Comments (Standard) STD 5, RFC 792, Internet Engineering Task Force, September 1981. (Obsoletes RFC0777). URL:ftp://ds.internic.net/rfc/rfc792.txt.

[Pos81b] J. Postel. Internet protocol. Request for Comments (Standard) RFC 791, Internet Engineering Task Force, September 1981. (Obsoletes RFC0760). URL:ftp://ds.internic.net/rfc/rfc791.txt.

[Pos81c] J. Postel. Transmission control protocol. Request for Comments (Standard) STD 7, RFC 793, Internet Engineering Task Force, September 1981. URL:ftp://ds.internic.net/rfc/rfc793.txt.

[Pos82] J. Postel. Simple mail transfer protocol. Request for Comments (Standard) STD 10, RFC 821, Internet Engineering Task Force, August 1982. (Obsoletes RFC0788). URL:ftp://ds.internic.net/rfc/rfc821.txt.

[PR83a] J. Postel and J. Reynolds. Telnet option specifications. Request for Comments (Standard) STD 8, RFC 855, Internet Engineering Task Force, May 1983. URL:ftp://ds.internic.net/rfc/rfc855.txt.

[PR83b] J. Postel and J. Reynolds. Telnet protocol specification. Request for Comments (Standard) STD 8, RFC 854, Internet Engineering Task Force, May 1983. (Obsoletes RFC0764). URL:ftp://ds.internic.net/rfc/rfc854.txt.

[PR85] J. Postel and J. Reynolds. File transfer protocol. Request for Comments (Standard) STD 9, RFC 959, Internet Engineering Task Force, October 1985. (Obsoletes RFC0765). URL:ftp://ds.internic.net/rfc/rfc959.txt.

[RMV96] James Roberts, Ugo Mocci, and Jorma Virtamo, editors. *Broadband network teletraffic: performance evaluation and design of bradband multiservice networks; final report of action COST 242*. Springer, 1996.

[RP94] J. Reynolds and J. Postel. Assigned numbers. Request for Comments (Standard) STD 2, RFC 1700, Internet Engineering Task Force, October 1994. (Obsoletes RFC1340). URL:ftp://ds.internic.net/rfc/rfc1700.txt.

[SFDC90] M. Schoffstall, M. Fedor, J. Davin, and J. Case. A simple network management protocol (SNMP). Request for Comments (Standard) STD 15, RFC 1157, Internet Engineering Task Force, May 1990. URL:ftp://ds.internic.net/rfc/rfc1157.txt.

[SG94] Abraham Silberschatz and Peter B. Galvin. *Operating System Concepts*. Addison-Wesley Publishing Company, fourth edition, 1994.

[SPG91] Abraham Silberschatz, James L. Peterson, and Peter B. Galvin. *Operating System Concepts*. Addison-Wesley Publishing Company, third edition, 1991.

[Ste97] W. Stevens. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, Internet Engineering Task Force, January 1997. URL:ftp://ds.internic.net/rfc/rfc2001.txt.

[Tan92] Andrew S. Tanenbaum. *Modern Operaing Systems*. Prentice-Hall International, Inc, 1992.

[Vid96] Attila Vidács. Self-similar traffic modeling techniques in atm networks. Master's thesis, Technical University of Budapest, May 1996.

[Wai90] D. Waitzman. A standard for the transmission of IP datagrams on avian carriers. RFC 1149, Internet Engineering Task Force, April 1990. URL:ftp://ds.internic.net/rfc/rfc1149.txt.

[WCS96] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl*. O'Reily & Associates, Inc, 2nd edition, September 1996.

Taulukot

Kuvat